

# NVDLA开源编译器 分析和对比

中科院软件所智能研究中心  
程序语言与编译技术实验室 (PLCT)

邱吉

OSDT2019

# 概览

- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

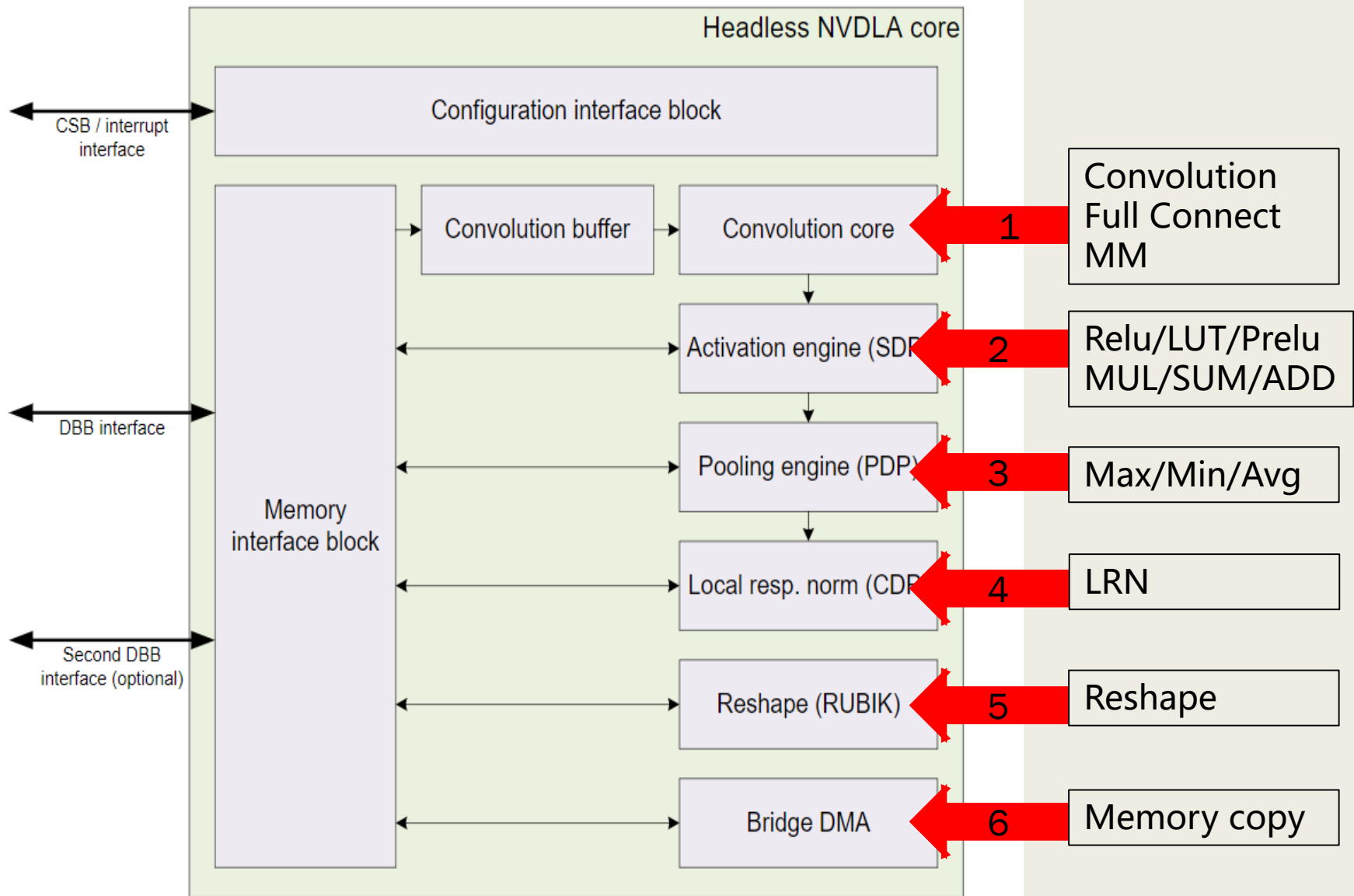
# 概览

- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

# NVDLA是什么

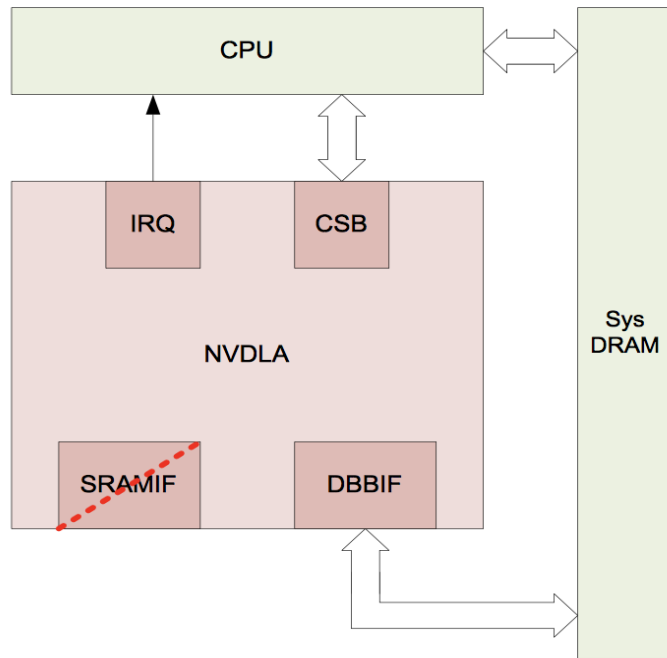
- NVDLA是由NVIDIA公司推出的开源深度学习推理加速器
- 端侧、inference
- 项目状态：并不是太活跃

GitHub repo	Issues@2018 oct	Issues@2019 Oct	Latest commit
<a href="https://github.com/nvdla/sw">https://github.com/nvdla/sw</a>	67	98	2019/9/29
<a href="https://github.com/nvdla/hw">https://github.com/nvdla/hw</a>	113	154	2018/8/12
<a href="https://github.com/nvdla/vp">https://github.com/nvdla/vp</a>	11	23	2018/8/22

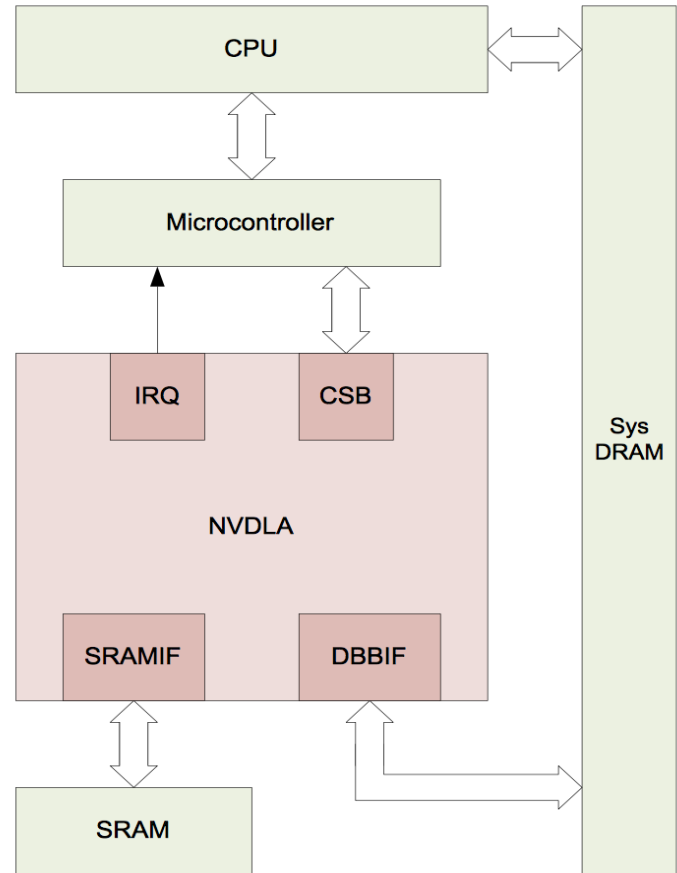


— A standard way to design deep learning inference accelerators.

# System architecture overview

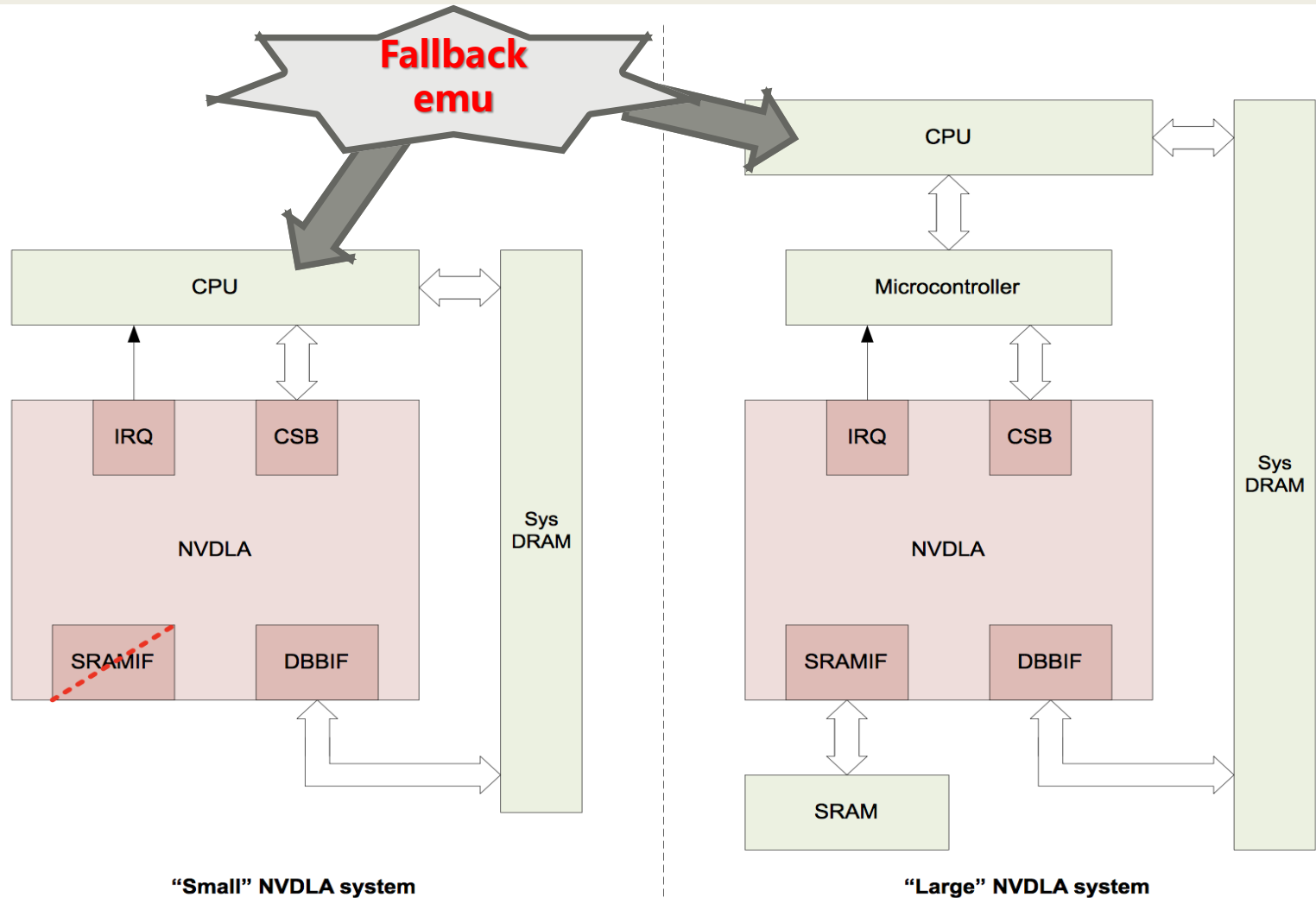


**"Small" NVDLA system**

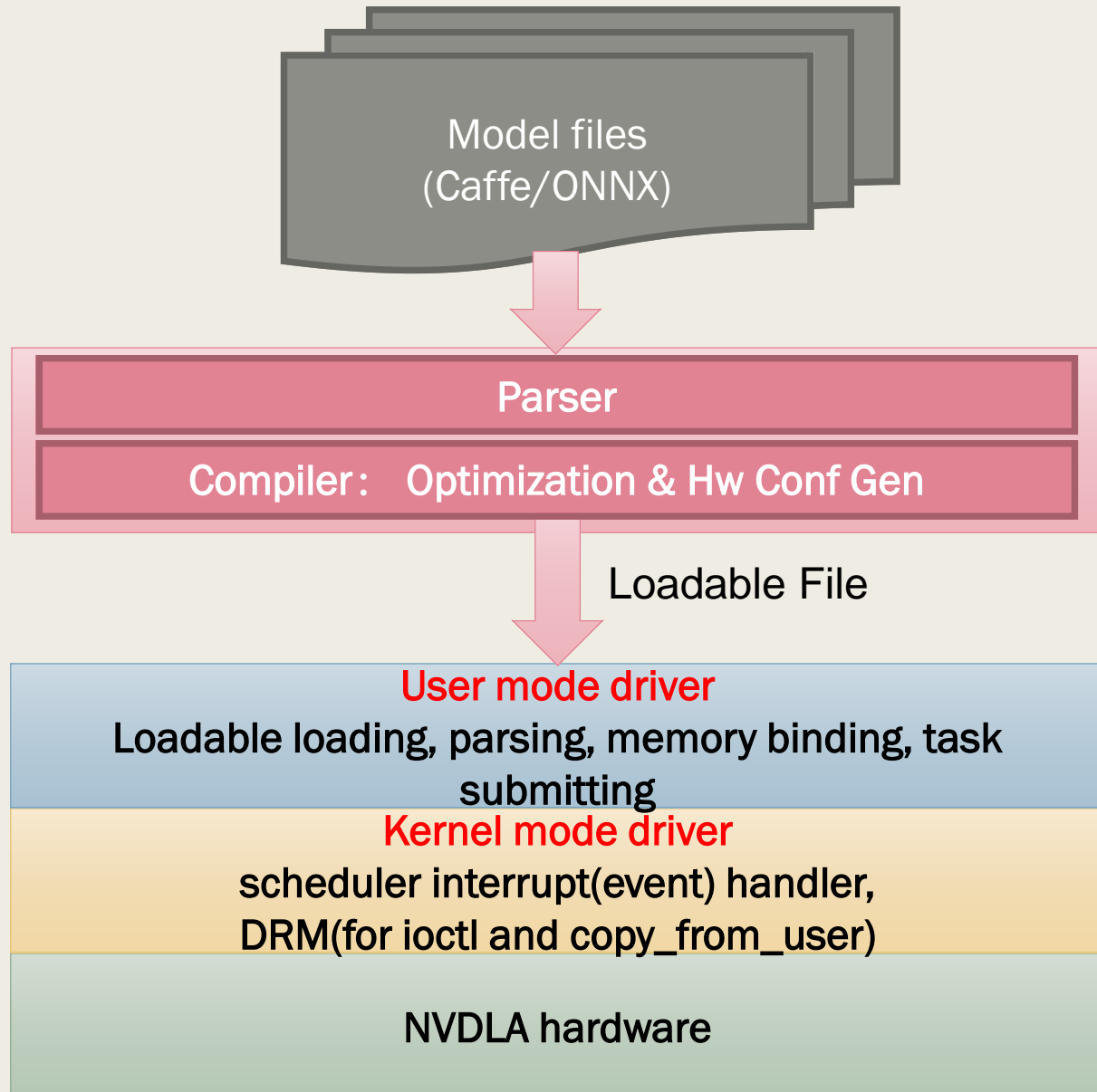


**"Large" NVDLA system**

# System architecture overview



# NVDLA 软件栈





# 概览

- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

# 为什么要探究NVDLA compiler

- NVDLA是典型的Deep Learning Domain Accelerator (DLA) ASIC
- UMD/KMD开源，是一个学习和研究DLA编译器的 Free & Ideal Target
- 到目前，已经有足够多且好的编译器研究对象

# NVDLA编译器概况

作者和名字	开源?	github 活跃区间	有IR?	可优化?	多前端?	多后端?
lcubecorp caffe2fb	是	2018/9- 2018/10	no	no	Caffe	no
NVIDIA  NVDLA compiler	是	2019/8/1 7	自定义 AST	没有IR不方便进行常规编译优化; 可以进行算子fusion和 split ; 可以进行低精度整型优化;	Caffe	no
Skymizer ONNC	是	2018/11- 2019/11	1to1 map to ONNX- IR, 自定义扩展	Yes Pass Manager: Traditional + Target Specific Opt Iteration Compile	ONNX*	yes 可对接LLVM的 CPU、DSP后端, 也可自建DLA后端 NVDLA只是其中之一
ITRI openDLA- FPGA项目	否	2018/10- 2019/10	-	-	-	-

# \* ONNX标准和格式

ONNX是一种针对机器学习所设计的开放式的文件格式，用于存储训练好的模型。它使得不同的人工智能框架（如Pytorch, MXNet）可以采用相同格式存储模型数据并交互。

ONNX的规范及代码主要由微软，亚马逊，Facebook 和 IBM 等公司共同开发，以开放源代码的方式托管在Github上。目前官方支持加载ONNX模型并进行推理的深度学习框架有：Caffe2, PyTorch, MXNet, ML.NET, TensorRT 和 Microsoft CNTK等

# NVDLA编译器概况

作者和名字	开源?	github 活跃区间	有IR?	可优化?	多前端?	多后端?
Icubecorp caffe2fb	是	2018/9- 2018/10	no	no	Caffe	no
NVIDIA NVDLA compiler	是	2019/8/1 7	自定义 AST	没有IR不方便进行常规编译优化; 可以进行算子fusion和split ; 可以进行低精度整型	Caffe	no
Skymizer ONNC	是	2018/11- 2019/11	1to1 map to ONNX- IR, 自定义扩展	Yes Pass Manager: Traditional + Target Specific Opt Iteration Compile	yes	可对接LLVM的CPU、DSP后端, 也可自建DLA后端 NVDLA只是其中之一
ITRI openDLA- FPGA项目	否	2018/10- 2019/10	-	-	-	-

分析  
对象

# 概览

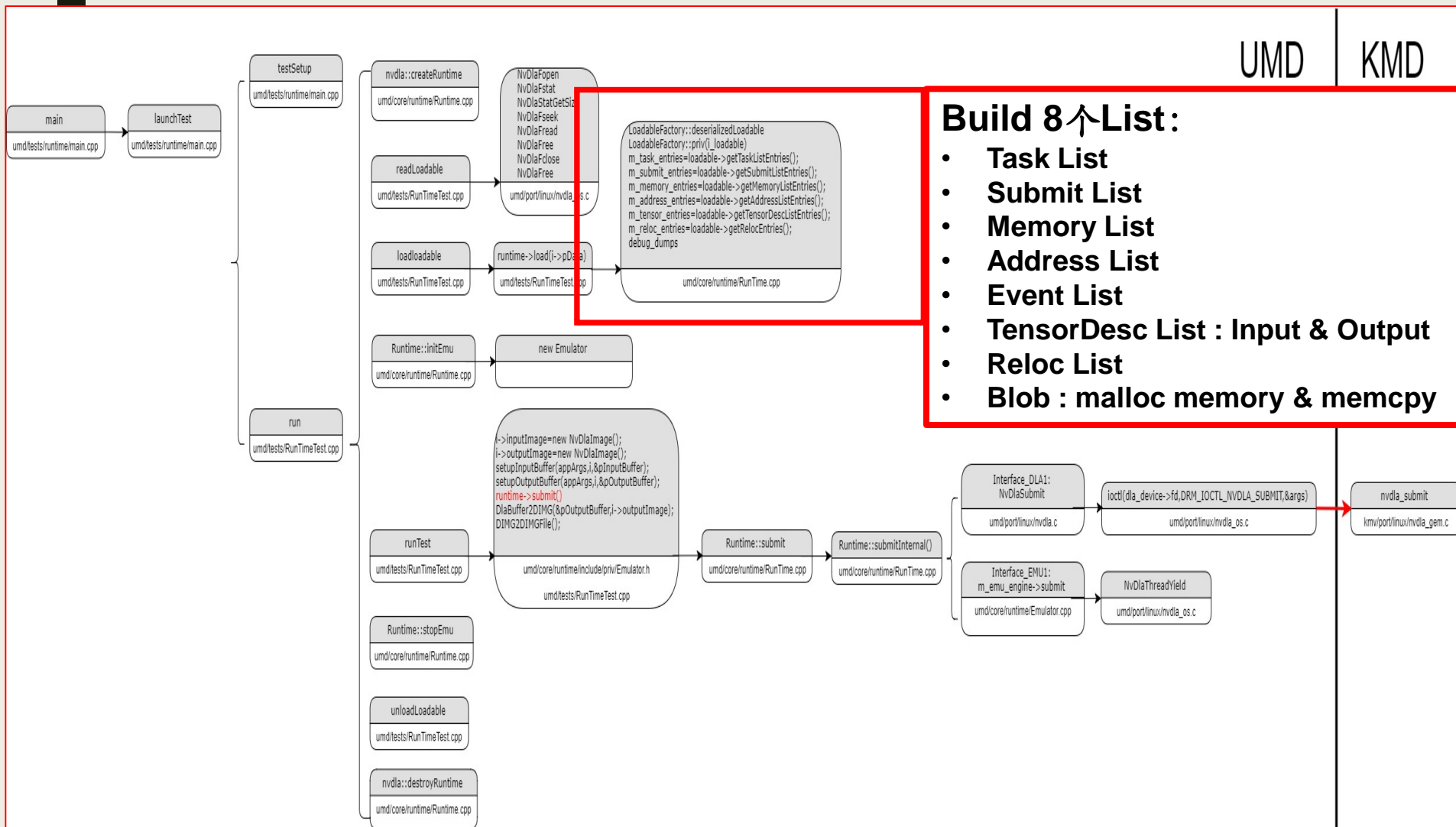
- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

# ISA-SPEC: Compiler vs KMD

- 规定kmd与compiler之间的任务描述协议
  - 与NVDLA之间: **"dla\_interface.h"** , 描述NVDLA的6个功能单元所能提供的所有配置和功能, 在CodeGen阶段, 按照这个协议来产生硬件配置, 超出这个范围的功能将不能在NVDLA上执行, compiler要完全follow这个协议, **但并不是真正的寄存器级别的接口** (寄存器级别接口在kmd/firmware/\*)
  - 与fallback的CPU之间: **"emu\_interface.h"** , 规定了在NVDLA硬件处理能力之外的运算, fallback到主控CPU上时的协议, compiler可以修改这个协议, 添加所需的额外接口, 但需要重新build和部署UMD firmware

# ABI: Compiler vs UMD

Loadable文件格式 “**NVDLALoadable.h**” :  
类似于Application Binary Interface + ELF文件格式的约定





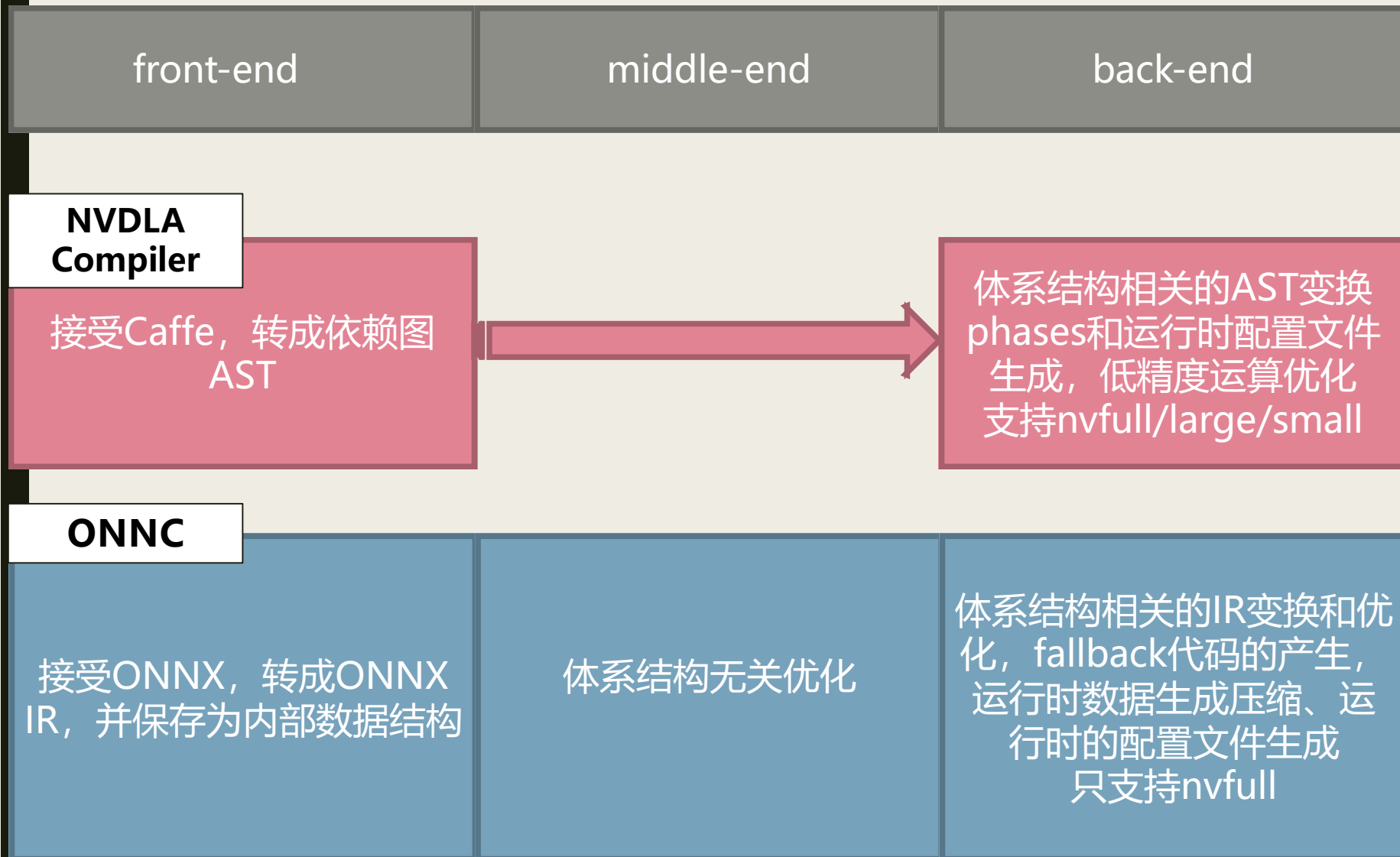
# 概览

- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

# DLA编译器在流程上和传统编译器的对比

	front-end	middle-end	back-end
<b>传统编译器</b>	接受高级语言（C、C++）， parser并转成IR或AST	体系结构无关的 IR变换和优化	Lowering 体系结构相关的 IR变换和优化，汇编指令生成
<b>DLA编译器</b>	接受DSL（model） parser并转成IR或其他表示 形式：AST/DFG	体系结构无关的 IR变换和优化	体系结构相关的IR变换和优化， fallback代码的产生， 运行时数据生成和压缩、 低精度运算优化，二进制指令 代码生成/运行时的配置 文件生成

# NVDLA Compiler和ONNC的对比



# 前端比较

## ■ NVDLA Compiler前端

- Caffe Parser
- 依赖于特定版本的  
protobuf 库, 2.6.1

## ■ 如何扩展

- 参考ParsetTest.cpp的  
parseCaffeNetwork函数来新增其他格式的  
model的parser
- 或利用模型转换工具,  
转成Caffe格式

## ■ ONNC前端

- ONNX Parser
- GOOGLE\_PROTOBUF  
VERIFY\_VERSION宏保  
证库兼容性

## ■ 如何扩展

- 设计理念是通过ONNX  
模型转换器来work, 不  
进行编译器自身代码前  
端的扩展

# 中端比较

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>■ NVDLA Compiler没有严格意义上的中端<ul style="list-style-type: none"><li>- 直接从Parser得到的AST DFG经过一系列phase变换, 发射hw的任务描述符</li><li>- 如何扩展: Parser后加入新的处理phase</li></ul></li></ul> | <ul style="list-style-type: none"><li>■ ONNC中端<ul style="list-style-type: none"><li>- Parser后生成了类似LLVM的数据结构, 并提供访问的Iterator<ul style="list-style-type: none"><li>■ Module -&gt; Function</li><li>■ ComputationGraph -&gt; BB</li><li>■ ComputationOperator -&gt; Inst</li><li>■ Value/Use -&gt; Value/Use</li></ul></li><li>- 类似LLVM的PassManager基础设施<ul style="list-style-type: none"><li>■ doInitialization</li><li>■ runOnModule, Tensor, Compute, Region</li><li>■ doFinalization</li></ul></li><li>- 支持用户扩展ONNC IR, 并添加CodeEmitVisitor</li></ul></li></ul> |
|--|--|

# 后端比较

## ■ NVDLA Compiler支持full/large/small

### ■ 算子支持

- ActivationOp
- BNOp
- BDMAGroupOp
- BDMASingleOp
- BiasOp
- CDPLRNOp
- ConcatOp
- ConvolutionOp
- FullConnectOp
- DeConvOp
- ScaleOP
- SDPEltWiseOp
- SDPSuperOp
- RubikOp
- SplitOp
- SoftMaxOp

### ■ Quantization支持

## ■ ONNC只支持full

### ■ 算子支持:

- AvgPool, MaxPool
- Add
- Concat
- Conv: 不支持Winograd conv
- Constant Init
- Input、Output
- LRN
- Mul
- Shuffle
- Relu
- Reshape
- Softmax: EMU
- Split
- Sum
- Transpose

# 后端比较

- NVDLA Compiler支持full/large/small

- 算子支持

- ActivationOP
- BNOp
- BDMAGroupOp
- BDMASingleOp
- BiasOp
- CL
- Conc
- Conv
- Full
- D
- ScaleOP
- SDPEltWiseOp
- SDPSuperOp
- RubikOp
- SplitOp
- SoftMaxOp

- Quantization支持

- ONNC只支持full

- 算子支持:

- AvgPool, MaxPool
- Add
- Concat
- Conv: 不支持Winograd conv
- Constant Init
- Input、Output
- LRN
- Mul
- Shuffle
- Relu
- Reshape
- Softmax: EMU
- Split
- Sum
- Transpose

100%硬件覆盖  
可作为类似  
LLVM种  
InstrInfo.td的  
参考文件

# Optimization对比

NVDLA Compiler	ONNC
<p>基本上都是fuse类型优化:</p> <p><code>fuse_add_bias_into_conv</code> <code>fuse_bn_into_conv</code> <code>fuse_scale_add_bias_into_conv</code> <code>fuse_scale_into_conv</code> <code>fuse_scale_bn_add_bias_into_conv</code></p> <p>+</p> <p>Quantization</p>	<p><code>fuse_add_bias_into_conv</code> <code>fuse_bn_into_conv</code> <code>fuse_consecutive_squeezes</code> <code>fuse_consecutive_transposes</code> <code>fuse_transpose_into_gemm</code> + dead code elimination constant propagation for difference shapes <code>extract_constant_to_initializer</code> <code>eliminate_identity</code> <code>eliminate_nop_pad</code> <code>eliminate_nop_transpose</code> <code>eliminate_unused_initializer</code></p>



# 关键目录说明-NVDLA Compiler

## Shell

sw/umd/apps/  
compiler/

main.cpp  
ParseTest.cpp  
CompilerTest.cpp  
GenerateTest.cpp

## Kernel

sw/umd/core/  
compiler/

caffe

**engine-ast**

include

\*.cpp:  
framework

# 关键目录说明-ONNC

- ONNC: 类似LLVM
  - lib
    - Target: 后端
    - IR: ONNX IR
    - CodeGen : 生存期分析, 内存操作数alloc&set
    - Transform: 计算图的变换和优化passes
    - Core: PassManager, AnalysisResolver计算pass之间的依赖
    - Runtime: 解释器的框架
    - Option/Support/ADT...
  - Tools
    - onnc: 编译器框架文件
    - onnc-jit: 及时编译器
    - onni: 解释器, X86
    - readonnx: 类似readelf
    - unittest: 单元测试

# 关键目录说明-ONNC-NVDLA

- ONNC/lib/target/NvDla
  - \*.cpp: backend backbone files
  - TargetInfo
  - CodeEmitVisitor
  - Config: nvfull
  - Computer: 自定义扩展的算子（非ONNX标准）

# 文档和社区支持对比

DLA编译器的快速上手和学习

## ■ NVDLA Compiler

- almost no documentation
- no git history
- github issues讨论和回复：不活跃
- 学习靠生读代码和零星注释

NVDLA编译器的产品化开发和进阶  
need both

## ■ ONNC:

- tutorial  
<https://github.com/ONNC/onnc-tutorial>
- docker support for build and NVDLA vp run
- 类LLVM的框架结构便于理解
- github issues讨论和回复：也不活跃
- 学习靠：过往编译器的开发经验+tutorial文档

# 概览

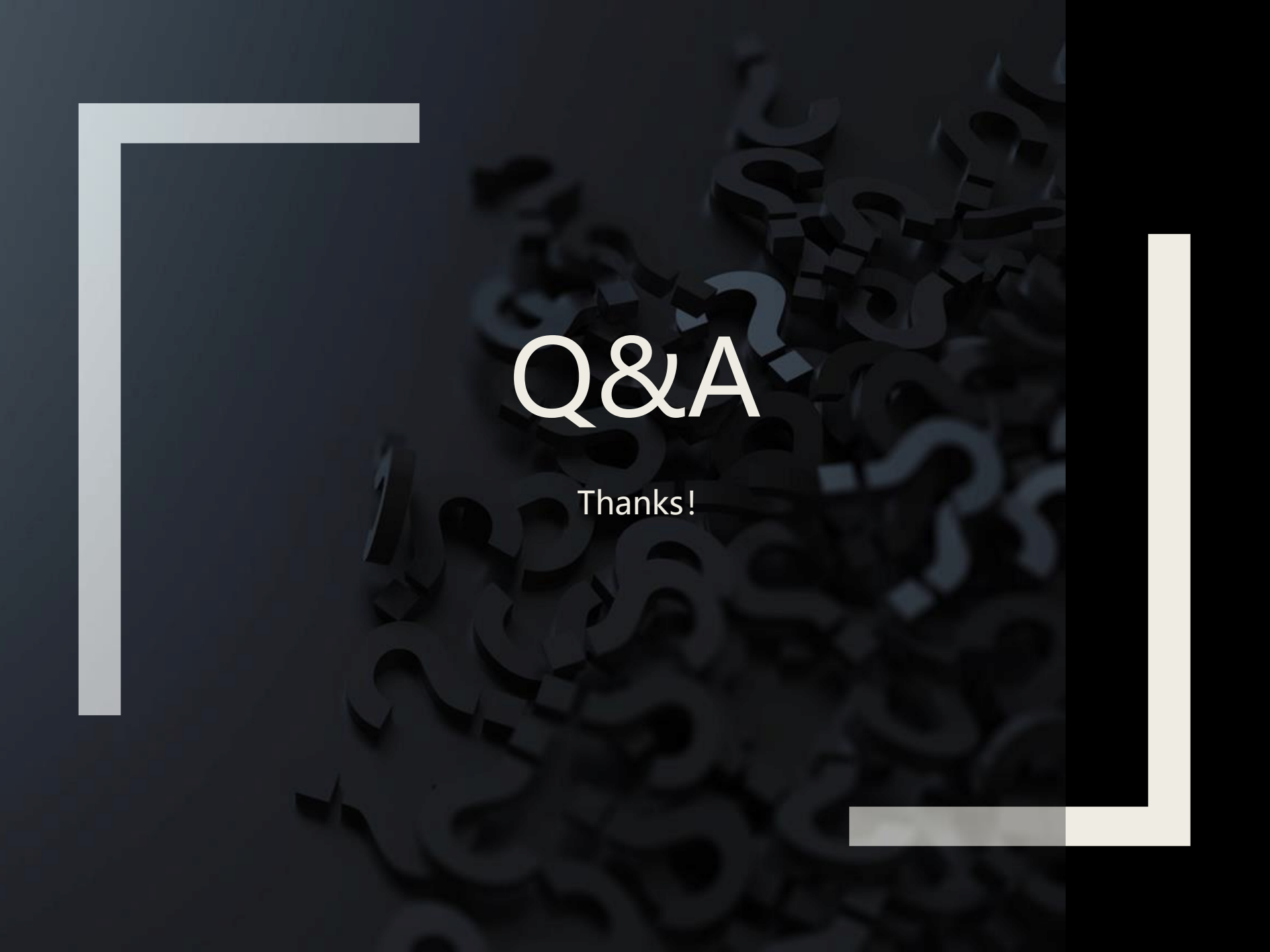
- NVDLA简介
  - 是什么
  - 软件栈结构
- NVDLA编译器概览
- 写编译器的先决条件：软硬件接口规范
  - ISA Spec
  - ABI
- NVDLA Compiler和ONNC的分析和对比
- DLA编译器实现的思考

# DLA编译器实现中的思考

- 前端是否要支持多种格式？或者依赖于某个格式标准和转换器？
- 是否需要IR？ Yes！ 需要什么样的IR？
  - Operator granularity： Processor ISA vs ASIC ISA
  - 是否便于分析、变换和优化
    - SSA formed IR
    - Def-Use Chain Analysis
- 与传统编译器不同的地方：
  - Memory Alloc/Spill/Split
  - Operator Assign/Spill/Split
  - Cost depends on
    - Opcode
    - Operand
    - Memory hierarchy
- 机器模型如何描述
  - 指令集形式
  - 寄存器配置形式
    - NVDLA paradigm
      - UMD暴露功能，但不暴露硬件细节
      - 开放UMD，闭源KMD，对二次开发适配提供firmware和header file

# Reference

- [http://nvdla.org/sw/compilation\\_tool.html#compiler-library](http://nvdla.org/sw/compilation_tool.html#compiler-library)
- <https://github.com/nvdla/sw>
- <https://github.com/onnc>
- <https://github.com/onnx/onnx>
- [https://github.com/icubecorp/nvdla\\_compiler](https://github.com/icubecorp/nvdla_compiler)
- <https://github.com/ITRI-ICLR/OpenDLA-FPGA>

The background is a dark gray gradient. It is filled with numerous 3D question marks of varying sizes and orientations, creating a textured effect. On the left side, there is a large, light gray L-shaped bracket. On the right side, there is a similar L-shaped bracket, but it is white and positioned slightly lower than the one on the left.

# Q&A

Thanks!