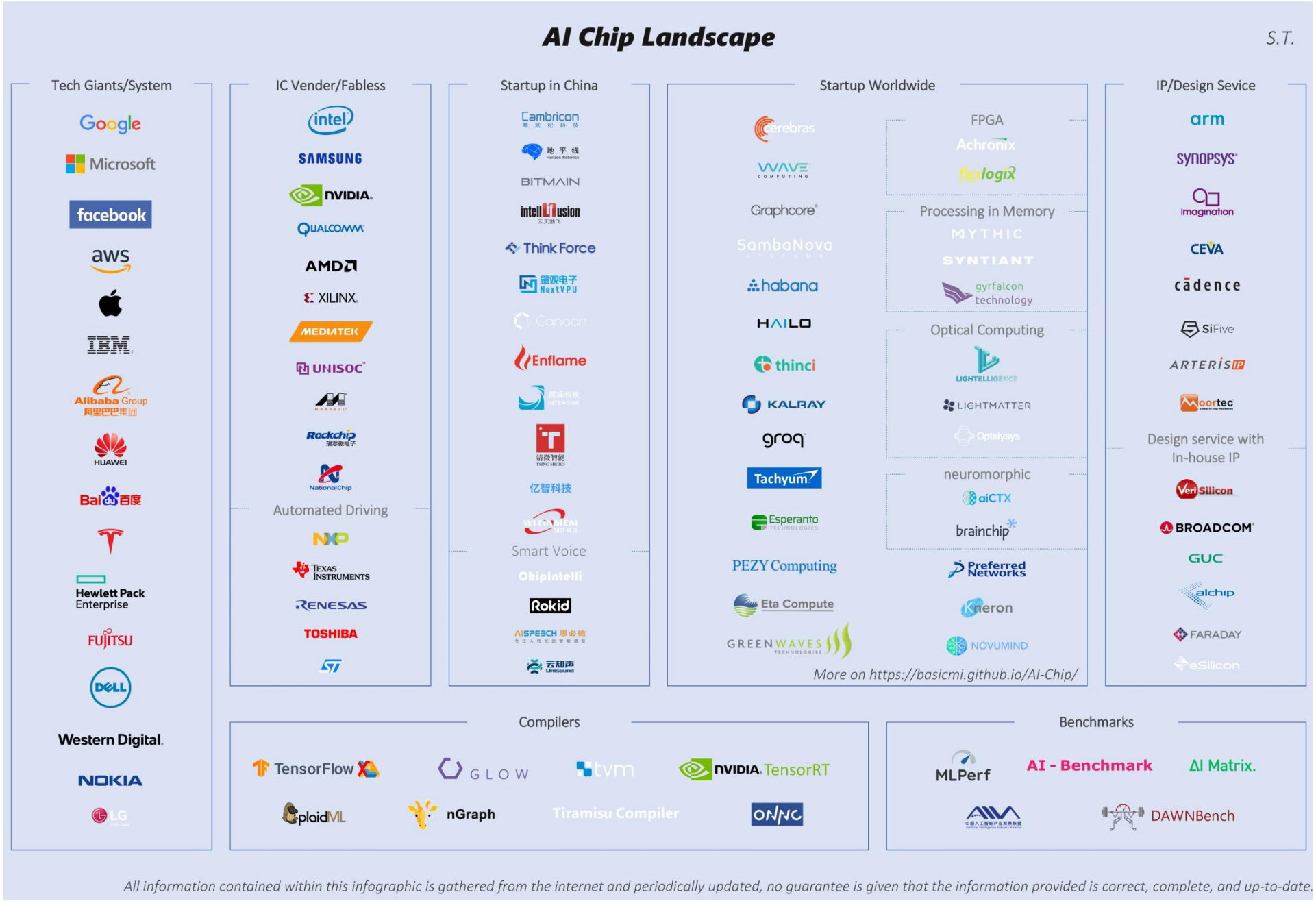
Open AI Toolchain for Edge Inference

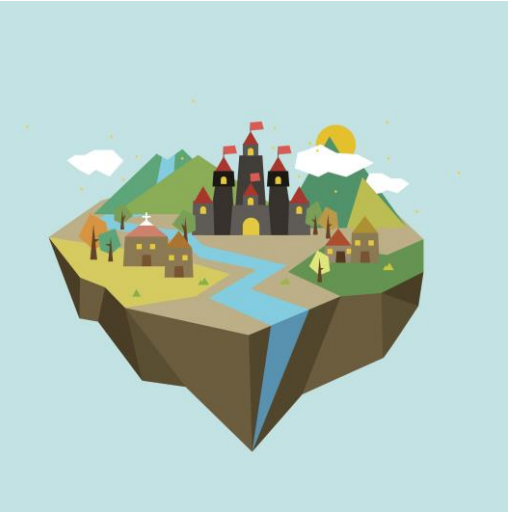
人工智能系统与集成电路研讨会

2019年8月18-19日



https://semiwiki.com/artificial-intelligence/273390-ai-chip-landscape-and-observations/

Mission and Vision



- Provide intelligent compiler for AI-On-Chip
  - Intelligent AI compiler for Deep Learning Accelerator (DLA)
  - Intelligent AI compiler for Heterogeneous Computing
- Support architecture exploration across full design spectrum
  - The next wave of innovation is happening at hardware-software Interface
  - Offer novel approach in IP-EDA ecosystem to accelerate innovation and time-to-market
  - Optimize the AI performance for off-the-shelf SoCs, FPGAs and custom silicon
- In-depth compiler technical experience
  - Based on open source compiler, ONNC (Open Neural Network Compiler), connecting ONNX (Open Neural Network Exchange) to AI-On-Chip
  - More ONNC information is available on website (https://onnc.ai/) and GitHub - (https://github.com/ONNC/onnc)

ONNX Model for Interfacing with Edge Inference Device

https://azure.microsoft.com/en-in/blog/onnx-runtime-for-inferencing-machine-learning-models-now-in-preview/

ONNX – Open Neural Network Exchange Format

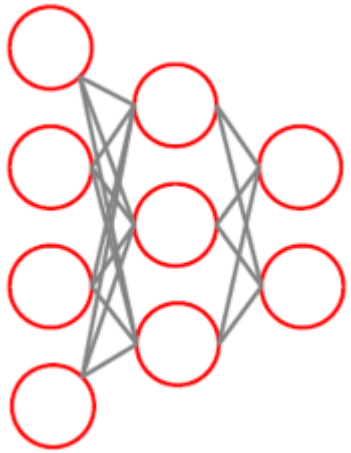
- ONNX is a open format to represent deep learning models
  - AI developers can more easily move models between state-of-the-art tools and choose the combination that is best for them
  - ONNX models are currently supported in Caffe2, Microsoft Cognitive Toolkit, MXNet, and PyTorch, and there are connectors for many other common frameworks and libraries.
  - ONNX is developed and supported by a community of partners

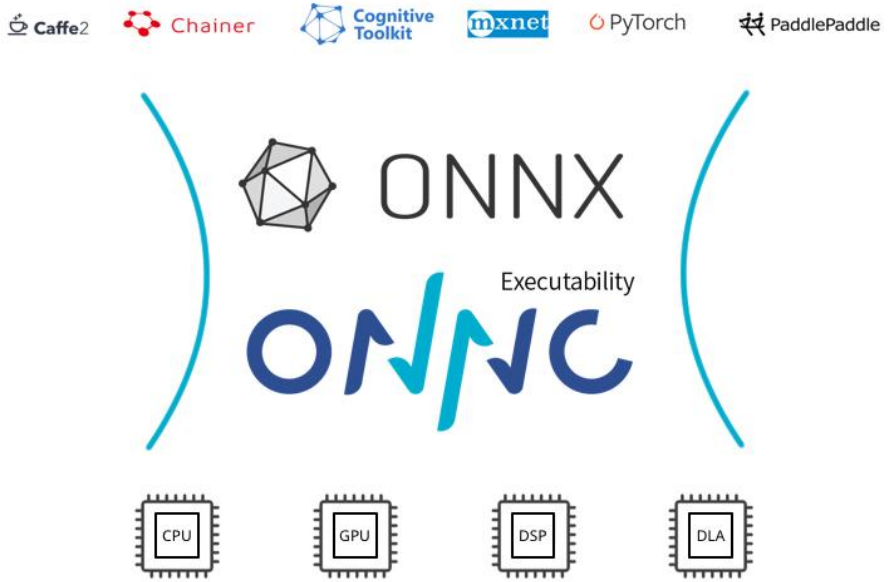
Reference: https://onnx.ai/

ONNC

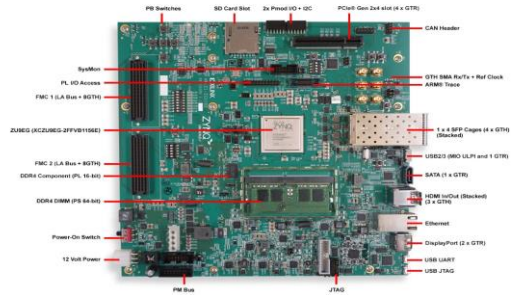
ONNC – Open Neural Network Compiler

- ONNC (Open Neural Network Compiler)
  - Collection of compiler and AI toolchains targeted on ONNX-based DLA
  - ONNC transforms ONNX models into executables for DLA
  - First open source in July 2018, the latest was released in March 2019



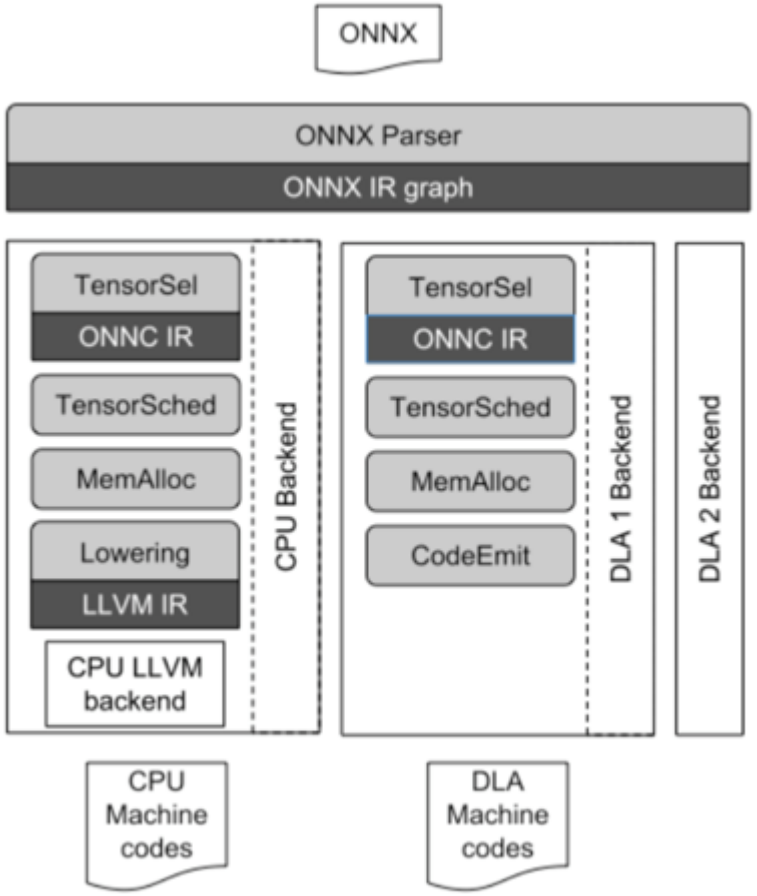


https://onnc.ai/#getting-started



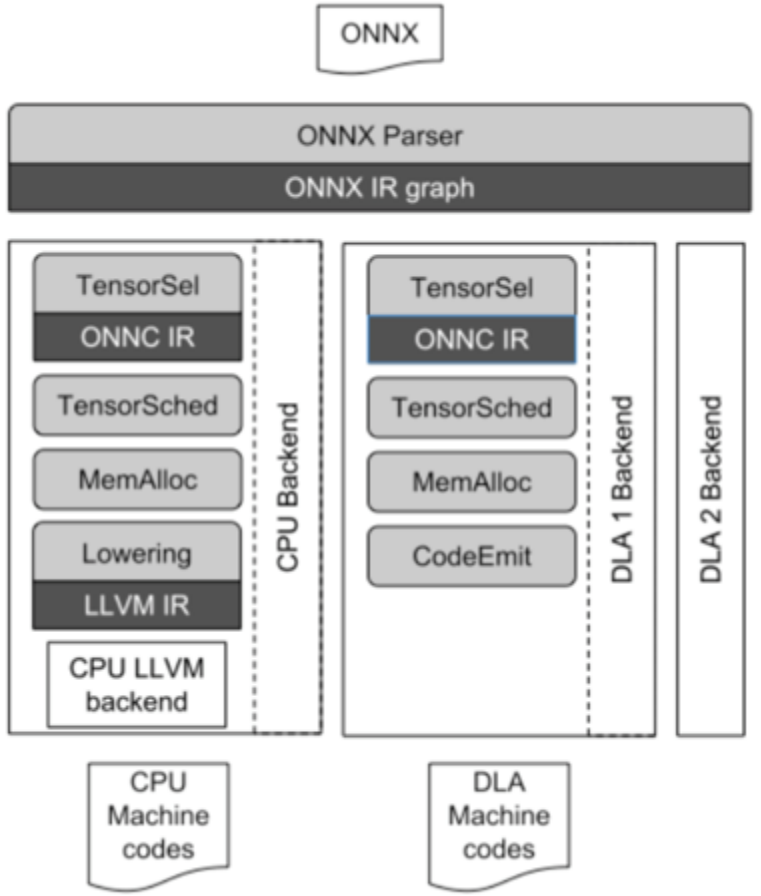
Compiler Designed for Heterogeneous Architecture

target	heterogeneous architecture system (HSA)	single architecture
physical feature depends on	operand	opcode



ONNC Compiler Architecture

- ONNC software stacks illustrates the functional blocks from importing an ONNX computation graph model to emitting corresponding hardware binaries
- ONNC paves another fast track for proprietary DLAs to execute ONNX models by defining ONNC IR, an intermediate representation (IR) that has one-to-one mapping to the ONNX IR
- Two other popular compilation frameworks in deep learning systems, TVM and Glow, built their software stacks on top of the LLVM backend
- The intermediate representations of LLVM have a finer granularity than ONNC IRs while mapping to hardware operators

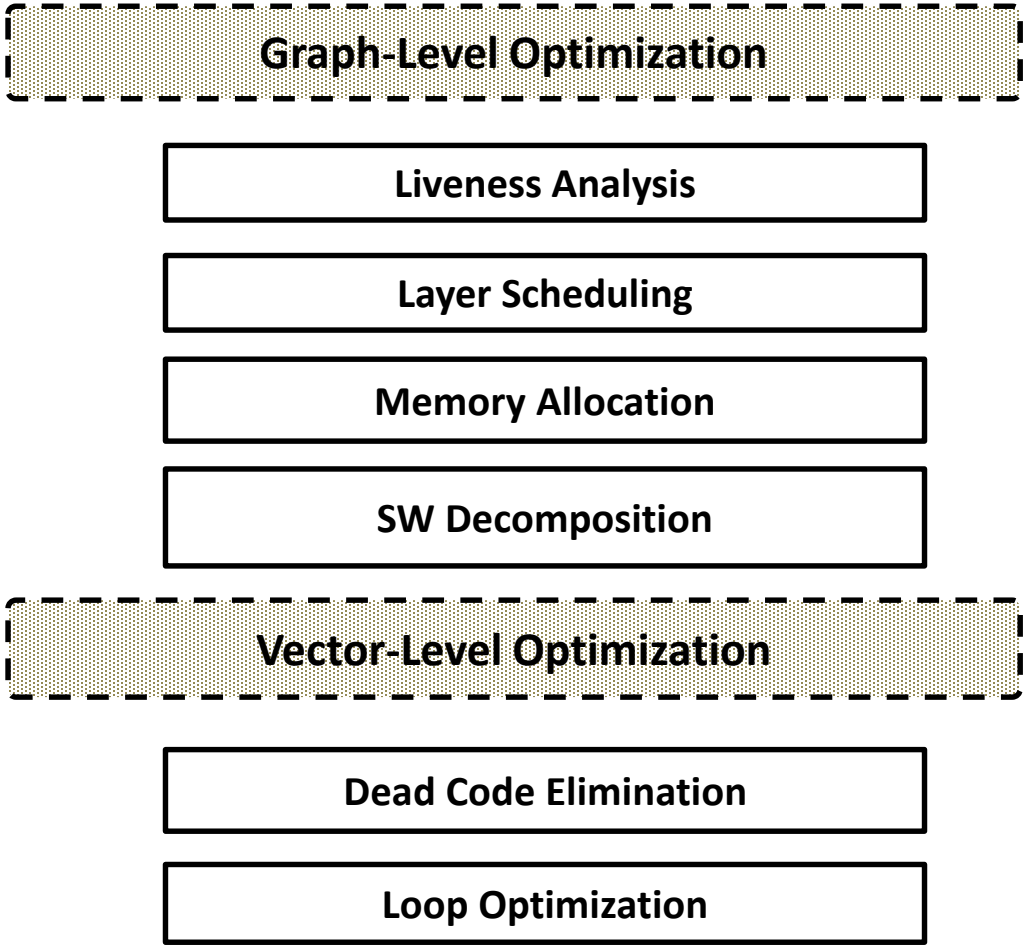


ONNC Compiler Advantages

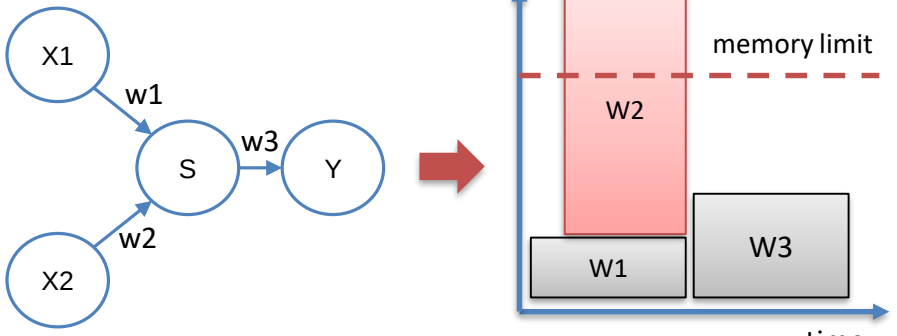
- ONNC IR and Extension
  - The ONNC IR has defined a set of common operators among which 116 IRs respectively correspond to 116 ONNX operators
  - To create new ONNC IRs, users may refer to the ONNC IR Extension Guide
- Pass Manager
  - ONNC's pass manager supports automatic scheduling based on the dependency defined by the pass designer
  - If a pass fails to achieve an optimization goal, it can opt to return a retry request and then pass manager will re-schedule the retry pass as well as all its dependent passes
- Vanilla Backend
  - ONNC provides a Vanilla backend as a template to ease the development of a new DLA backend. New IRs or new passes might be required in porting to a new target

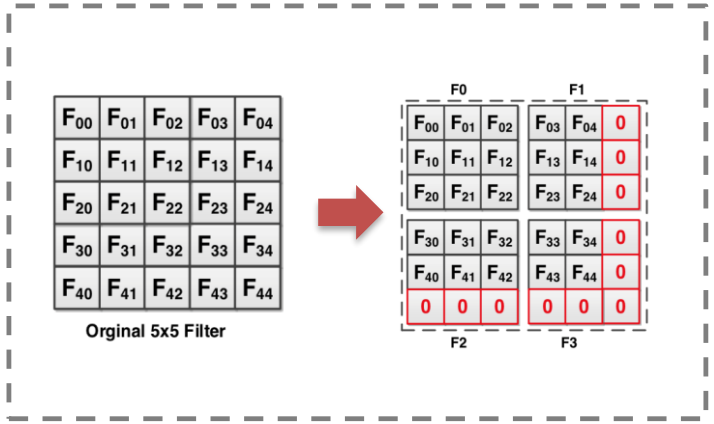
ONNC Optimization Flows

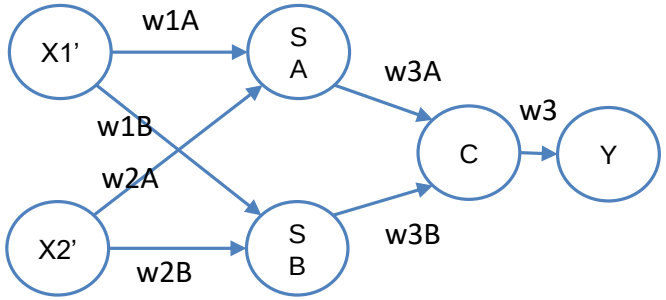
- There are two kinds of optimization algorithms for neural network
  - Graph-level optimization
  - Vector-level optimization
- Graph-level optimization handles with matrices
  - Separate a matrix into pieces
  - Merge several matrices into big one
  - Set the order of matrix multiplications
- Vector-level optimization handles with vectors
  - Reorder the cross products of vectors

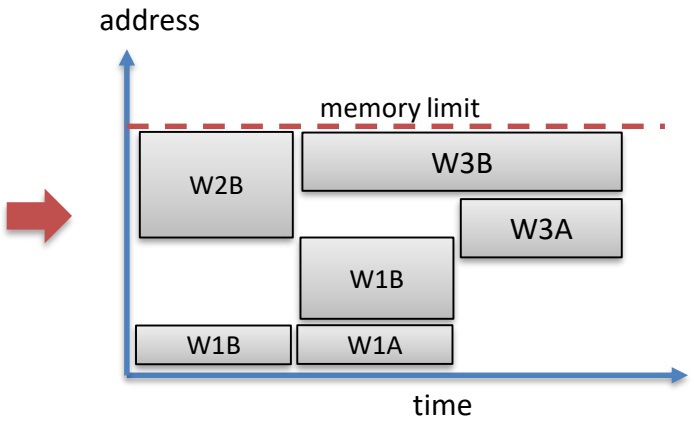


Layer Splitting - Handle the memory limit



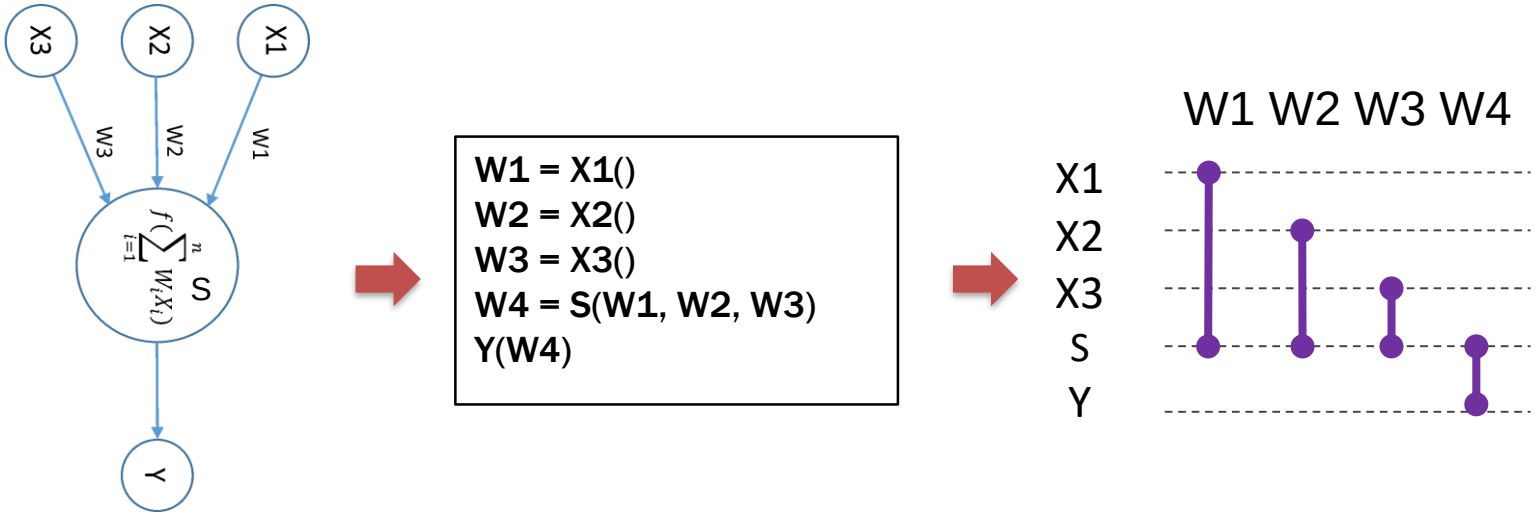






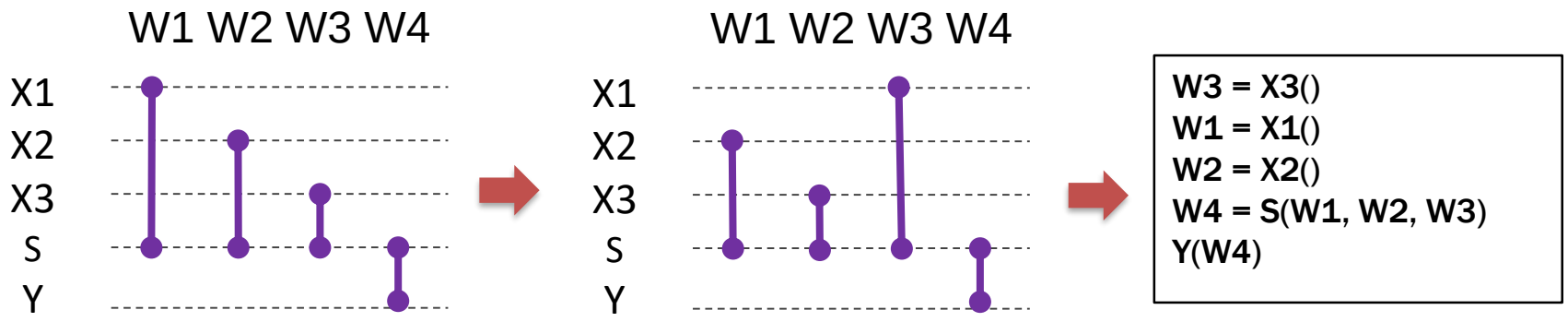
Liveness Analysis of Tensors

- Find out the live range of every tensor
- Leverage use-define chain of ONNX
- By the help of simple liveness analysis, we can reuse local memory and eliminate ½ memory consumption with greedy allocation



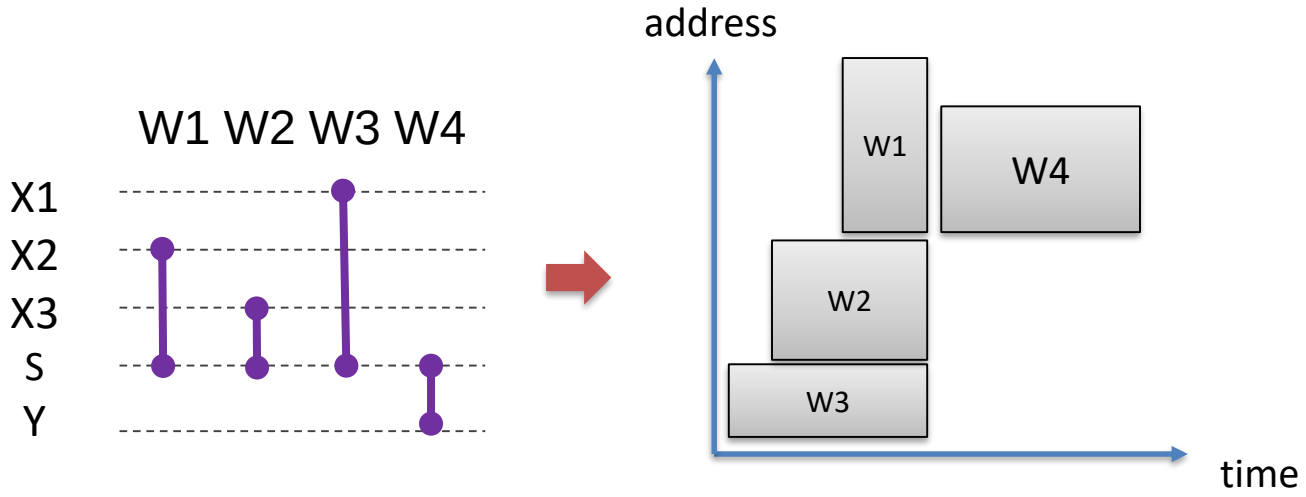
Layer Scheduling

- If size W2 > W1 > W3, then we can reorder X1 X2 X3 to reduce the memory consumption



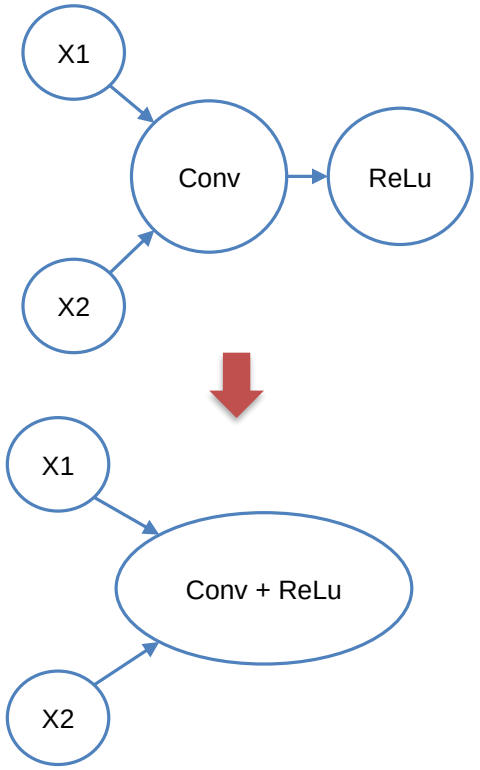
Memory Allocation

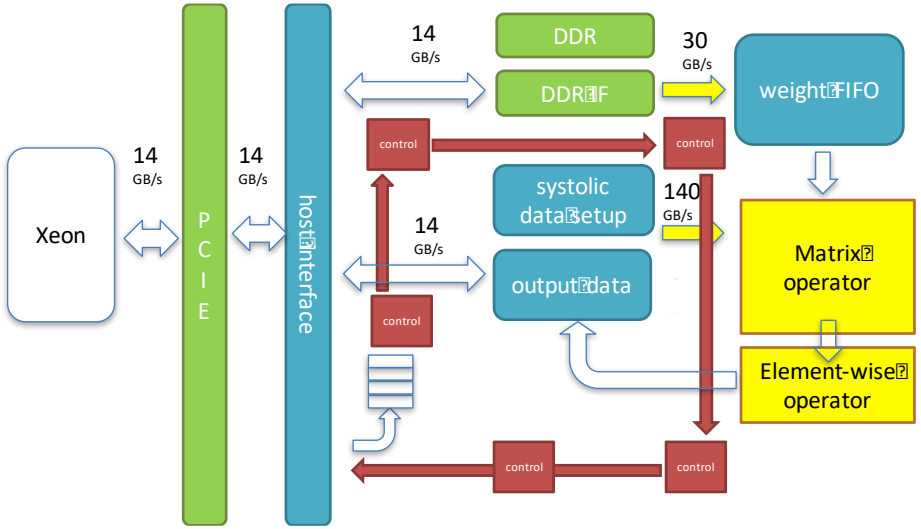
- Memory allocation is use to allocate memory for each layer
- Layer Scheduling affects the results of memory allocation



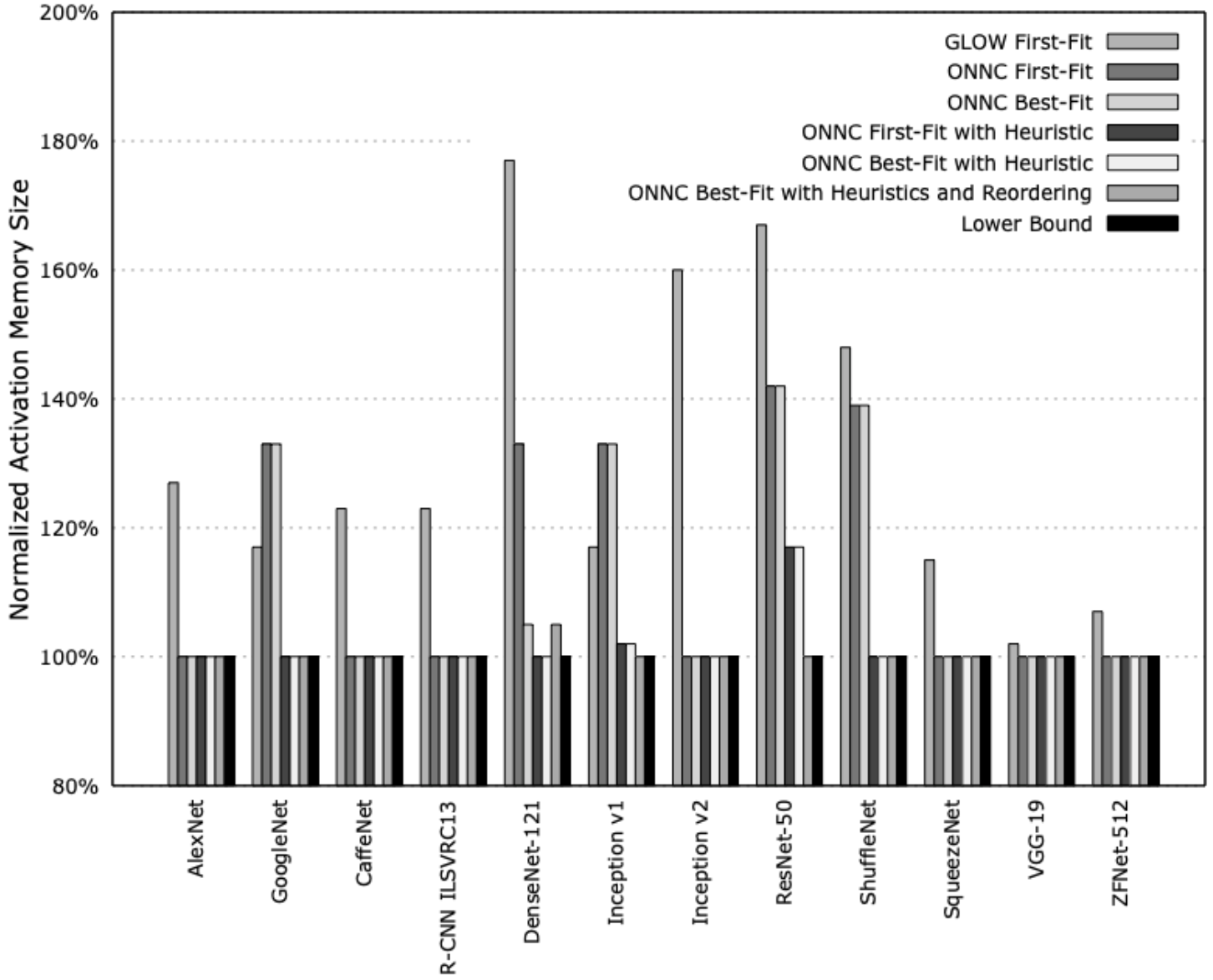
Layer Fusion

- Weight stationary and output stationary architectures usually have dedicated element-wise function unit.
- If we can leverage the element-wise function unit, then we can save data movement from outside to the inside core

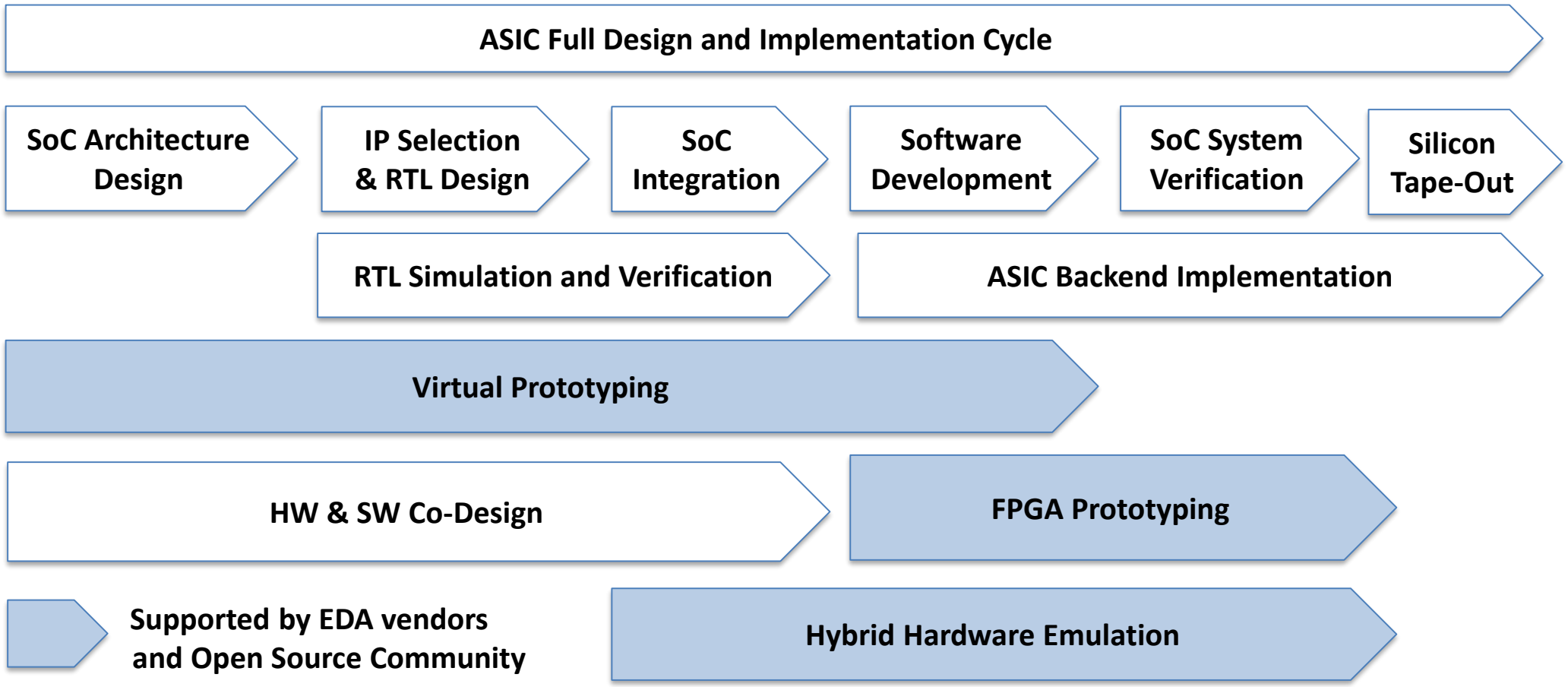




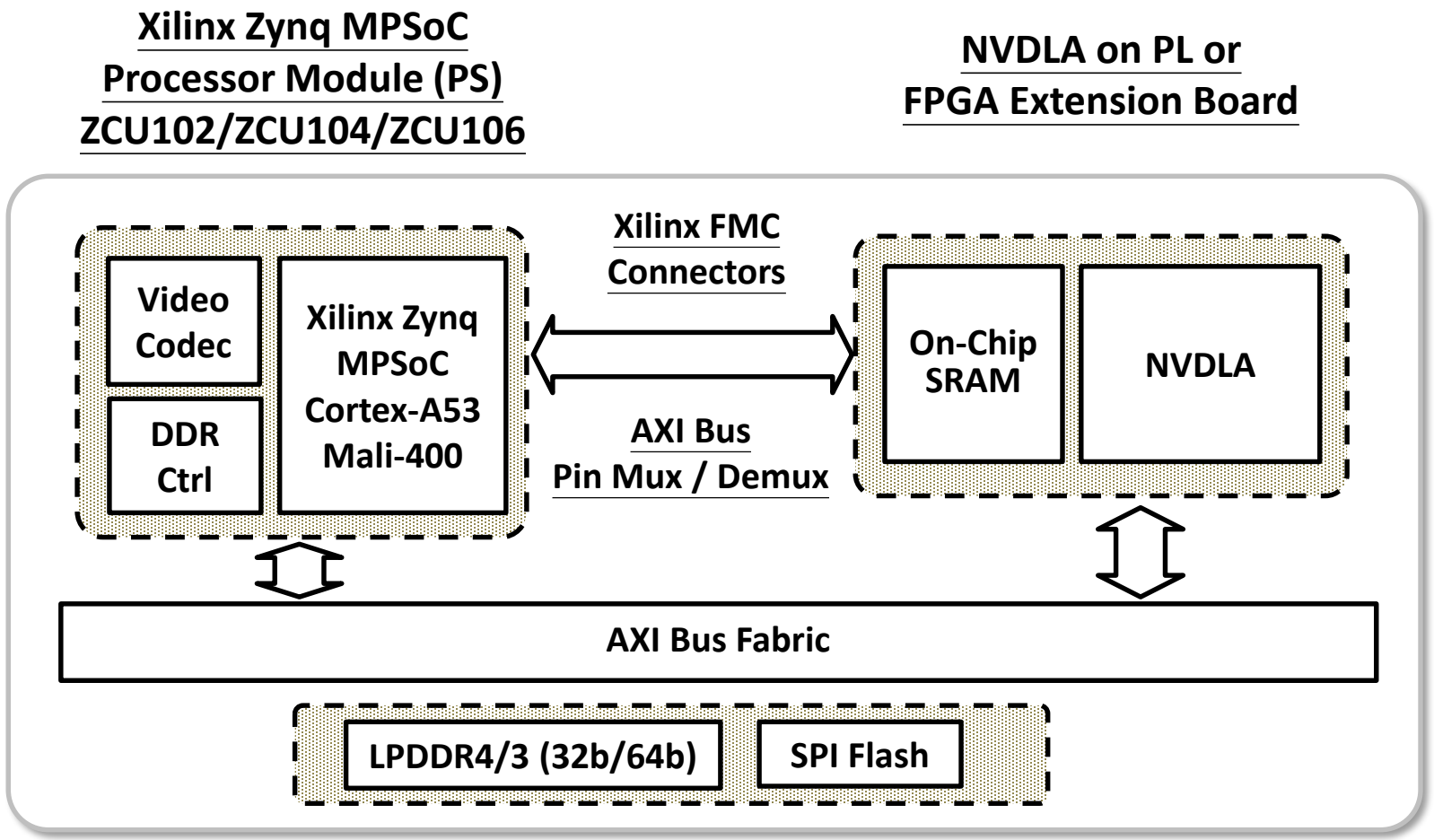
Near-optimal results: ONNC Best-Fit with Heuristic and Reordering



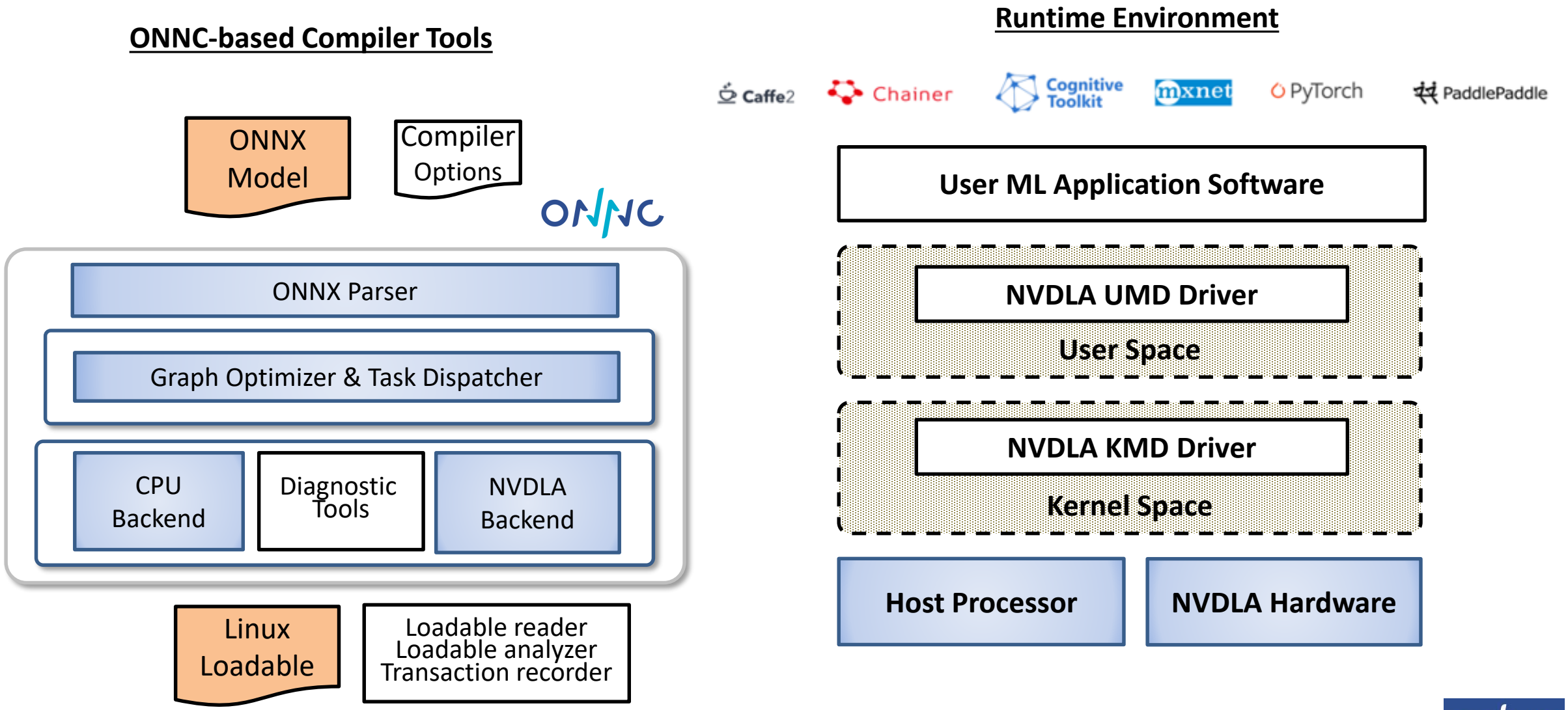
Domain-Specific Hardware & Software Co-Design



SoC Architecture and FPGA Prototyping System

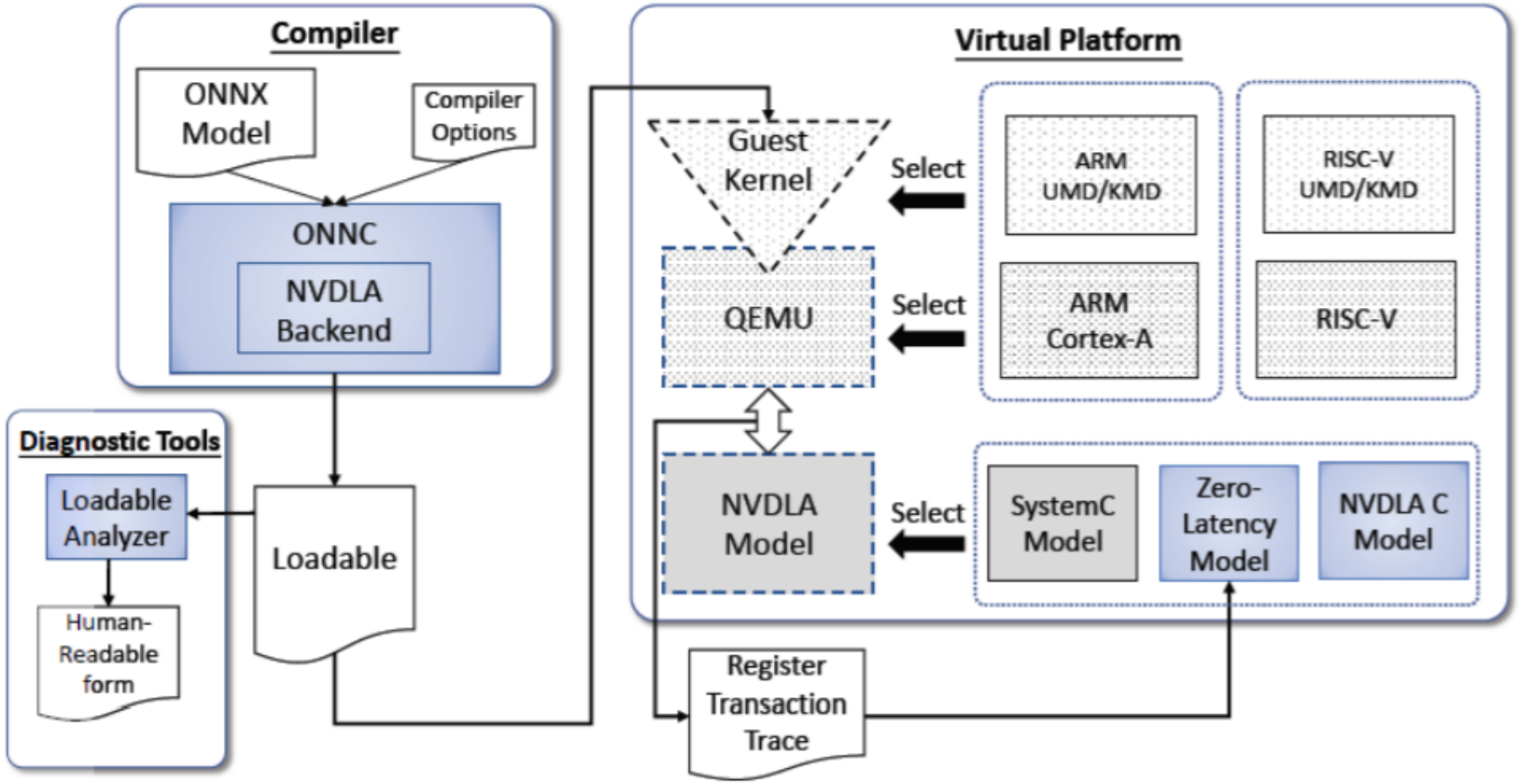


NVDLA ONNC-Based Software Framework



NVDLA Virtual Platform and ONNC SDK

- NVDLA virtual platform is built to support various hardware design tradeoffs
- Explore the hardware/software co-design and optimize at the system level



Computing Bonds vs. Memory Bonds

Alexnet, 256 MAC, 128kB C-buffer

Latency (ms)		DRAM BW				
		1GB/s	2GB/s	4GB/s	8GB/s	10GB/s
Core Speed	100MHz	75.8	47.2	32.9	25.7	24.3
	200MHz	66.9	37.9	23.6	16.4	15.0
	400MHz	63.4	33.5	19.0	11.8	10.4
	800MHz	61.8	31.7	16.7	9.5	8.1
	1000MHz	61.5	31.4	16.3	9.0	7.6

GoogleNet, 256 MAC, 128kB C-buffer

Latency (ms)		DRAM BW				
		1GB/s	2GB/s	4GB/s	8GB/s	10GB/s
Core Speed	100MHz	43.38	42.68	42.43	42.31	42.28
	200MHz	23.17	21.69	21.34	21.22	21.19
	400MHz	14.62	11.59	10.84	10.67	10.65
	800MHz	12.4	7.31	5.79	5.42	5.37
	1000MHz	12.16	6.79	4.86	4.39	4.34

ResNet50, 256 MAC, 128kB C-buffer

Latency (ms)		DRAM BW				
		1GB/s	2GB/s	4GB/s	8GB/s	10GB/s
Core Speed	100MHz	193.2	153.1	137.6	132.1	131.9
	200MHz	143.5	96.6	76.5	68.8	67.7
	400MHz	129.8	71.8	48.3	38.3	36.7
	800MHz	126.8	64.9	35.9	24.2	22.1
	1000MHz	126.3	64.3	34.3	21.6	19.3

Alexnet (~0.73 GOP, 61M weights)

- Huge fully connected weights
- DRAM speed dominates
- Computation power cannot help

GoogleNet (~3.2 GOP, 7M weights)

- Small filter size (1x1)
- Benefit parallelism in CNN operations
- Computation power dominates
- DRAM speed cannot help

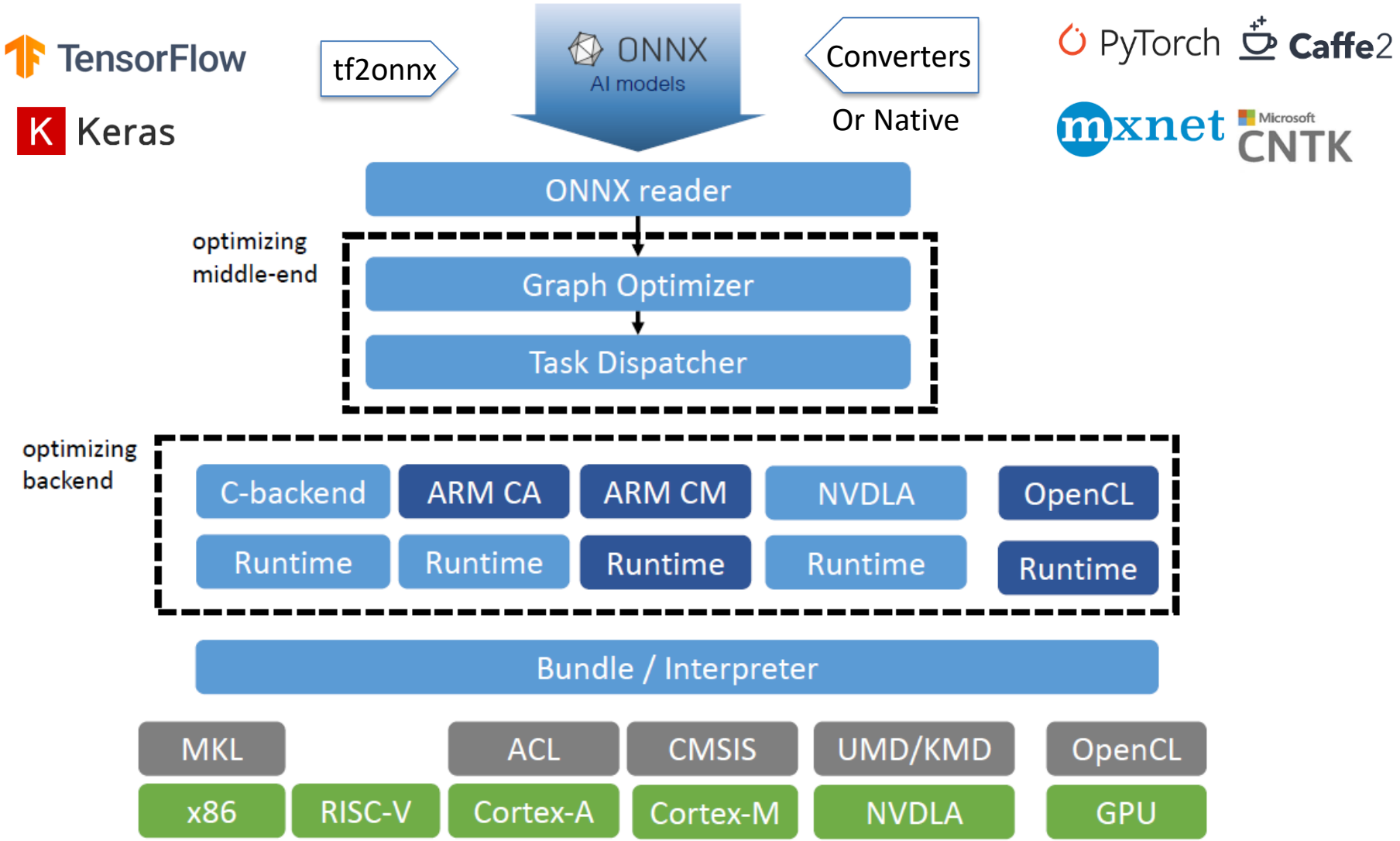
ResNet50 (~7.8 GOP, 25M weights)

- Large CNN operations, large weights
- Residual → directly add two data cubes → DRAM speed dominates
- Computation power and DRAM speed are evenly important

Open AI Toolchain for Edge Inference

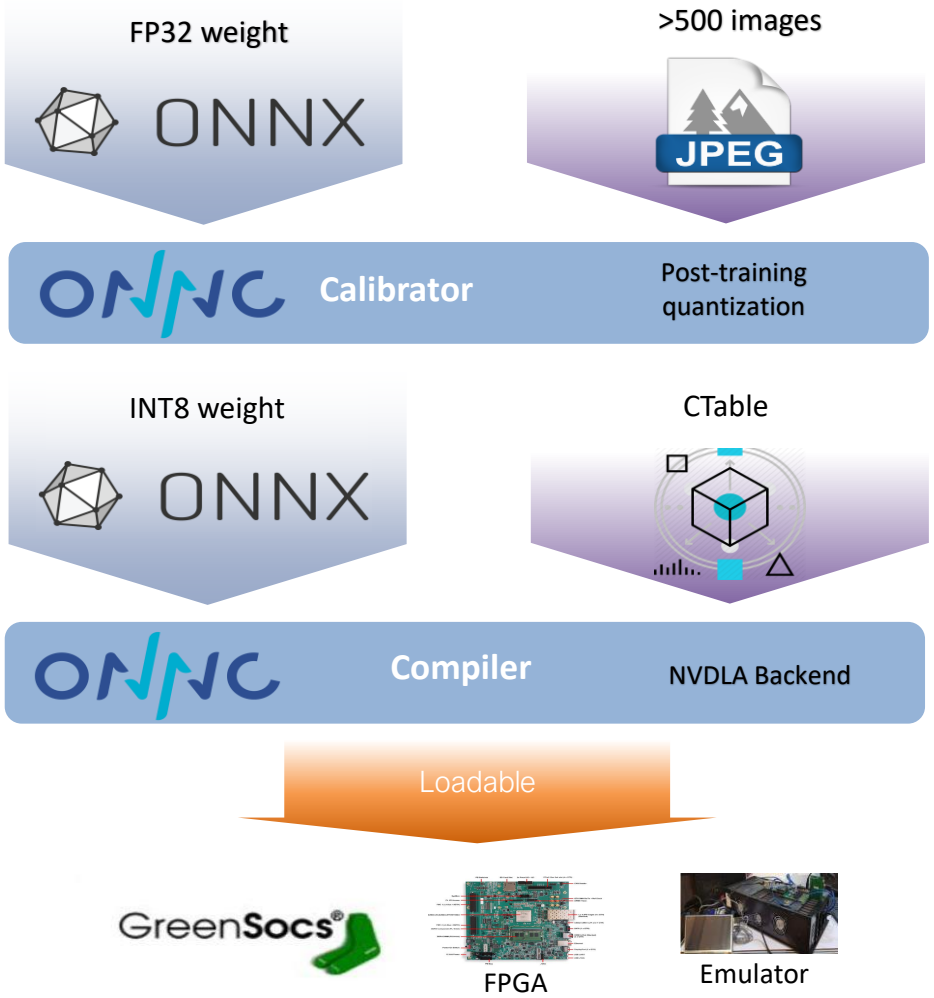
- Propose Open AI Toolchain for edge inference
  - Based on open-source compiler framework
    - ONNC (Open Neural Network Compiler)
  - Support ONNX-based DLA (Deep Learning Accelerator)
    - CNN (NVDLA, …), RNN, CIM, …, etc.
  - Support NN models from various frameworks
    - Native support or through converters
- Target Users
  - AI chip makers, silicon users, FPGA users
  - SoC architect, academic research
  - Model developer for edge inference device

Open AI Toolchain Software Framework



Calibration tool for NVDLA

- NVDLA hardware provides precision-preservation architecture
- ONNC Calibrator profiles distributions of weight parameters, input tensors and activations and create calibration table (CTable) to adjust each layer.
- ONNC Calibrator does some basic optimizations and produces a new ONNX model with INT8 weights
- ONNC compiler reads new ONNX model and CTable to produce quantized loadable file



ONNC on GitHub – Getting Started

GitHub, Inc. [US] | https://github.com/ONNC/onnc

Supported platforms

ONNC supports Ubuntu/x86_64 and MacOSX.

Here is a list of verified versions:

- Ubuntu/x86_64
  - 16.04
- MacOSX
  - High Sierra

Getting Started

There are three ways to build ONNC:

1. Build ONNC via Docker
   Please refer to the ONNC Utilities document.
2. Build ONNC via ONNC umbrella
   Please follow the instructions of README.md in **onnc-umbrella**.
   Here is the version of external library we are using in ONNC.
3. Build ONNC without ONNC umbrella
   Please refer to the ONNC Automake build instruction or ONNC CMake build instruction

https://github.com

onnc

Open Neural Network Compiler

C++ ★ 197 37 BSD-3-Clause Updated 3 days ago

onnc-umbrella

umbrella project helps you to build up onnc from scratch

Shell ★ 8 13 BSD-3-Clause Updated 14 days ago

ONNC on GitHub – Build ONNC with Docker Image

GitHub, Inc. [US] | https://github.com/ONNC/onnc/blob/master/docs/ONNC-Utilities.md

4. Build ONNC with the Docker Image

Although the Docker image include a source code tree, it might not be the latest release version of ONNC. We strongly suggest you clone the latest version of ONNC from the GitHub repository, mount the source code directory to the Docker image, and modify the source code with your favorite editor on your host machine. You may clone the source code from the GitHub ONNC repository (https://github.com/ONNC/onnc). To download large model files when cloning the ONNC source, you have to install Git LFS (https://github.com/git-lfs/git-lfs/wiki/Installation) first.

```
$ mkdir -p <source_dir> && cd <source_dir>
$ git clone https://github.com/ONNC/onnc.git
```

Once the latest source code is ready, you may invoke the following command to enter ONNC build environment:

```
$ docker run -ti --rm --cap-add=SYS_PTRACE -v <source_dir>/onnc:/onnc/onnc onnc/onnc-community
```

`<source_dir>` is the directory where you cloned the latest ONNC source code and the `-ti` option provides a interactive interface for the container, the `-v` option mounts the directory to the Docker image, the `--cap-add=SYS_PTRACE` option enables debug support (e.g. gdb) in the container. You can make some change to the source code (`<source_dir>/onnc`) and run the following command to build ONNC.

```
// run in the container cli, under build directory `/onnc/onnc-umbrella/build-normal` by default
$ smake -j8 install
```

ONNC on GitHub – Running ONNX Models

GitHub, Inc. [US] | https://github.com/ONNC/onnc/blob/master/docs/ONNC-Utilities.md

Running a Single Benchmark

You may run a single model for benchmarking using the following shell command:

```
// run in the container cli
$ onni <model_file_path>/model.onnx <input_file_path>/input_0.pb -verbose=<level>
```

where `<model_file_path>` is the path to the model file for the pre-trained ONNX model and `<input_file_path>` is the path to the corresponding input file. In the ONNC Docker container, the model file path is `/models/<model_name>` and the input file path is `/models/<model_name>/test_data_set_<0~6>`. `<level>` indicates different levels of verbose information. Higher-level information is a superset of all lower-level information. For example, level 4 will include all information from level 1 to level 4.

Information for each verbose level:

- Level 1: Inference time & memory usage
- Level 2: ONNX operator statistics
- Level 3: Inference time & ONNX operator statistics per layer
- Level 4: Memory allocation log

Here is an example of running AlexNet and printing out all information.

```
// run in the container cli
$ onni /models/bvlc_alexnet/model.onnx /models/bvlc_alexnet/test_data_set_0/input_0.pb -verbose=4
```

Q & A

人工智能系统与集成电路研讨会

2019年8月18-19日