Implementation

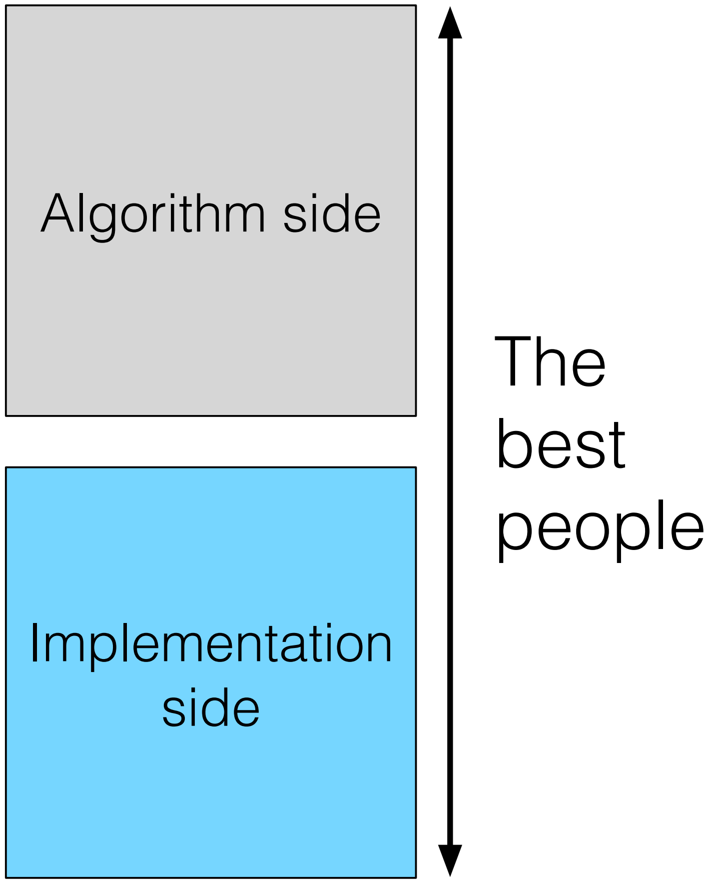
Arthur J. Redfern
arthur.redfern@utdallas.edu

Disclaimer

- Previous math lectures
  - A broad presentation of material (linear algebra, calculus, probability and algorithms)
  - Perhaps a little biasing from me in terms of presentation, importance and intuition
  - But in general the material stands on its own and there's not ambiguity at our level of review
- Previous network design and training lectures
  - A broad presentation of material (design and training)
  - A little more biasing from me in terms of presentation, importance and intuition
  - But pointers to many many references were provided for you to go deeper on any topic
- This series of network implementation lectures
  - We'll still cover a lot of topics, but the presentation of material will be more narrow
  - You'll get more of my opinion in terms of the right way to do things
  - But pointers will still be given to additional references and you're free to draw your own differing conclusions

Why Discuss Implementations

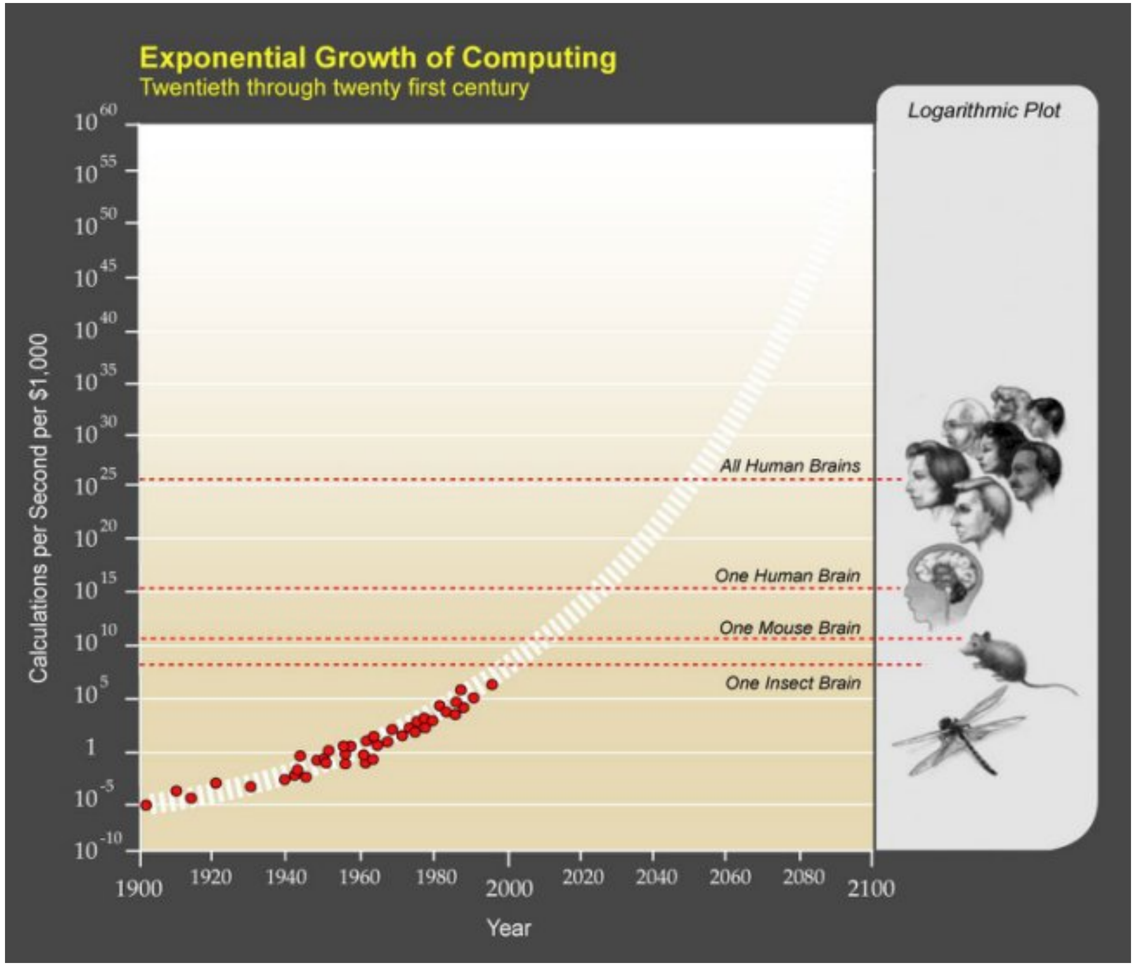
- Progress in xNNs is directly linked to progress in improved implementations
- At large companies with big ML related business
  - About 1/2 the people are on the algorithm side
  - About 1/2 the people are on the implementation side
- The best people understand both
  - I want you to understand both



For some interesting thoughts related to this see The Bitter Lesson http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Outline

- Graph processing
- xNN
  - Design
  - Software
  - Hardware
  - Configurations
  - Performance



Co Design For Optimality

- Design networks for optimal performance on hardware

- Design software to optimally map networks to hardware

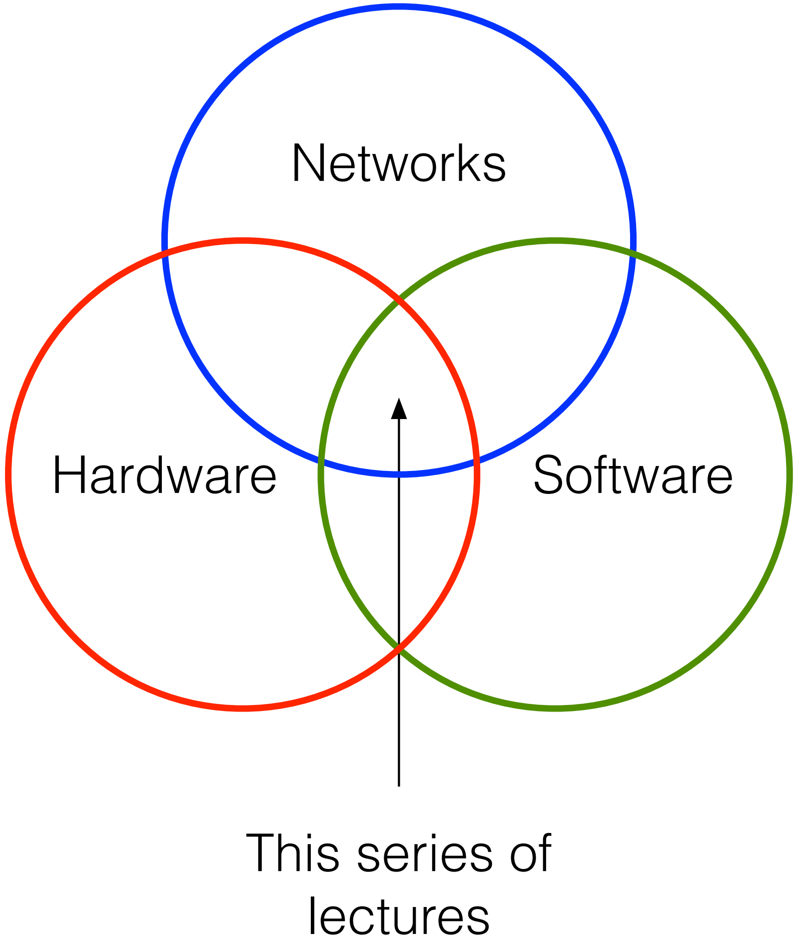
- Design hardware to optimally execute networks

A Lecturer's Apology

- Presentation of material in this lecture is sequential
  - Design (networks)
  - Software
  - Hardware

- But all of these topics are actually optimized together at the same time
  - As such, there are dependencies in the material

- A suggested slide reading strategy to address this interdependence
  - Start with a bit of a high level view of everything
  - Then sequential presentation of topics in more detail from me
  - Then go back and re read having seen everything once before



Graph Processing

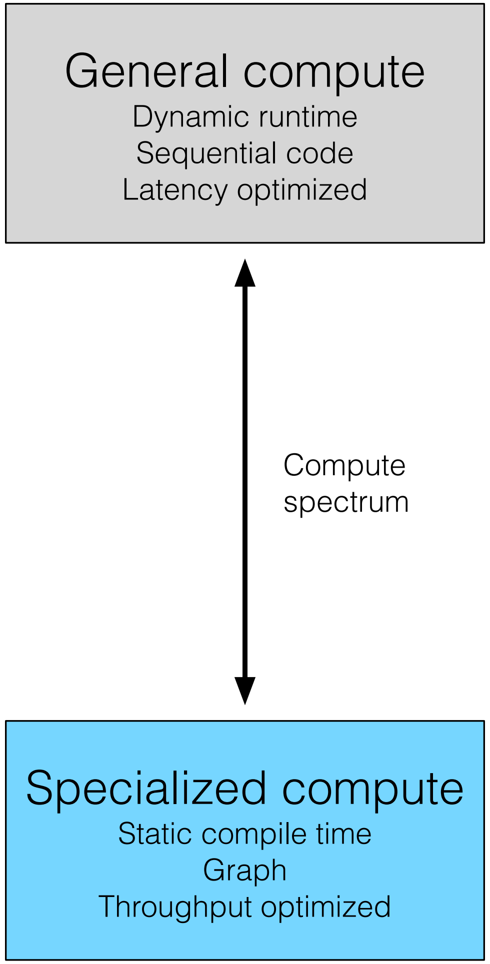
The Future Of Hardware And Software

Yes, that's a slightly grandiose slide title / slight exaggeration; no, it's not that far from the truth

- Is a bifurcation where only 2 points matter
  - Big code, small general compute
  - Small code, big specialized compute

- Big code, small general compute → map to host (x86, ARM, RISC-V, ...)
  - Hardware agnostic software
  - Runtime intelligent hardware
    - Cache, branch prediction, out of order processing, speculative execution, ...
    - This has been beaten to death, gains are small and incremental; you're picking up crumbs
  - Examples: high level operating systems, control code, ...

- Small code, big specialized compute → map to (a better version of a) DSA
  - Compile time intelligent software
  - Runtime deterministic hardware
    - This is where the action and ability to differentiate in hardware is
  - Examples: xNNs, almost all other technologies you're going to be interested in, ...

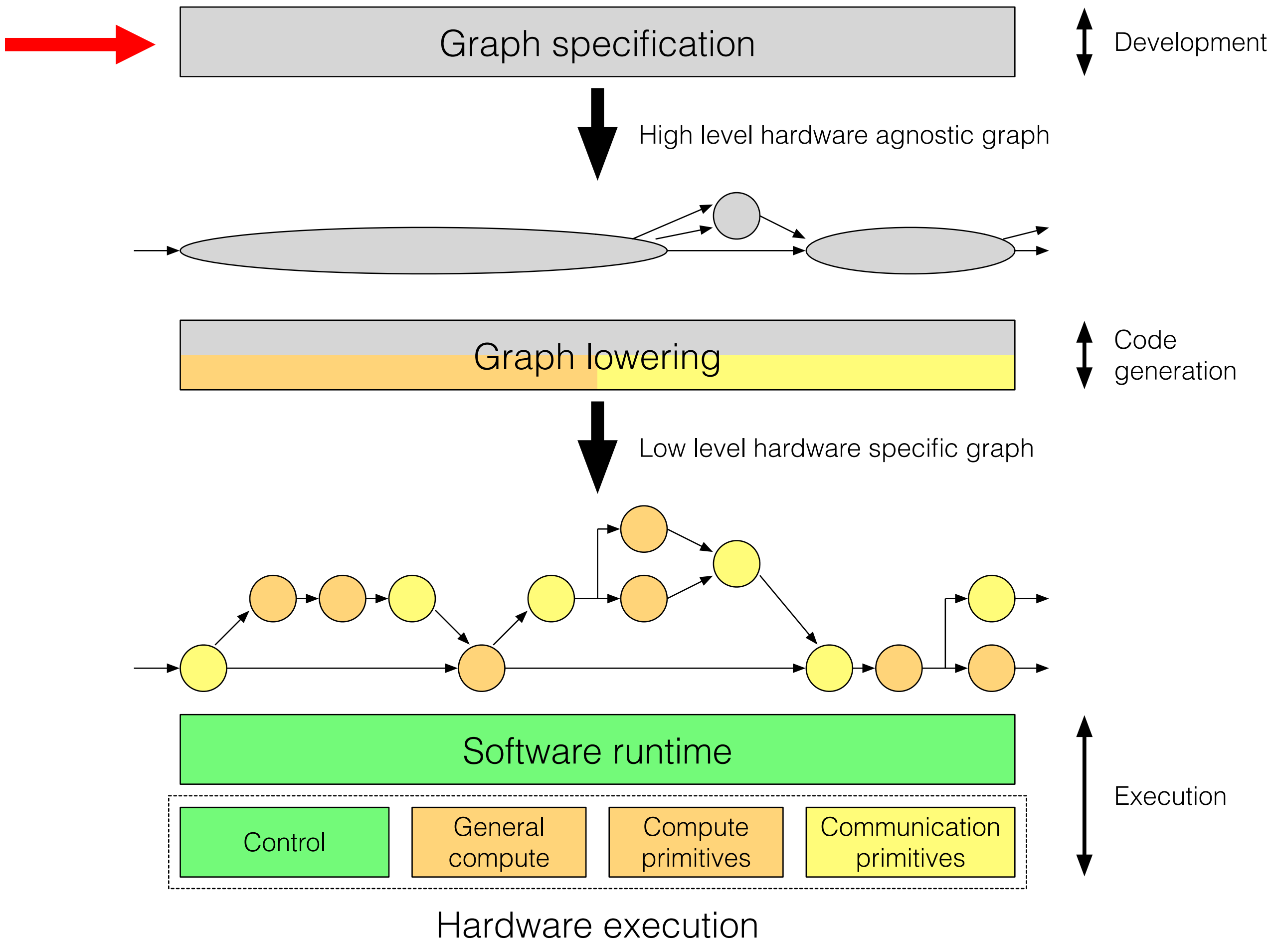


Graph Specification

- Graph specification (as used here) is a high level hardware agnostic algorithm description

- Nodes represent operators
  - Compute
  - Communication is not explicitly included
    - Implicit instruction movement
    - Implicit data movement

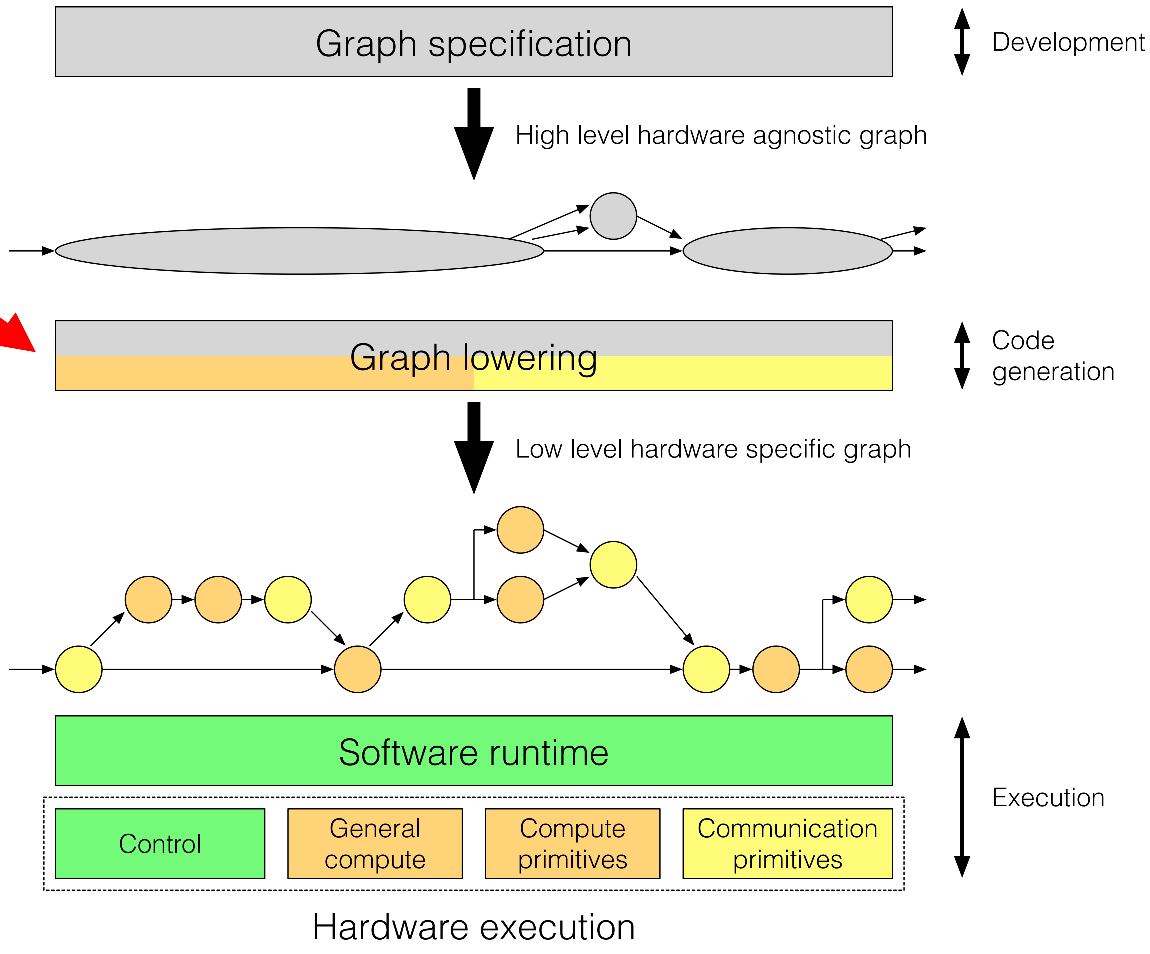
- Edges represent dependencies
  - Typically tensors (memory)



Graph Lowering

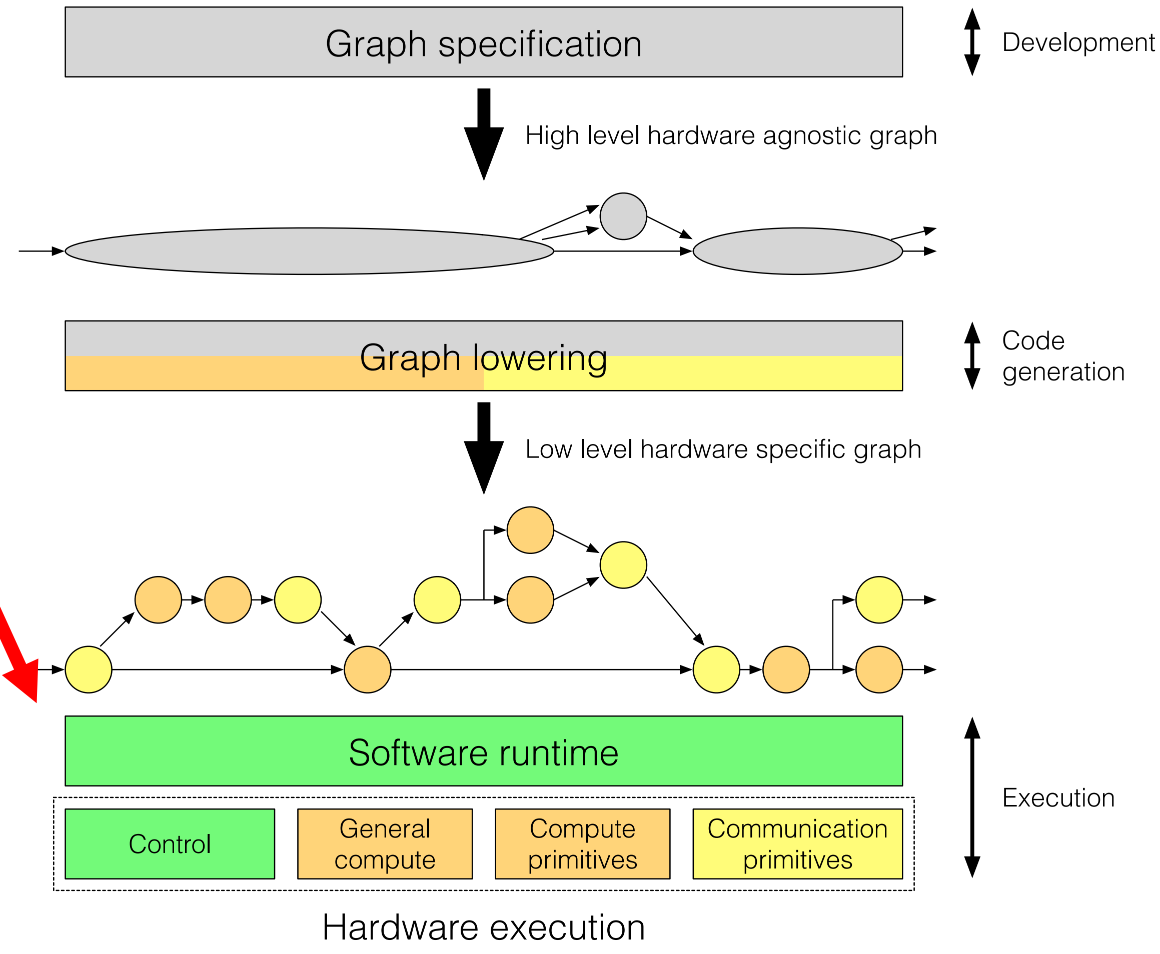
- Graph lowering maps a high level hardware agnostic graph to a low level hardware specific graph
    - Compute and communication nodes including code generation
    - Data and instructions edges including memory placement
    - Nodes and edges map 1 to 1 to hardware

- Graph lowering is an iterative optimization process
    - Domain agnostic and domain specific components
    - Hardware agnostic, hardware aware and hardware specific components



Graph Execution

- Graph execution includes software runtime and hardware execution
- The software runtime runs on the control processor and cycles through nodes on the low level graph
    - Also doing some things like tying dynamic externally managed tensors into the graph
- The hardware executes the nodes using computation and communication primitives for key nodes and general compute for everything else



xNN Design

How To Design High Performance xNNs

- Definitions (as used here)
  - Accuracy is a measure of the correctness of the information extracted from data
  - Performance is a measure of how fast the network runs

- Implicit in the definition of performance is “on hardware”
  - So there’s a need to know a bit about hardware in general and a bit about the specific hardware that the xNN is running on
  - We’ll think of this as hardware aware xNN design

- Some hardware aware xNN design considerations for improving performance
  - Operator selection: type, quantization, sparsification and compression
  - Network sizing: depth, width and input size

Training / Testing Have Different Constraints

Training

- Can batch inputs (allowing the amortizing of weight movement across multiple inputs)
- Need batch norm operations
- Need error calculation
- Need to maintain memory space for reverse mode automatic differentiation so there's less reuse of memory
- Typically need higher precision floating point

Testing

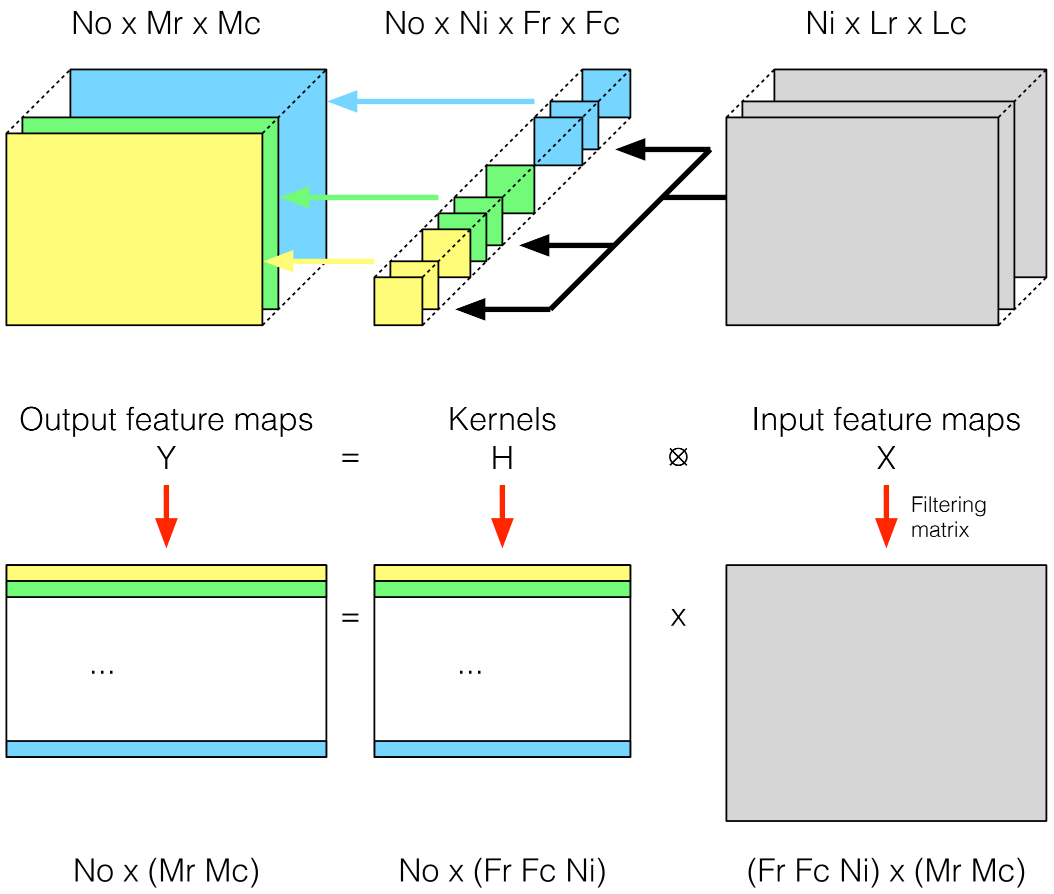
- Sometimes can batch inputs; sometimes process 1 input at a time for latency reasons
- Absorb batch norm operations
- Need network output
- Don't need to maintain memory space for reverse mode automatic differentiation so there's more reuse of memory possible
- Often ok with lower precision fixed point

xNN Design – Operator Selection

Key Operators Are Built On Matrix Mult

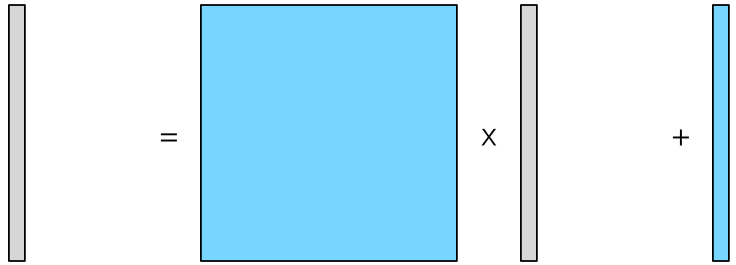
NN, RNN and variants, CNN style 2D convolution and variants, attention / self attention and variants, average pooling, ...

CNN style 2D convolution

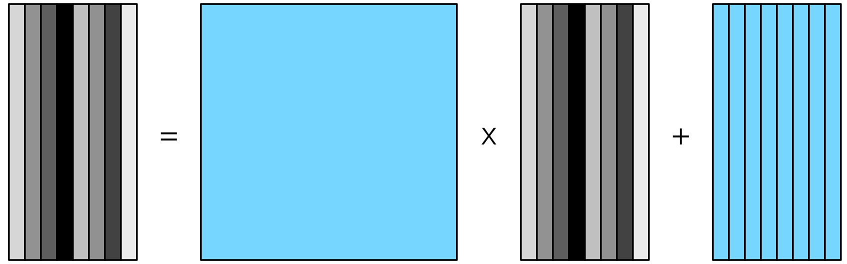


Fully connected layer

Single input (communication limited by the weight matrix)



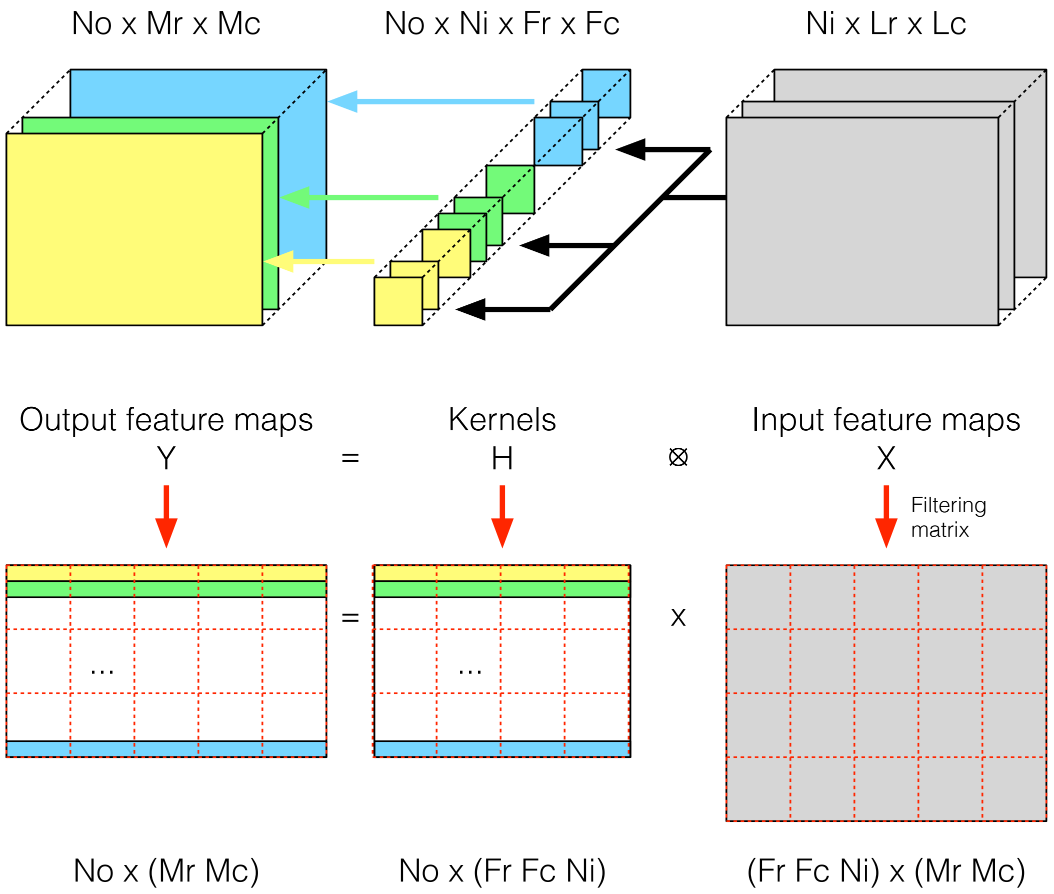
Batch input or multiple regions (compute limited by matrix multiplication)



Key Operators Are Built On Matrix Mult

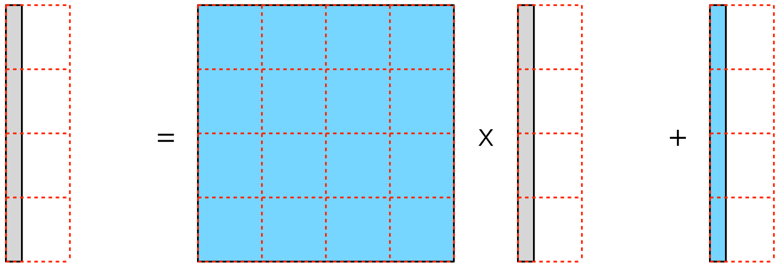
Arbitrarily large matrix multiplication is efficient to accelerate via a matrix multiplication primitive

CNN style 2D convolution

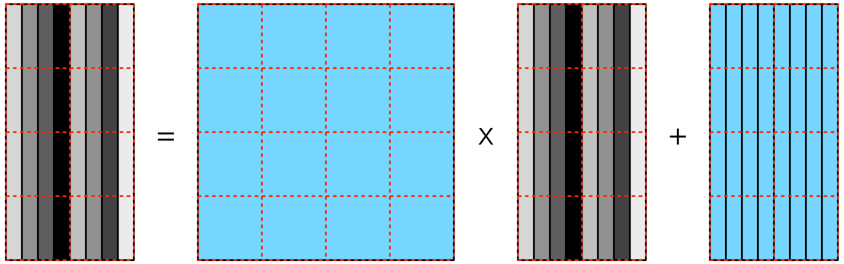


Fully connected layer

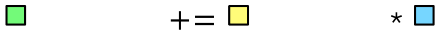
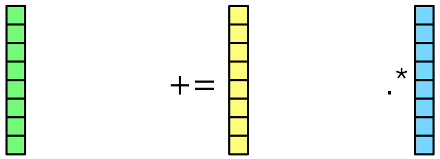
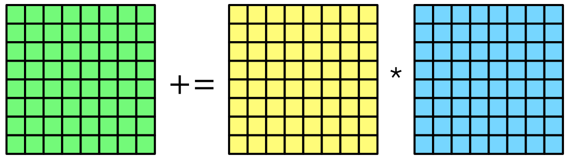
Single input (communication limited by the weight matrix)



Batch input or multiple regions (compute limited by matrix multiplication)



Matrix Multiplication Is Special

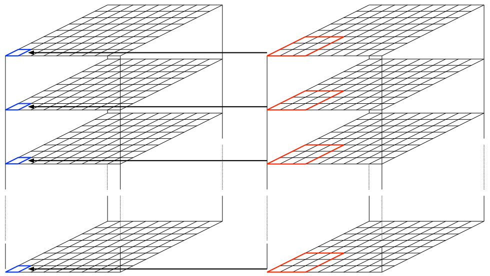
Arch	Compute	Visualization	Compute / data	Memory
Scalar (traditional CPUs)	c += a*b		$\frac{1}{3}\ \frac{\text{MAC}}{\text{data}}$	Cache
Vector (curr CPUs, DSPs)	c += a.*b		$\frac{N}{3N} = \frac{1}{3}\ \frac{\text{MAC}}{\text{data}}$	Cache + scratch xD --> 1D stream
Matrix (GPUs, FPGAs)	C += A*B		$\frac{N^3}{3N^2} = \frac{N}{3}\ \frac{\text{MAC}}{\text{data}}$	xD --> 1D stream

Why are bubbles spherical shaped? Why choose a square matrix? Because square matrix sizes maximize the compute to data ratio (max M*N*K given M*N + M*K + N*K = constant --> M = N = K)

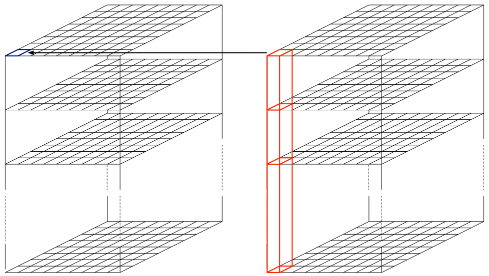
Why is 1 large accelerator better than many small accelerators? Because it minimizes the excess data movement and delay for inherently sequential operations

What This Means For Operator Choice

- If you look at top performing networks from an accuracy vs {MACs, weights} perspective, they will invariably use a lot of low arithmetic intensity operations
  - Less weights and MACs
  - Depth wise convolution (2D std conv) lowering to matrix vector multiplication
  - Global avg pool → matrix vector multiplication for squeeze and excite layers
  - GhostNet takes this a step further
- But this can be deceptive to predicting the overall performance on an optimized accelerator as data movement can become the real bottleneck (we'll see this soon)



Spatial mixing



Channel mixing

- So it really makes sense to pay attention to data movement as much as compute
  - And with respect to data movement, understand what parts of data movement you can hide and which parts you can't
  - This is network and op dependent
  - Predict via a low level graph approach

xNN Design – Quantization

Quantization Improves Performance

Assuming that the hardware can take advantage of it

- Quantization improves performance via reducing the number of bits used to represent feature maps and parameters
  - Quantization is ideally done ~ towards the end of training via a deflationary style approach
  - Quantization can also be done after training

- Challenges include
  - Additive, concatenative and batch norm operations
  - Feature maps are dynamic

For more details see Quantizing deep convolutional networks for efficient inference: A whitepaper (https://arxiv.org/abs/1806.08342)

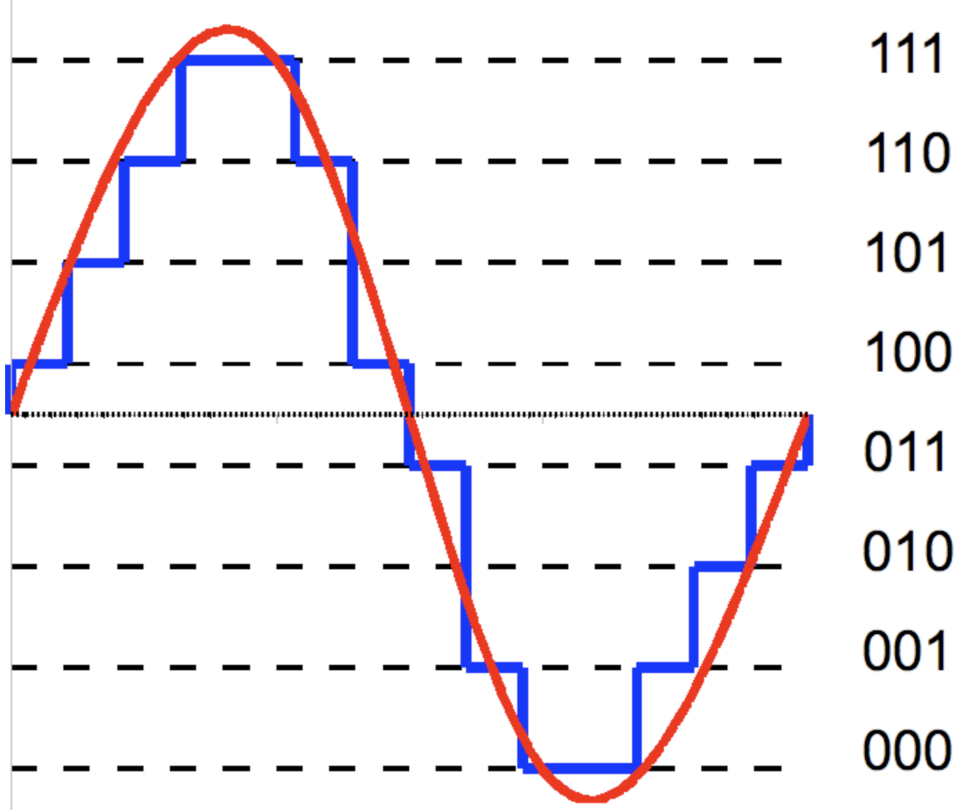


Figure from https://en.wikipedia.org/wiki/Quantization_(signal_processing)

The Performance Impact Of Quantization

Assuming that the hardware can take advantage of it

- Memory and communication scale linearly with the number of bits
- The complexity of addition and comparison operations scale ~ linearly with the number of bits
- The complexity of multiplication operations scale to the ~ square of the number of bits (linearly with each operand)

8 and 16 bit fixed point multiplication

Let x_8^{lo}, x_8^{hi}, y_8^{lo} and y_8^{hi} be 8 bit integers and let x_{16} and y_{16} be 16 bit integers such that

$$x_{16} = 2^8\, x_8^{hi} + x_8^{lo}$$
$$y_{16} = 2^8\, y_8^{hi} + y_8^{lo}$$

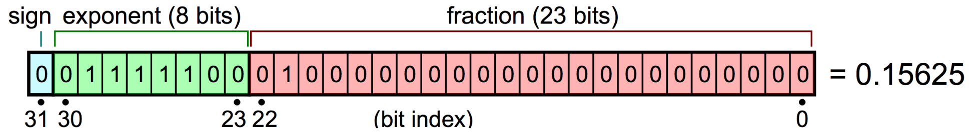
Then 16 bit integer multiplication can be implemented via 4x 8 bit multiplication, 3 shifts and 3 adds

$$\begin{aligned} x_{16}\, y_{16} &= (2^8\, x_8^{hi} + x_8^{lo})\,(2^8\, y_8^{hi} + y_8^{lo}) \\ &= 2^{16}\, x_8^{hi}\, y_8^{hi} + 2^8\, x_8^{hi}\, y_8^{lo} + 2^8\, x_8^{lo}\, y_8^{hi} + x_8^{lo}\, y_8^{lo} \end{aligned}$$

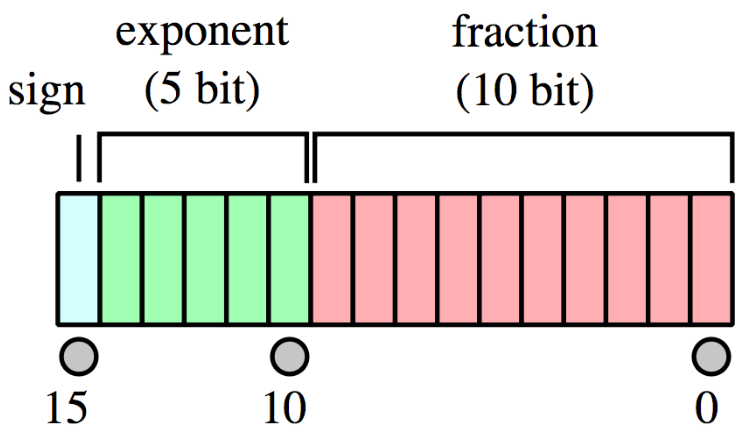
2x the number of bits → ~ 4x the integer multiplier complexity

Common Floating Point Formats

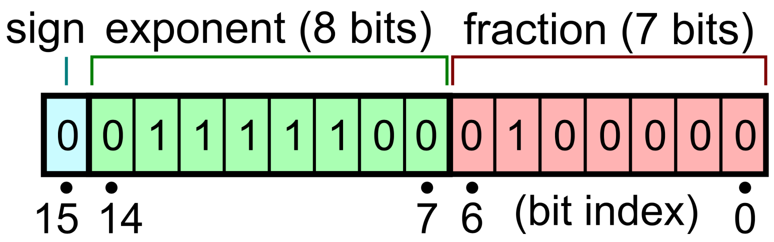
- IEEE 754 float 32: 1 bit sign, 8 bits exponent, 23 bits significand
  - Ignoring special values signaled by exponent values of 0 and 255
  - Value = sign x range x precision
    $= (-1)^{\text{sign}} \times 2^{\text{exponent} - 127} \times 1.\text{significand}_2$
    $= \{-1, 1\} \times 2^{\{-126, \ldots, 127\}} \times [1, 2)_{\text{23 bits precision}}$



- IEEE 754 float 16: 1 bit sign, 5 bits exponent, 10 bits significand
  - Ignoring special values signaled by exponent values of 0 and 31
  - Value $= (-1)^{\text{sign}} \times 2^{\text{exponent} - 15} \times 1.\text{significand}_2$
    $= \{-1, 1\} \times 2^{\{-14, \ldots, 15\}} \times [1, 2)_{\text{10 bits precision}}$



- bfloat 16: 1 bit sign, 8 bits exponent, 7 bits significand
  - Ignoring special values signaled by exponent values of 0 and 255
  - Value $= (-1)^{\text{sign}} \times 2^{\text{exponent} - 127} \times 1.\text{significand}_2$
    $= \{-1, 1\} \times 2^{\{-126, \ldots, 127\}} \times [1, 2)_{\text{7 bits precision}}$



Figures from https://en.wikipedia.org/wiki/IEEE_754 and https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

Common Floating Point Uses

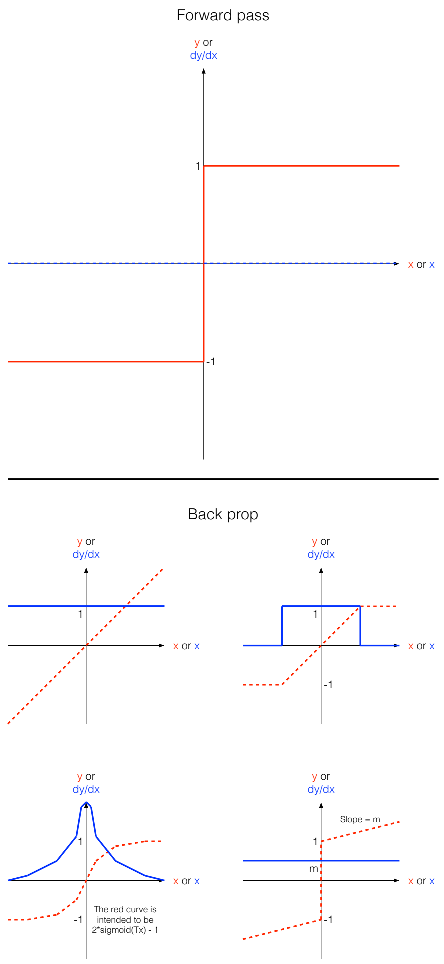
- IEEE 32 bit float (24.8): the starting point / default training precision

- Bfloat16 (8.8): if your xNN library and hardware supports bfloat16 you can typically use this as a drop in replacement for 32 bit float and everything will work with minimal or no modification
  - Same 8 bit exponent range, precision goes from 24 to 8 bits but xNNs are relatively insensitive to that
  - Training performance will improve (less memory, less data movement, faster compute)
  - Bfloat16 is also convenient to use for inference
    - Better performance than 32 bit float (but worse performance than 8 bit fixed)
    - Directly use it after training in bfloat16 (no additional quantization is needed)
    - After training in 32 bit float, quantize to it to bfloat16 via dropping precision

- IEEE 16 bit float (11.5): not a fan, but mentioning it here as you'll potentially run into it on specific GPUs
  - Can sometimes use for training
  - Will typically need to play some range modification tricks and potentially also still do accumulation of gradients in 32 bit float

Common Fixed Point Formats

- Int8 (8 bits): this is the current sweet spot for CNN inference
  - Need to do scaling (implicitly / explicitly) to keep things in range
  - Post training quantization is sub optimal, better to quantize during the last few epochs of training
  - Insertion of 8 bit quantization operations into the data path and weight path to mimic the inference operation kills off the gradient flow
  - Solution is to use something like the straight through operator decoupling decoupling the forward and backward paths or a smoothed quasi quantization function that is adapted to converge to the exact quantization function
  - Challenges are aligning ranges in places where multiple paths add or are concatenated together; for this reason bias is usually kept at 32 bit fixed
- Binary (1 bit): this is the extreme end of quantization
  - 1 bit weights and feature map inputs to CNN style 2D convolution
  - 8 bit feature maps at the output of CNN style 2D convolution and in identity paths
  - Longer multi stage training pipelines are used addressing issues similar to the standard quantization with respect to back propagation challenges



Floating Point To Fixed Point Conversion

Signed (symmetric) fixed point is an integer in $\{-2^{bits-1}, \ldots, 2^{bits-1} - 1\}$ and unsigned fixed point is an integer in $\{0, \ldots, 2^{bits} - 1\}$

- Signed example for $\{X_q, s_X\}$ = quantize(**X**, bits)
  - Target range of $\{-2^{bits-1} - 1, \ldots, 2^{bits-1} - 1\}$
    - Keeping symmetric about 0 by ignoring the value -2^{bits-1}
    - For 8 bits this is {–127, ..., 127}
    - Assume scale s_X is chosen such that there's no clipping
  - maxAbsX = max(|**X**|)
  - s_X = maxAbsX / $(2^{bits-1} - 1)$
  - X_q = round(**X** / s_X)

- Unsigned example for $\{X_q, s_X\}$ = quantize(**X**, bits)
  - Target range of $\{0, \ldots, 2^{bits} - 1\}$
    - For 8 bits this is {0, ..., 255}
    - Assume input **X** is non negative
    - Assume scale s_X is chosen such that there's no clipping
  - maxX = max(**X**)
  - s_X = maxX / $(2^{bits} - 1)$
  - X_q = round(**X** / s_X)

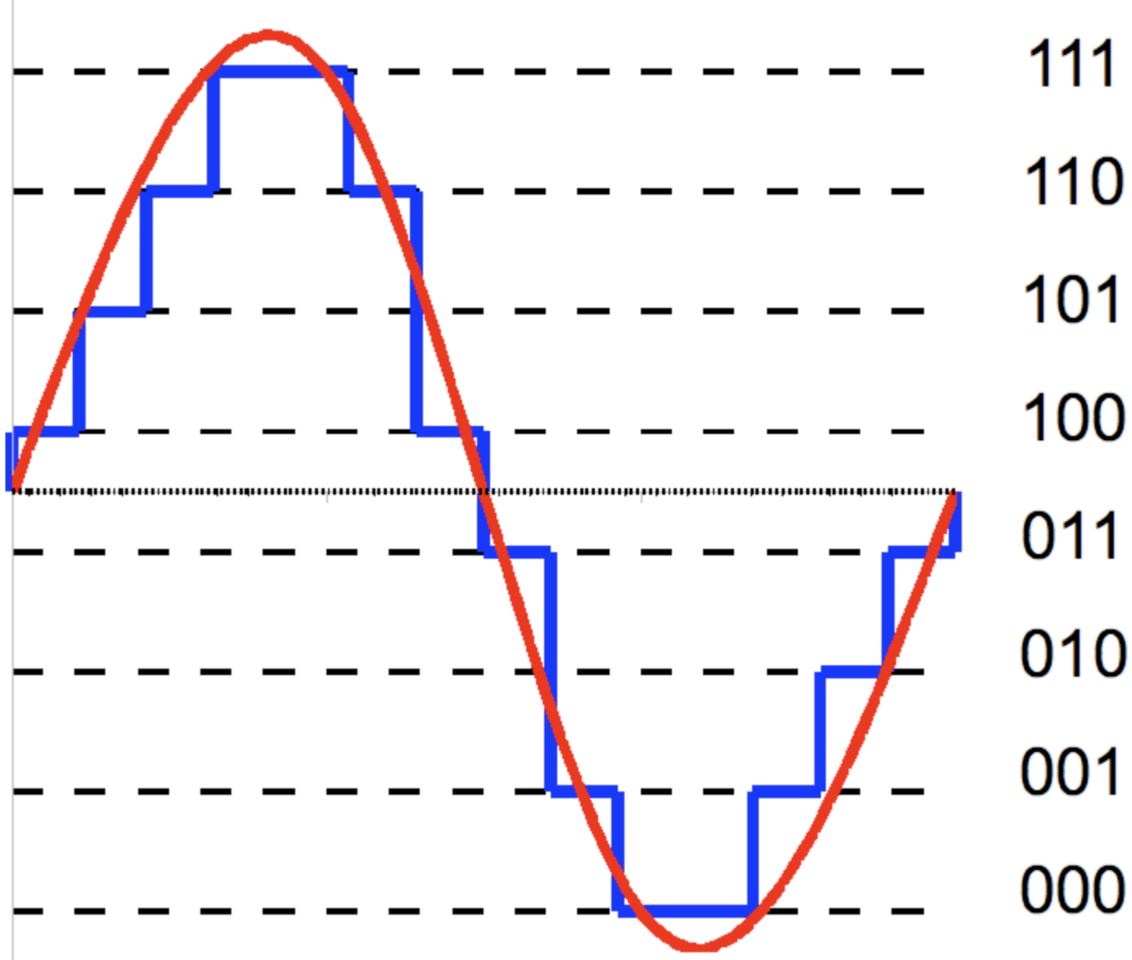


Figure from https://en.wikipedia.org/wiki/Quantization_(signal_processing)

Example Training Graph Modifications

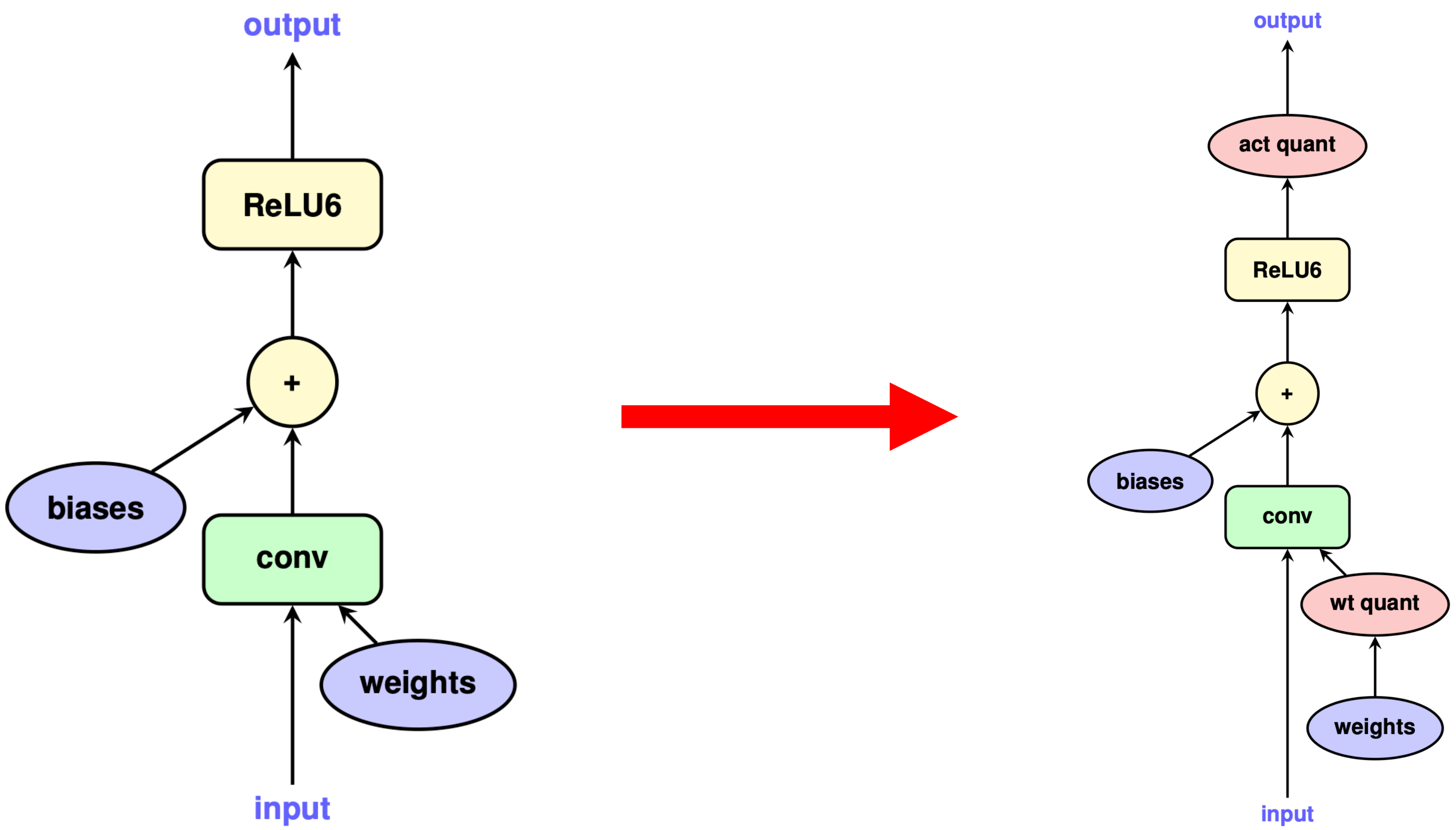


Figure from https://arxiv.org/abs/1712.05877

Example Training Graph Modifications

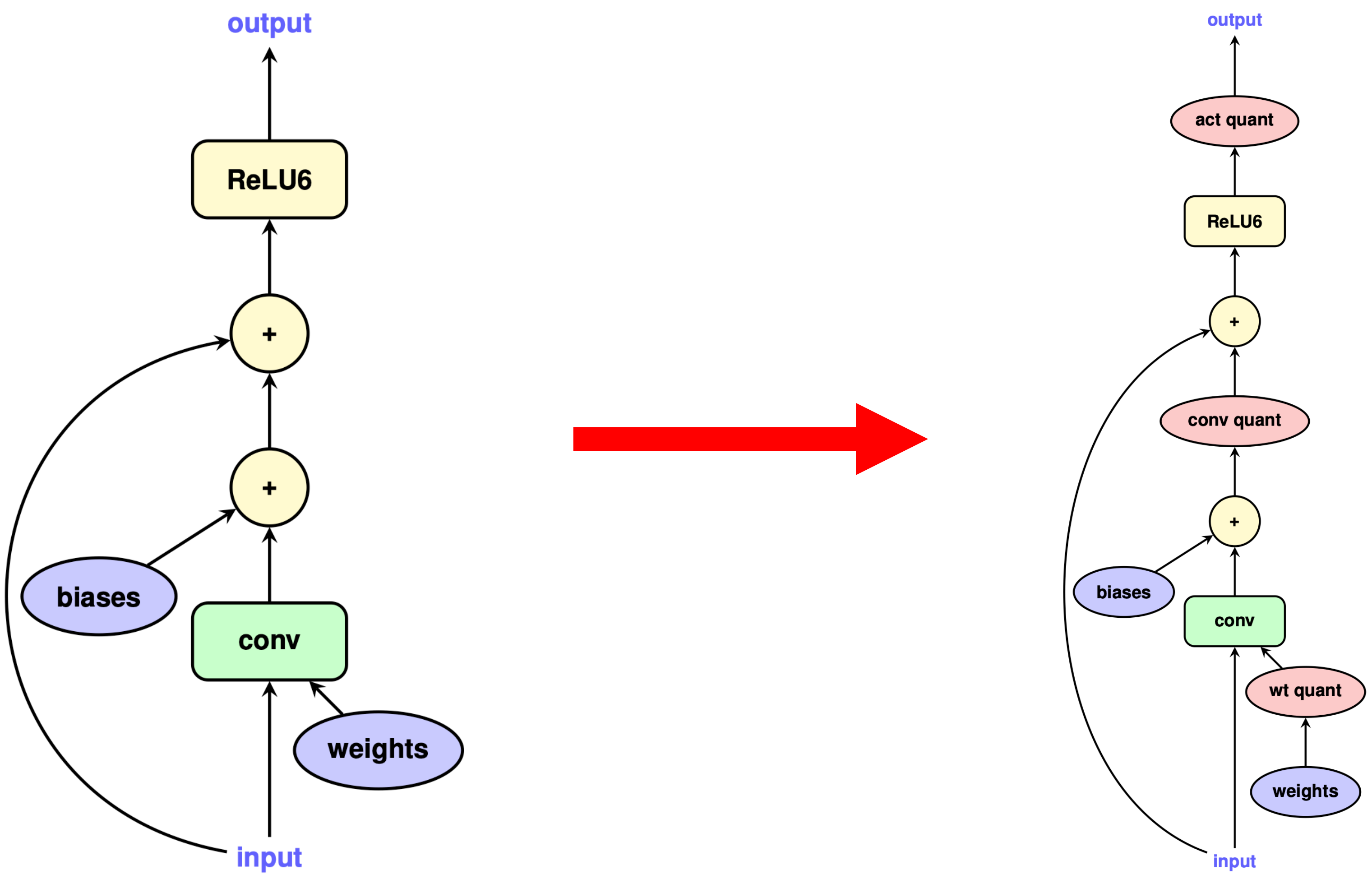


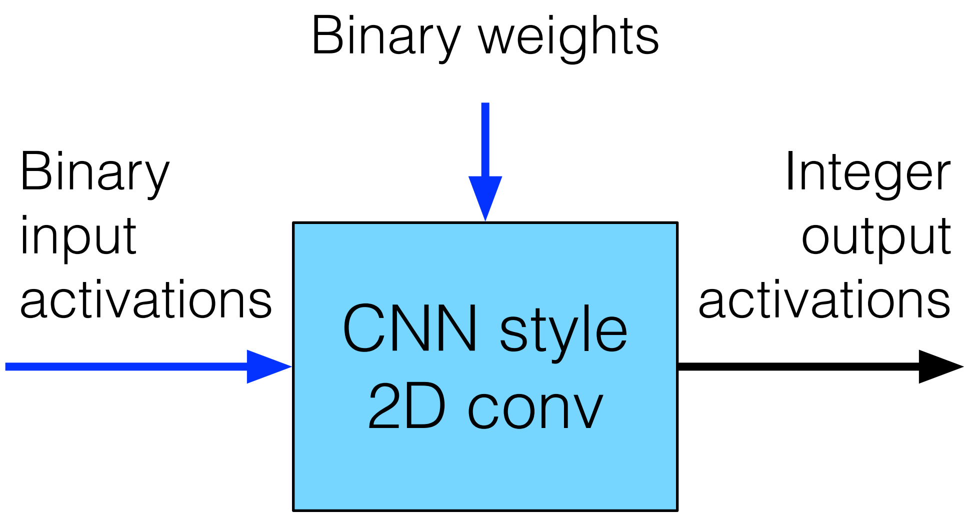
Figure from https://arxiv.org/abs/1712.05877

Binary CNNs == Extreme End Point Of Quant

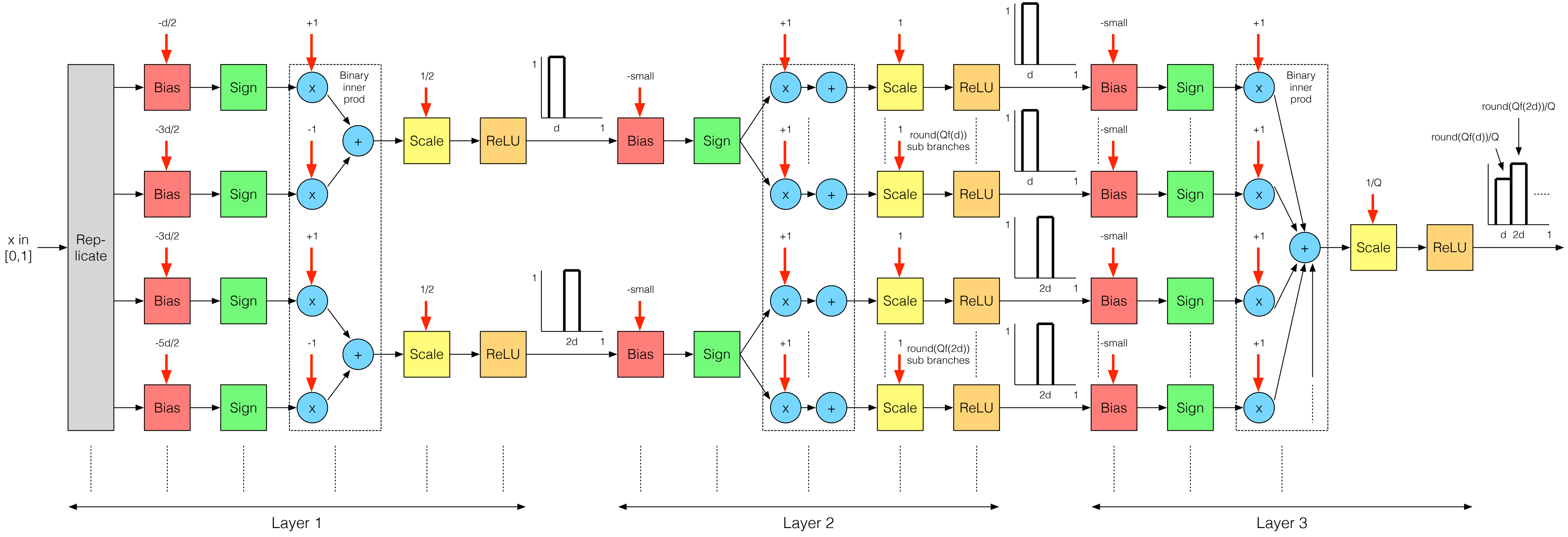
- Matrix matrix multiplication operations are fully binary (binary inputs and weights), other operations can be real (or int 8)

- Offer the highest opportunity for performance per watt or area, typically at the expense of accuracy

- Strategies for improving binary CNN accuracy
  - Real residual identity connections around the binary convolution
  - Learnable bias before binarization in the data path
  - Parallel convolution branches
  - Real batch norm at the convolution output

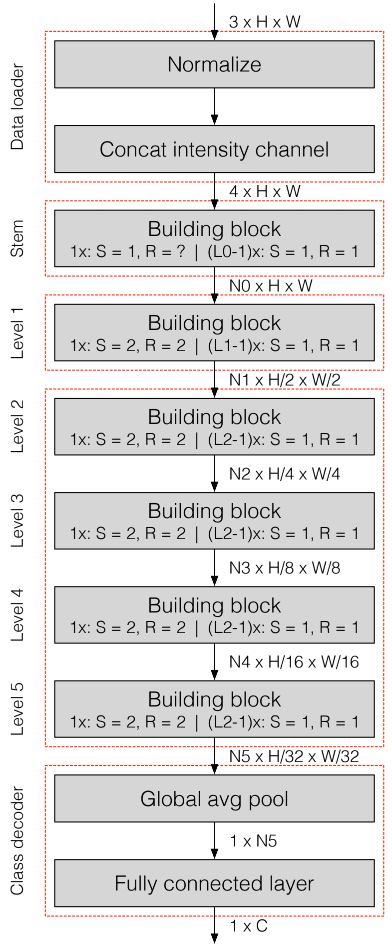


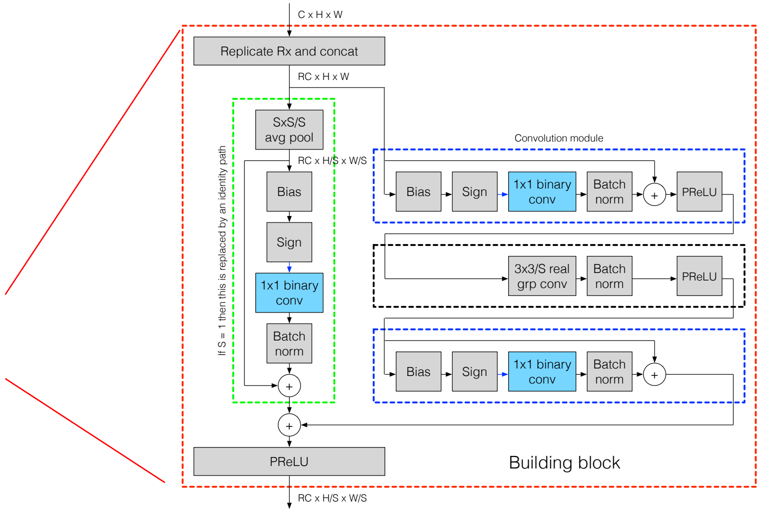
Binary CNNs Are Still Universal Func Approx



And Can Achieve High Accuracy In Practice

- Example
  - CNN with all matrix ops fully binary
  - 3 x 224 x 224 input + intensity channel
  - 1-1-1-2-6-2 block repeats
  - 32-64-128-256-512-1024 channels
  - Global avg pool – fully connected decoder
- Complexity
  - Binary params 9.04 e6
  - Binary MACs 2.41 e9
  - Real params 1.15 e6 (~ all in last FC layer)
  - Real OPs/2 0.09 e9
- BCNN: a binary CNN with all matrix ops quantized to 1 bit precision
  - https://arxiv.org/abs/2010.00704



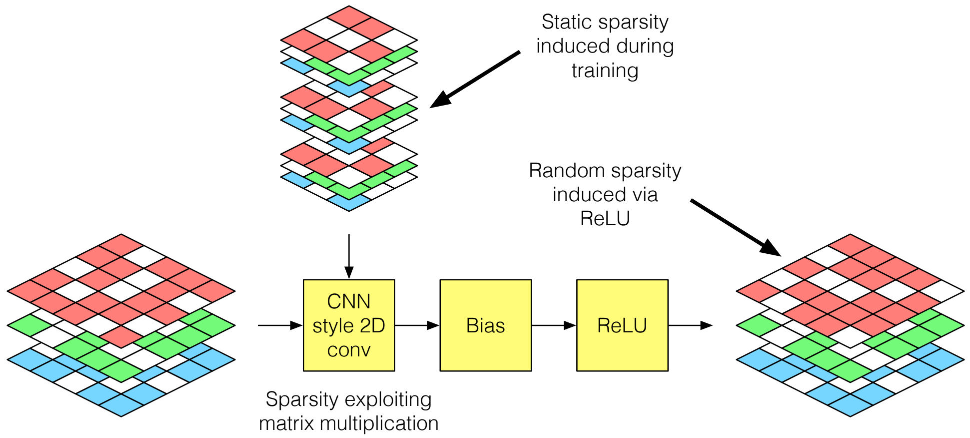


71.1% top 1 classification accuracy on the ImageNet 2012 validation
(0.5% better than the real MobileNetV1)

xNN Design – Sparisty

Sparsification Improves Performance

Assuming that the hardware can take advantage of it



- Sparsification improves performance via reducing memory, data movement and the required number of computations
- Examples
  - Structured sparsity in filter coefficients
    - Specified in the network design
    - Grouping in channel, separation in row col
  - Random sparsity in feature maps
    - Typically from the ReLU operator
  - Random sparsity in filter coefficients
    - Via pruning a dense network
    - Or training a sparse network

We've already seen this type of sparsity

We'll focus on this type of sparsity here

Prune A Dense xNN Vs Train A Sparse xNN

- Basic pruning strategy
  1. Train a dense network
  2. Prune weights to create a sparse network
  3. Re train the sparse network starting from the final weights in the previous step
  4. Repeat to step 2 (optional)

- Basic sparse training strategy
  1. Train a dense network (optional)
  2. Prune weights to create a sparse network
  3. Train the sparse network starting from the initial weights in the dense network
  4. Repeat to step 2 (optional)

Which to use?

- Pruning achieves a higher level of accuracy than sparse training but has the downside of taking more time to train (algorithm dependent)
- Additional training time isn't that big a deal for embedded sized networks but accuracy is very important
- Additional training time may be a big deal for very large networks
- Mentally it's ok to think of things as a spectrum from pruning to sparse training that all the techniques fit within

xNN Design – Sparisty – Pruning A Dense Network

One Shot Vs Iterative Pruning

- Basic pruning strategy
  1. Train a dense network
  2. Prune weights to create a sparse network
  3. Re train the sparse network starting from the final weights in the previous step
  4. Repeat to step 2

- Iterative pruning achieves a higher level of accuracy than one shot pruning
  - It's most common to remove a fixed amount of weights per iteration to achieve the target level of sparsity

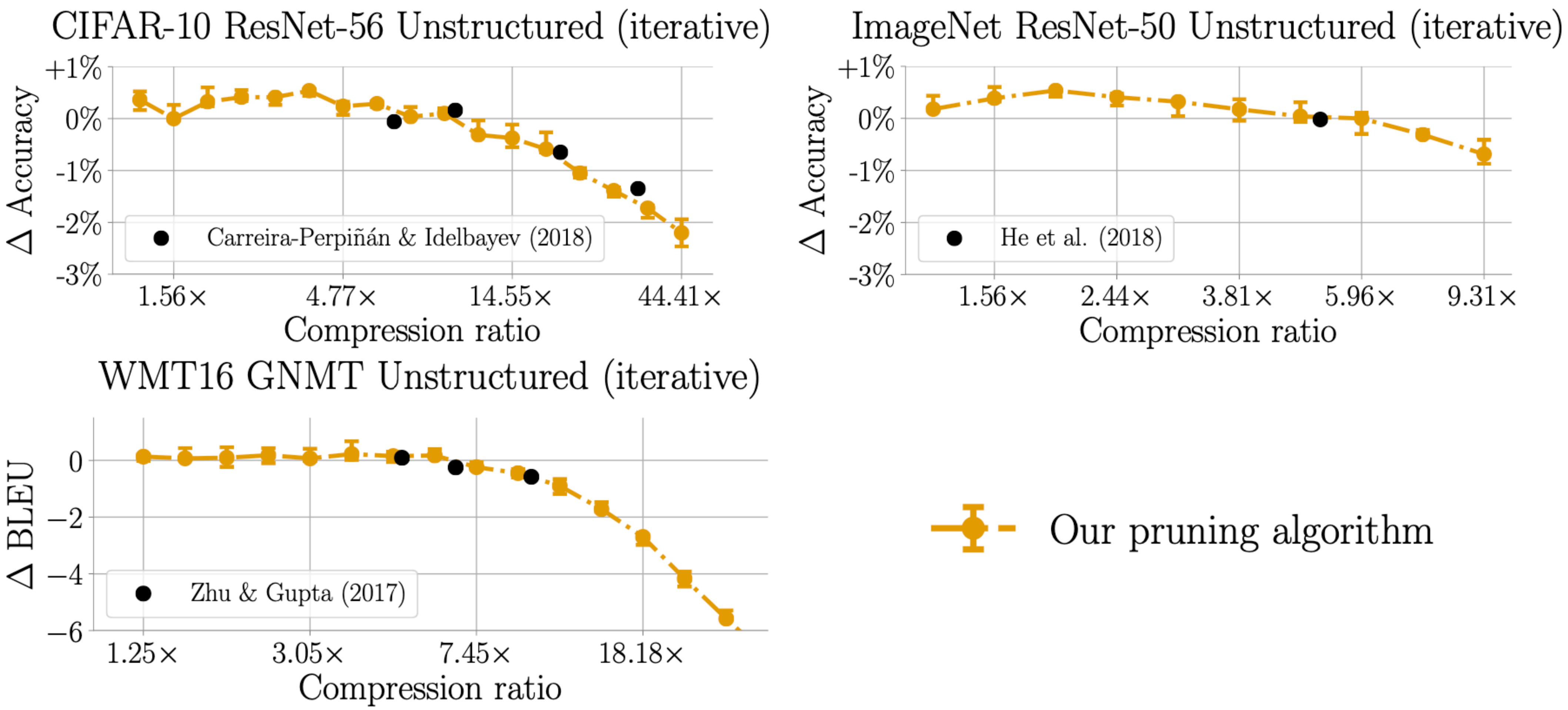
One shot pruning removes all the weights at once (no step 4) required to achieve the target level of sparsity (e.g., remove 80% of weights)

Iterative pruning removes a fraction of the weights at each iteration until the target level of sparsity is achieved

- Can remove a fixed amount per iteration (e.g., 20% for 4 iterations → 80% of weights removed)
- Can remove a variable amount per iteration (e.g., 35% iteration 1, 25% iteration 2, 15% iteration 3, 5% iteration 4 → 80% of weights removed)
- Avoids complete and quasi layer collapse that single shot can suffer from

Gradual pruning is an alternative to one shot and iterative pruning where there's no pruning for some number of initial epochs, then pruning from 0% sparsity to target % sparsity for some number of middle epochs, followed by no additional pruning for the final epochs; recent papers have favored iterative pruning over gradual pruning

Accuracy Vs Sparsity



Accuracy Vs Sparsity

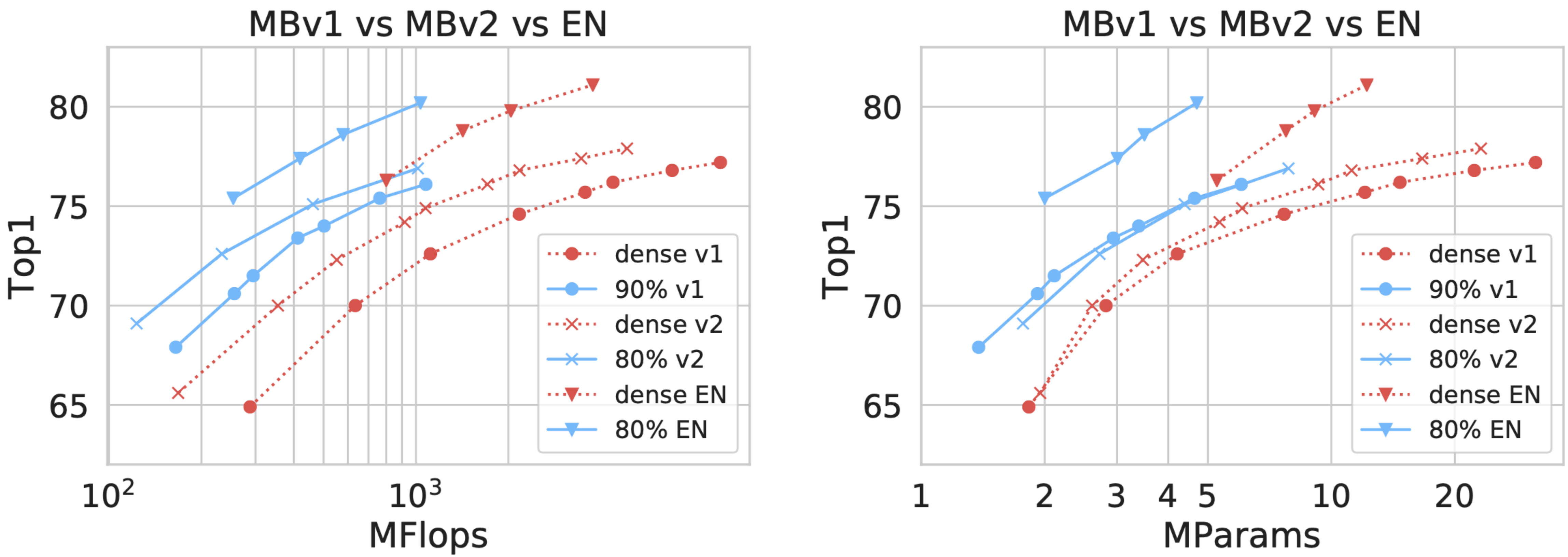


Figure from https://arxiv.org/abs/1911.09723

Where To Remove Weights From

- Basic pruning strategy
  1. Train a dense network
  2. Prune weights to create a sparse network
  3. Re train the sparse network starting from the final weights in the previous step
  4. Repeat to step 2

- Options for how many weights to remove per prune able operator
  - Uniformly per prune able operator (e.g., 50 prune able operators, remove 80% of the weights from each individually)
  - Non uniformly per prune able operator (e.g., remove 80% of the weights globally from the 50 prune able operators)

Remove weights from operators that lower to matrix ops, ideally matrix matrix multiplication

- CNN style 2D convolution
- Transformers (probably)
- Fully connected layers (maybe)
- RNN layers (maybe)

Avoid removing weights from some specific locations

- Stem (not many weights)
- Final encoder (maybe)
- Decoder (maybe)

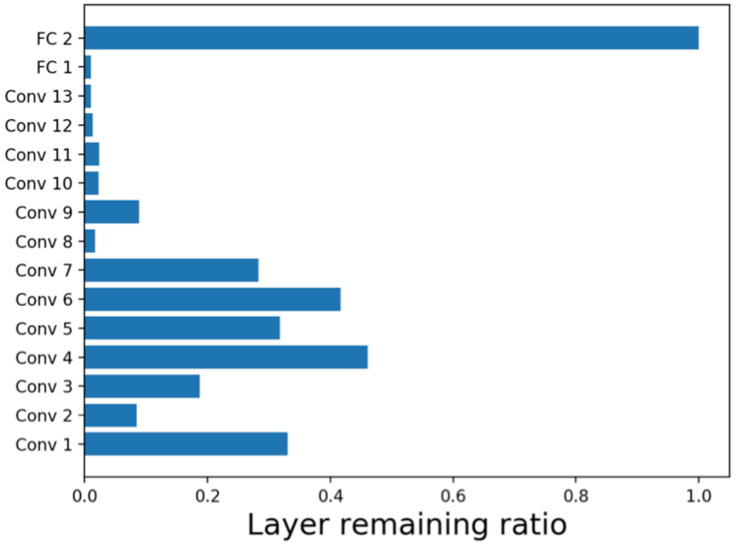
Avoid removing weights from some specific layers

- Depth wise conv layers (not many weights)
- Squeezer and excite layers (for reweighting)
- Bias (not many weights)
- Batch norm (not many weights)
- Probably anything but those on the ok list

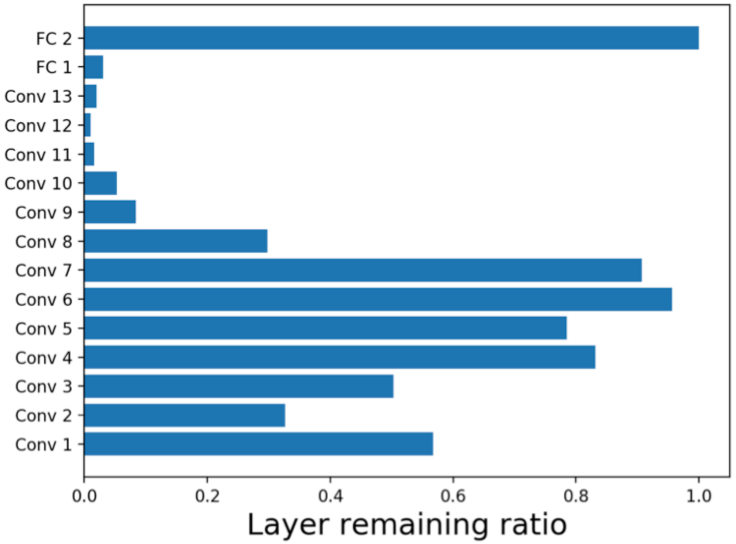
Avoid creating problems

- Poor gradient flow (probably)
- Channel loss via full pruning (probably)

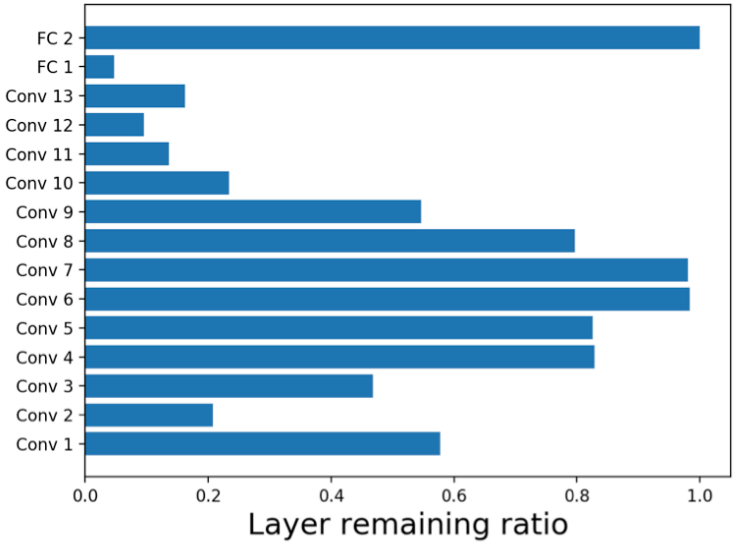
Per Layer Pruning



(a) $\alpha = 10^{-5}$, 6.66% remaining



(b) $\alpha = 10^{-6}$, 15.47% remaining



(c) $\alpha = 10^{-7}$, 35.35% remaining

Figure from https://arxiv.org/abs/2005.06870

RegNetX Params And MACs

- Number of params and MACs shown for RegNetX-800MF modified to have only 1 repeat per block (to better show the number of params and MACs in a block)
- Observe that the number of MACs per block is ~ similar but the number of params varies widely
  - Few in the initial blocks
  - Many in the later blocks
- This implies pruning needs to target both blocks with many AND blocks with few parameters if the goal is a reduction in the number of MACs

	Level 1	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**MACs**	**Filter Cx**
Repeats	1	2	2	2	2	463947008	3150944
Channels	32	64	128	288	672		
Groups	1	4	8	18	42		
Stride	2	2	2	2	2		

	Repeat	**Out H**	**Out W**	**No**	**Ni**	**Fr**	**Fc**	**S**	**G**	**MACs**	**Filter Cx**
Input		224	224	3							
Level 1 stem											
Conv, BN, ReLU	1	112	112	32	3	3	3	2	1	10838016	864
Level 2 transition											
Split right											
Conv, BN, ReLU	1	112	112	64	32	1	1	1	1	25690112	2048
Conv, BN, ReLU	1	56	56	64	64	3	3	2	4	28901376	9216
Conv, BN	1	56	56	64	64	1	1	1	1	12845056	4096
Split left											
Conv, BN	1	56	56	64	32	1	1	2	1	6422528	2048
Add, ReLU	1	56	56	64	64						
Level 2 block											
Split right											
Conv, BN, ReLU	1	56	56	64	64	1	1	1	1	12845056	4096
Conv, BN, ReLU	1	56	56	64	64	3	3	1	4	28901376	9216
Conv, BN	1	56	56	64	64	1	1	1	1	12845056	4096
Split left											
Identity	1	56	56	64	64						
Add, ReLU	1	56	56	64	64						
Level 3 transition											
Split right											
Conv, BN, ReLU	1	56	56	128	64	1	1	1	1	25690112	8192
Conv, BN, ReLU	1	28	28	128	128	3	3	2	8	14450688	18432
Conv, BN	1	28	28	128	128	1	1	1	1	12845056	16384
Split left											
Conv, BN	1	28	28	128	64	1	1	2	1	6422528	8192
Add, ReLU	1	28	28	128	128						
Level 3 block											
Split right											
Conv, BN, ReLU	1	28	28	128	128	1	1	1	1	12845056	16384
Conv, BN, ReLU	1	28	28	128	128	3	3	1	8	14450688	18432
Conv, BN	1	28	28	128	128	1	1	1	1	12845056	16384
Split left											
Identity	1	28	28	128	128						
Add, ReLU	1	28	28	128	128						
Level 4 transition											
Split right											
Conv, BN, ReLU	1	28	28	288	128	1	1	1	1	28901376	36864
Conv, BN, ReLU	1	14	14	288	288	3	3	2	18	8128512	41472
Conv, BN	1	14	14	288	288	1	1	1	1	16257024	82944
Split left											
Conv, BN	1	14	14	288	128	1	1	2	1	7225344	36864
Add, ReLU	1	14	14	288	288						
Level 4 block											
Split right											
Conv, BN, ReLU	1	14	14	288	288	1	1	1	1	16257024	82944
Conv, BN, ReLU	1	14	14	288	288	3	3	1	18	8128512	41472
Conv, BN	1	14	14	288	288	1	1	1	1	16257024	82944
Split left											
Identity	1	14	14	288	288						
Add, ReLU	1	14	14	288	288						
Level 5 transition											
Split right											
Conv, BN, ReLU	1	14	14	672	288	1	1	1	1	37933056	193536
Conv, BN, ReLU	1	7	7	672	672	3	3	2	42	4741632	96768
Conv, BN	1	7	7	672	672	1	1	1	1	22127616	451584
Split left											
Conv, BN	1	7	7	672	288	1	1	2	1	9483264	193536
Add, ReLU	1	7	7	672	672						
Level 5 block											
Split right											
Conv, BN, ReLU	1	7	7	672	672	1	1	1	1	22127616	451584
Conv, BN, ReLU	1	7	7	672	672	3	3	1	42	4741632	96768
Conv, BN	1	7	7	672	672	1	1	1	1	22127616	451584
Split left											
Identity	1	7	7	672	672						
Add, ReLU	1	7	7	672	672						
Decoder											
Global avg pool	1	1	1	672	672						
Linear	1	1	1	1000	672					672000	672000
Total										463947008	3150944

Which Weights To Remove

- Basic pruning strategy
  1. Train a dense network
  2. Prune weights to create a sparse network
  3. Re train the sparse network starting from the final weights in the previous step
  4. Repeat to step 2

After pruning, do you do anything to the remaining weights?

- Maybe nothing?
- Maybe rescale them to keep the total energy in an operator the same?
- The smaller the fraction of weights that are pruned per iteration the less this matters
- The longer the re training the less this matters
- Network structures with a lot of branching and combining are possibly affected more

Ideally the set of weights that minimizes the validation accuracy loss in the re trained network relative to the original dense network

- Output contribution is a step removed proxy for this

In practice, it's common to simply remove the weights with the smallest magnitude

- Probably want to account for per channel batch norm scaling in determining the true magnitude

Can encourage small mag weights during training

- L2 regularization is commonly used to improve generalization
- L1 regularization pushes small weights towards 0 a little faster
- L0 regularization is really what you want, can get it via a relaxation method

How To Retrain

- Basic pruning strategy
  1. Train a dense network
  2. Prune weights to create a sparse network
  3. Re train the sparse network starting from the final weights in the previous step
  4. Repeat to step 2

- Learning rate rewinding achieves a higher level of accuracy than fine tuning

Fine tuning uses the ~ final learning rate for some number of epochs

Learning rate rewinding uses the learning rate schedule that starts from an earlier epoch in training and continues to the final epoch

- Common to use the learning rate schedule from the very start of training (i.e., do a completely new training starting from the final weights in the previous step)
- There are some similarities with this and cyclical learning rate schedules

Putting It All Together

- Basic pruning strategy
  1. Train a dense network
     - 5 epochs linear warmup 100 epochs cosine decay
     - L2 regularization
  2. Prune weights to create a sparse network
     - Only prune non grouped conv layers
     - Avoid the first and last encoder layer and the decoder
     - Globally or per operator remove 20% of the weights with the smallest magnitudes (weight * batch norm scale) that don’t remove full channels
  3. Re train the sparse network starting from the final weights in the previous step
     - Same schedule and regularization as dense network training
  4. Repeat to step 2
     - 4 times to achieve 80% sparsity

xNN Design – Sparisty – Training A Sparse Network

Why People Are Interested In Sparse Training

- While the pruning strategy described in the previous section is probably the better strategy for most cases, it's worthwhile to understand this approach too

- The nice part of training a sparse network is that promise of faster training
  - This is likely of less relevance when the networks are relatively small (in the grand scheme of things)
  - It's not clear if faster training is realized in practice
  - Regardless, this section will include some of the key strategies for training sparse networks

Challenges

- Poor gradient flow at initialization
  - The extreme form of this is layer collapse (more of an issue for very high sparsity levels and unsophisticated methods of choosing which weights to prune)

- Poor gradient flow during training

Training Options

- The lottery ticket hypothesis: sparse subnetworks exist at initialization that can be trained to match the generalization performance of the corresponding dense network
  - The sparse subnetwork is referred to as the lottery ticket

- So the fundamental question is how to efficiently find the sparse subnetwork

- Different sparse training options can be classified based on when you find the lottery tickets
  - After training (I know, weird)
  - During training
  - Before training

Finding Lottery Tickets After Training

- This is the initial lottery ticket work
  - These sparse subnetworks were found by training a dense network, pruning, and initializing the resulting sparse networks with the same weights that the original dense network was pruned with

- Theoretical analysis found
  - Lottery tickets don't improve the gradient flow at initialization or the gradient flow during training
  - Lottery tickets ~ re learn the pruning solution

- Rebuttals of the initial work
  - Others observed that the initial formulation did not generalize to larger networks with higher learning rates

- Rebuttals to the rebuttals (sort of)
  - Late rewinding was proposed to address this
  - The basic idea is not to initialize the sparse network with the initial weights, but rather the weights after a few epochs of training (effectively, more likely to be in the final basin)

Weight Rewinding

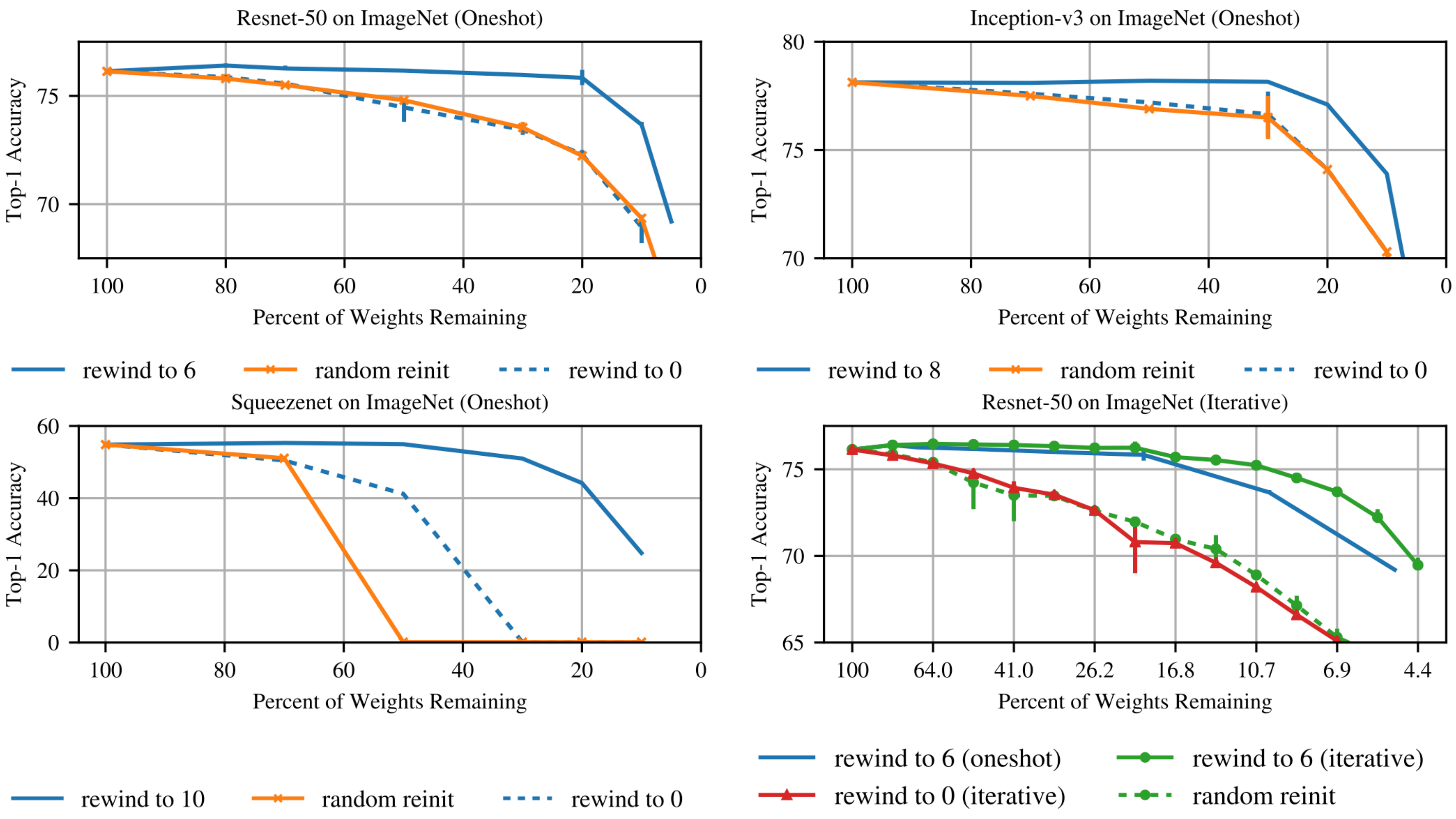


Figure from https://arxiv.org/abs/1903.01611

Basin Of Convergence

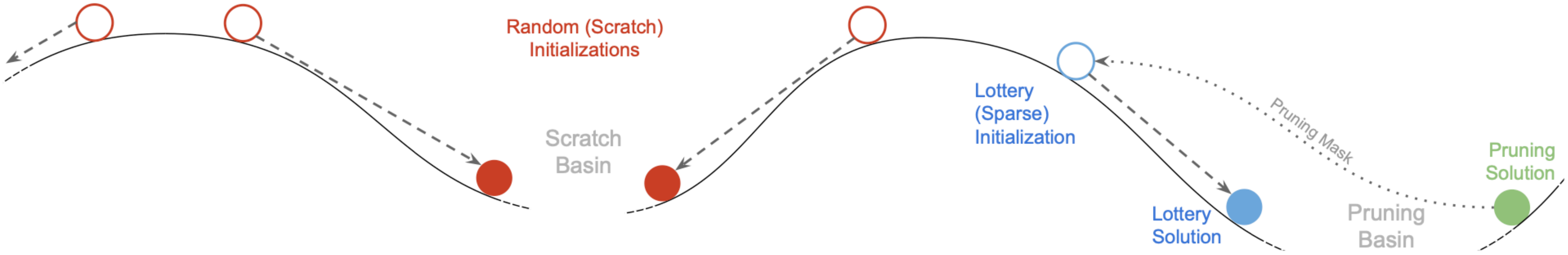


Figure from https://arxiv.org/abs/2010.03533

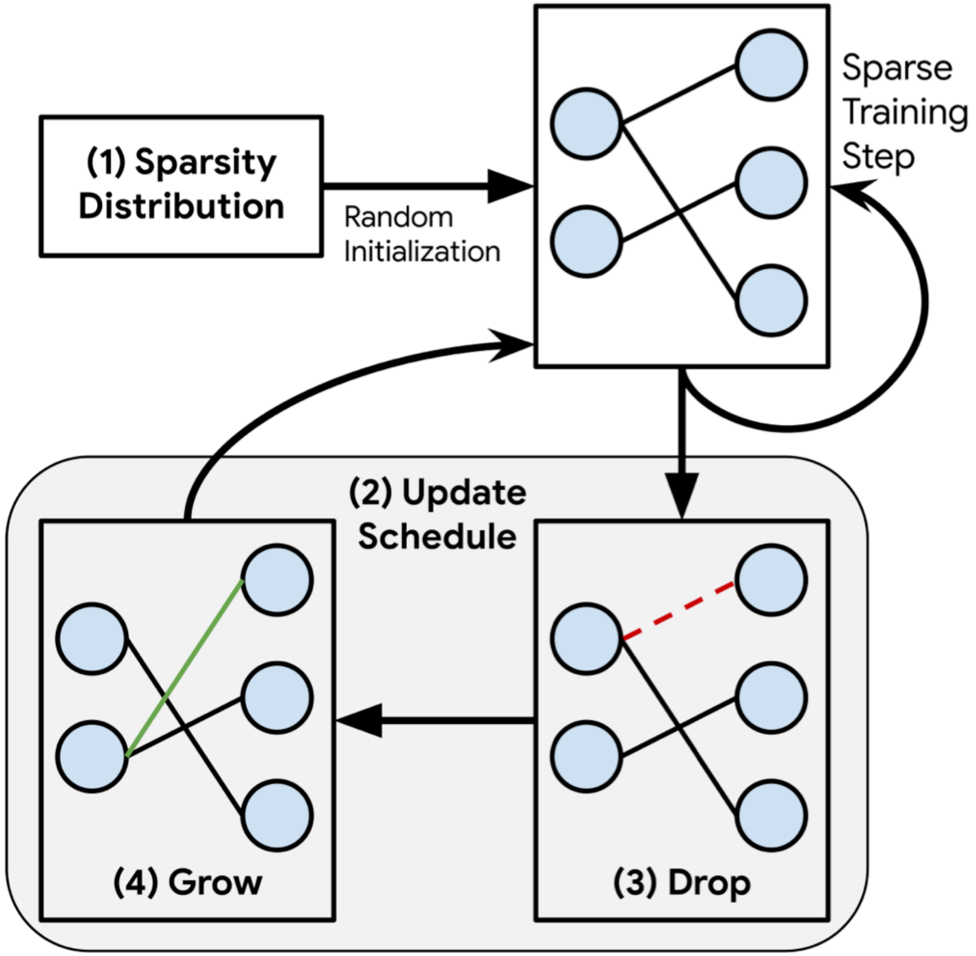
Finding Lottery Tickets During Training

- This is the dynamic sparse training work
  - The basic idea is to train a network y = f(w, m, x) to find both the weights w and a 0/1 mask m that's applied to the weights
  - A key is the need to be able to both flip mask values from 1 to 0 (pruning, typically small magnitude) and also flip mask values from 0 to 1 (this is new, effectively adding connections back in)
  - DST methods that generalize best have improved gradient flow
- Various strategies for flipping mask values from 0 to 1
  - Sparse evolutionary training (SET) chooses new connections randomly
  - RigL chooses new connections with high gradient magnitude and matches pruning performance with sufficient training time; the newly chose connections seem to improve the gradient flow early in the training
- The dynamic sparse training paper uses different functions in the forward and backward directions for flipping mask values from 0 to 1
  - The forward direction uses the mask
  - The backward direction ignores the mask (i.e., effectively the straight through estimator) such that all weights are always updated

A nice part of the dynamic sparse training paper is that pruning is done continuously during training (vs at discrete discontinuous steps); the continuous sparsification paper also does this

The downside is the need to flow gradient info backwards through a function that doesn't match the forward function; some of the quantization training tricks can help out here

RigL With A Training Budget



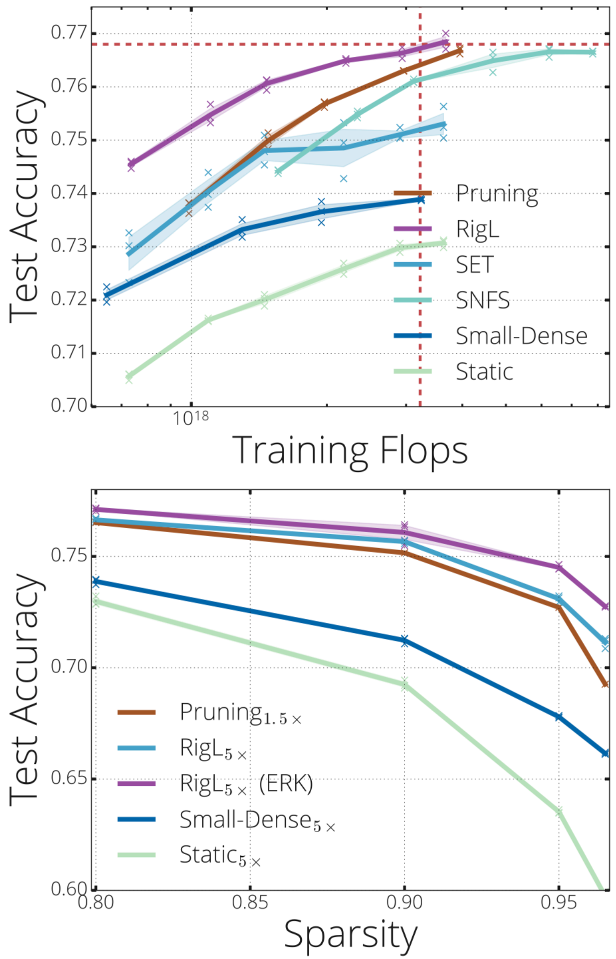


Figure from https://arxiv.org/abs/1911.11134

Finding Lottery Tickets Before Training

- Basic strategy (note there is no data / training)
  - Define a data independent loss based only on parameters
  - Define synaptic saliency for any parameter as above; note that this depends on the data independent loss e
  - For some number of iterations
    - Compute the data independent loss e
    - Compute the synaptic saliency S(w)
    - Mask parameters with the lowest synaptic saliency to 0
    - Repeat

- Example error e and synaptic saliency S(w) formulations
  - e based on the elementwise product of weights across layers (the equation in the paper is somewhat ambiguous in its definition of the weight representation)
  - $S(w) = (\partial e / \partial w) \odot w$

Commentary

Methods

- Iterative synaptic flow pruning (SynFlow)
- Other: random, magnitude, SNIP (scores weights based on the gradient of the training loss), GraSP (scores weights based on the grad Hessian product)

Layer size and synaptic saliency are inversely proportional

- If a layer has a lot of weights, then the synaptic saliency of those weights is less
- This explains why gradient based methods and mag based methods without iteration tend to prune the largest layer first leading to layer collapse
- It also points to the benefit of iterative pruning – after removing weights from a large layer it’s smaller so the remaining weights after re training tend to have a higher score and are less likely to be pruned

Layer Size Vs Synaptic Saliency

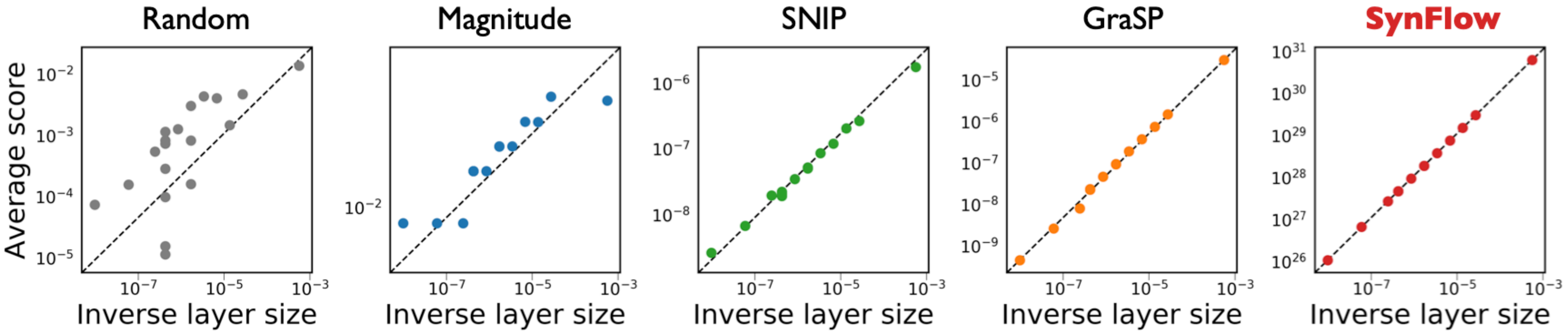


Figure from https://arxiv.org/abs/2006.05467

Accuracy Vs Sparsity

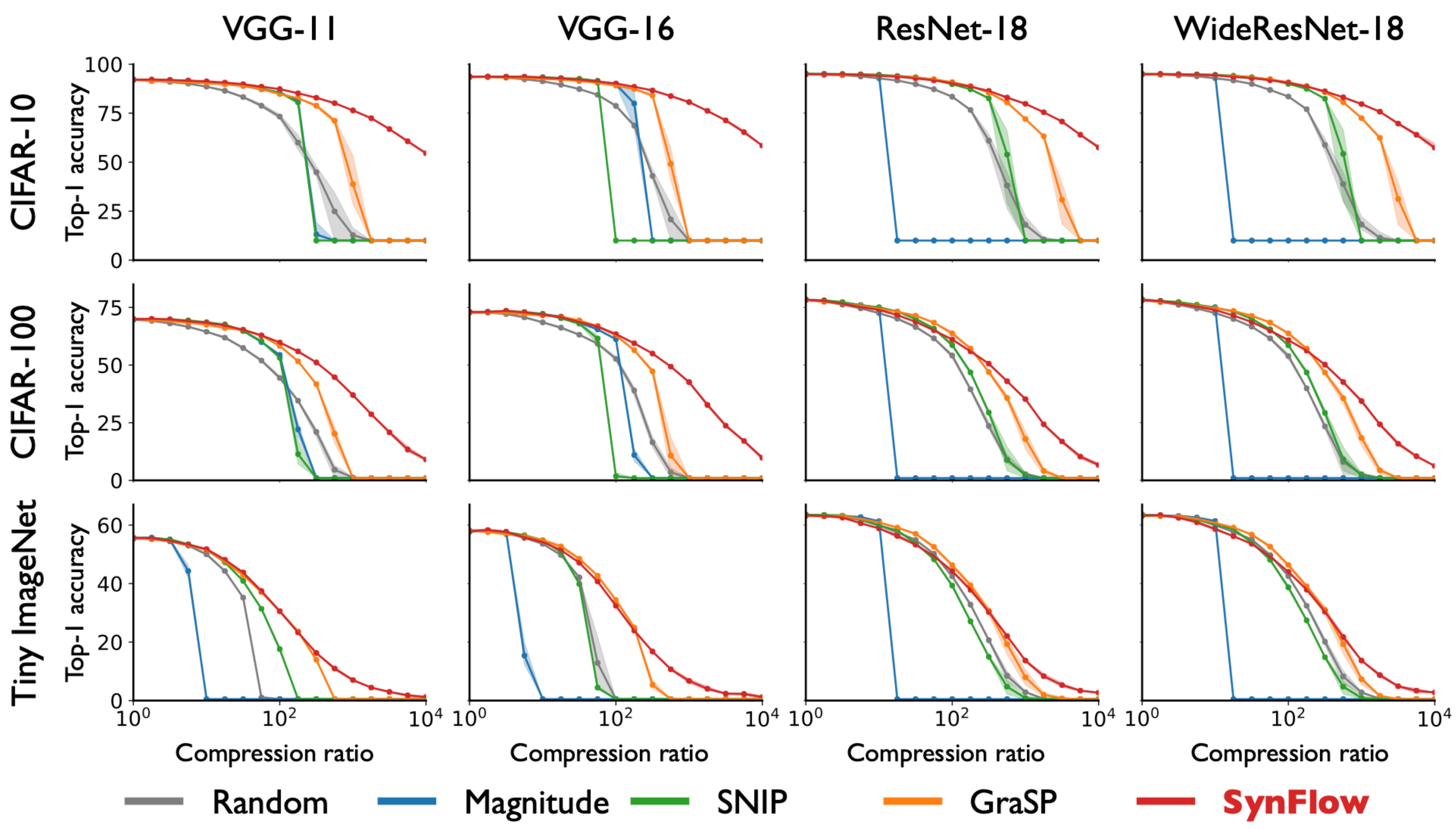


Figure from https://arxiv.org/abs/2006.05467

xNN Design – Compression

Compression Improves Performance

Assuming that the hardware can take advantage of it

- Compression improves performance via reducing memory and communication requirements
- Compression exploits redundancy within or across symbols
- Examples
  - Feature maps (dynamic)
  - Parameters (static)
  - Gradients (dynamic)

Coding Intuition Using A Toy Example

To transmit the message x_k with the pmf and code in ex 1 it's expected to take 16*(1/16)*4 = 4 bits
To transmit the message x_k with the pmf and code in ex 2 it's expected to take 1*(1/2)*1 + 15*(1/30)*5 = 3 bits

Random variable $X(s) = x_k$	Probability mass function ex 1 $p_X(X(s) = x_k)$	Binary coding ex 1 $C_X(x_k) = c_k$	Probability mass function ex 2 $p_X(X(s) = x_k)$	Binary coding ex 2 $C_X(x_k) = c_k$
0	1/16	0000	1/2	0
1	1/16	0001	1/30	10001
2	1/16	0010	1/30	10010
3	1/16	0011	1/30	10011
4	1/16	0100	1/30	10100
5	1/16	0101	1/30	10101
6	1/16	0110	1/30	10110
7	1/16	0111	1/30	10111
8	1/16	1000	1/30	11000
9	1/16	1001	1/30	11001
10	1/16	1010	1/30	11010
11	1/16	1011	1/30	11011
12	1/16	1100	1/30	11100
13	1/16	1101	1/30	11101
14	1/16	1110	1/30	11110
15	1/16	1111	1/30	11111

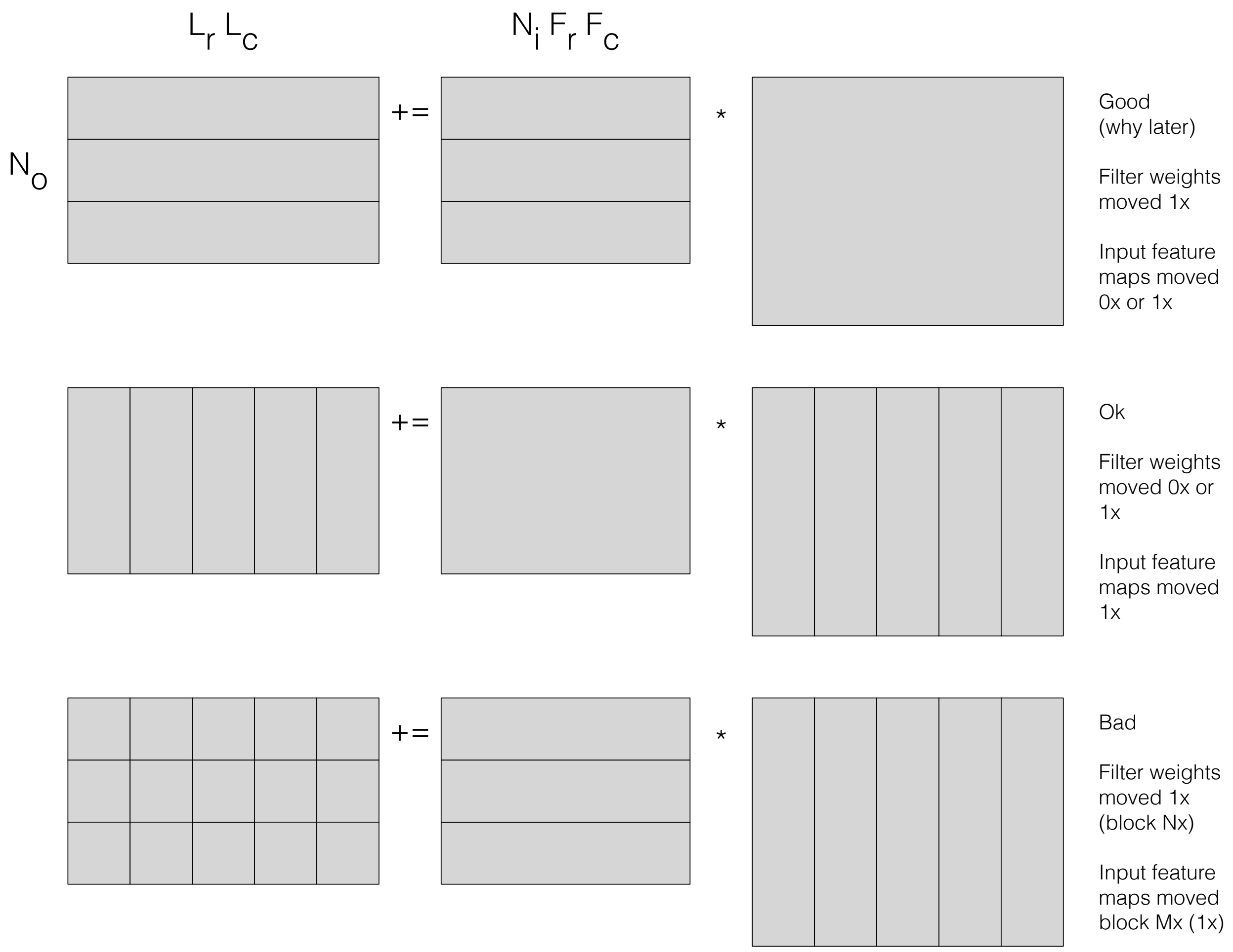
xNN Design – Sizing

Appropriate Sizing Improves Performance

Assuming that the network is size to better match the hardware

- Want to match the network size to the hardware

- Common issues
  - Over or under saturating computation
  - Over or under saturating memory and communication

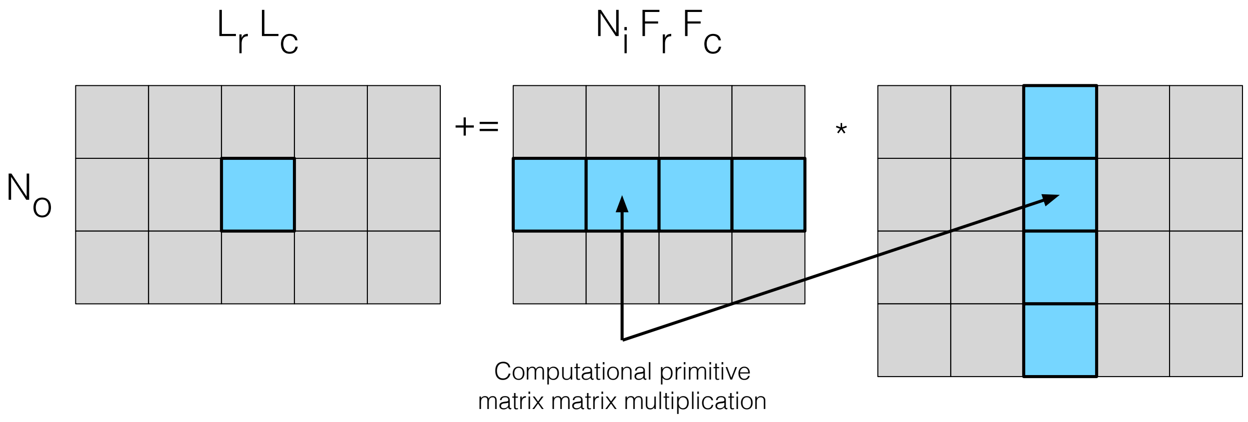


Typically in the beginning of the network feature maps dominate memory and at the end of the network filter coefficients dominate memory

Appropriate Sizing Improves Performance

Assuming that the network is size to better match the hardware

- Want to match the network size to the hardware
- Common issues
  - Over or under saturating computation
  - Over or under saturating memory and communication



Inefficiently tiling CNN style 2D convolution with a matrix multiplication primitive under utilizes hardware

xNN Software

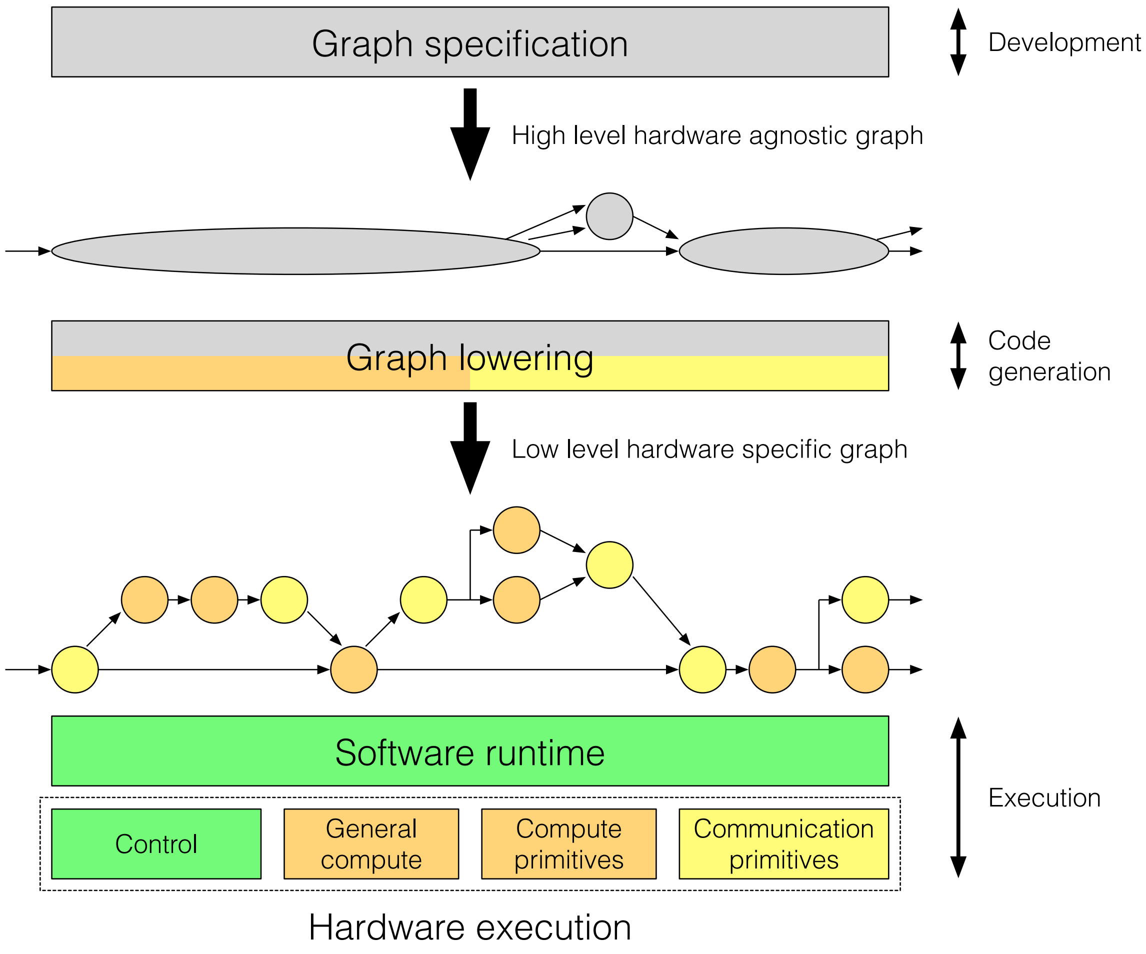
Flow

- Software maps xNN designs to hardware

- Software components
  - xNN graph specification
  - xNN graph lowering
  - xNN graph execution (software)

xNN Graph Specification

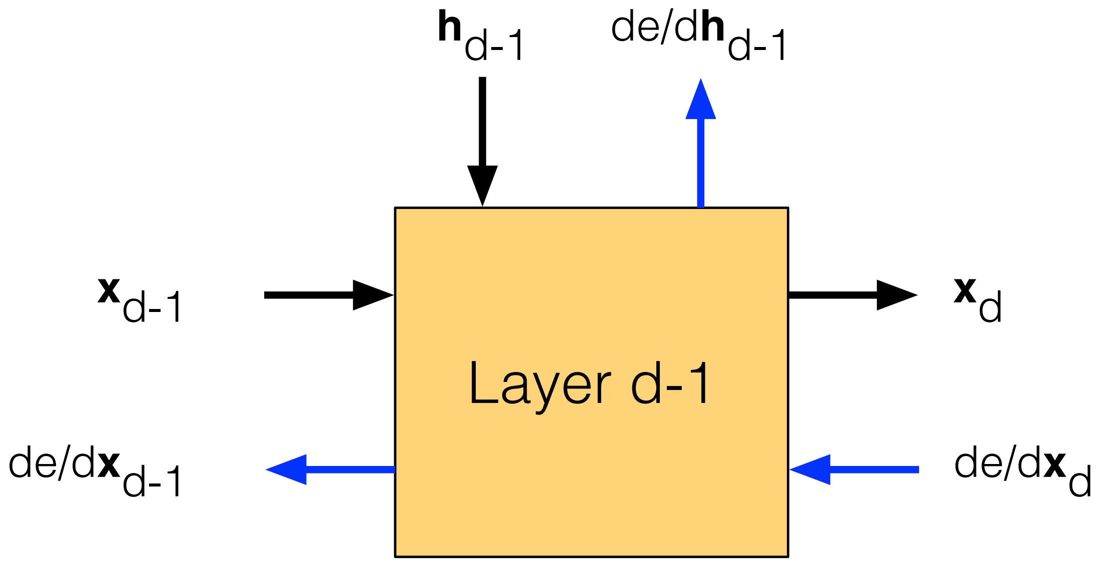
- High level hardware agnostic (or aware) graph specification
- Every node in the graph
  - Has a mapping from input to output optionally controlled via a set of parameters
  - Has a mapping from sensitivity of the error with respect to the output to sensitivity of the error with respect to the input
  - Optionally has a mapping from sensitivity of the error with respect to the output to sensitivity of the error with respect to the parameters if there are trainable parameters



xNN Graph Specification

- High level hardware agnostic (or aware) graph specification

- Every node in the graph
  - Has a mapping from input to output optionally controlled via a set of parameters
  - Has a mapping from sensitivity of the error with respect to the output to sensitivity of the error with respect to the input
  - Optionally has a mapping from sensitivity of the error with respect to the output to sensitivity of the error with respect to the parameters if there are trainable parameters



$$\begin{aligned}
\mathbf{x}_d &= \mathbf{f}_{d-1}(\mathbf{h}_{d-1}, \mathbf{x}_{d-1}) \\
de/d\mathbf{x}_{d-1} &= (d\mathbf{x}_d/d\mathbf{x}_{d-1})\,(de/d\mathbf{x}_d) \\
&= (d\mathbf{f}_{d-1}/d\mathbf{x}_{d-1})\,(de/d\mathbf{x}_d) \\
de/d\mathbf{h}_{d-1} &= (de/d\mathbf{x}_d)\,(d\mathbf{x}_d/d\mathbf{h}_{d-1})
\end{aligned}$$

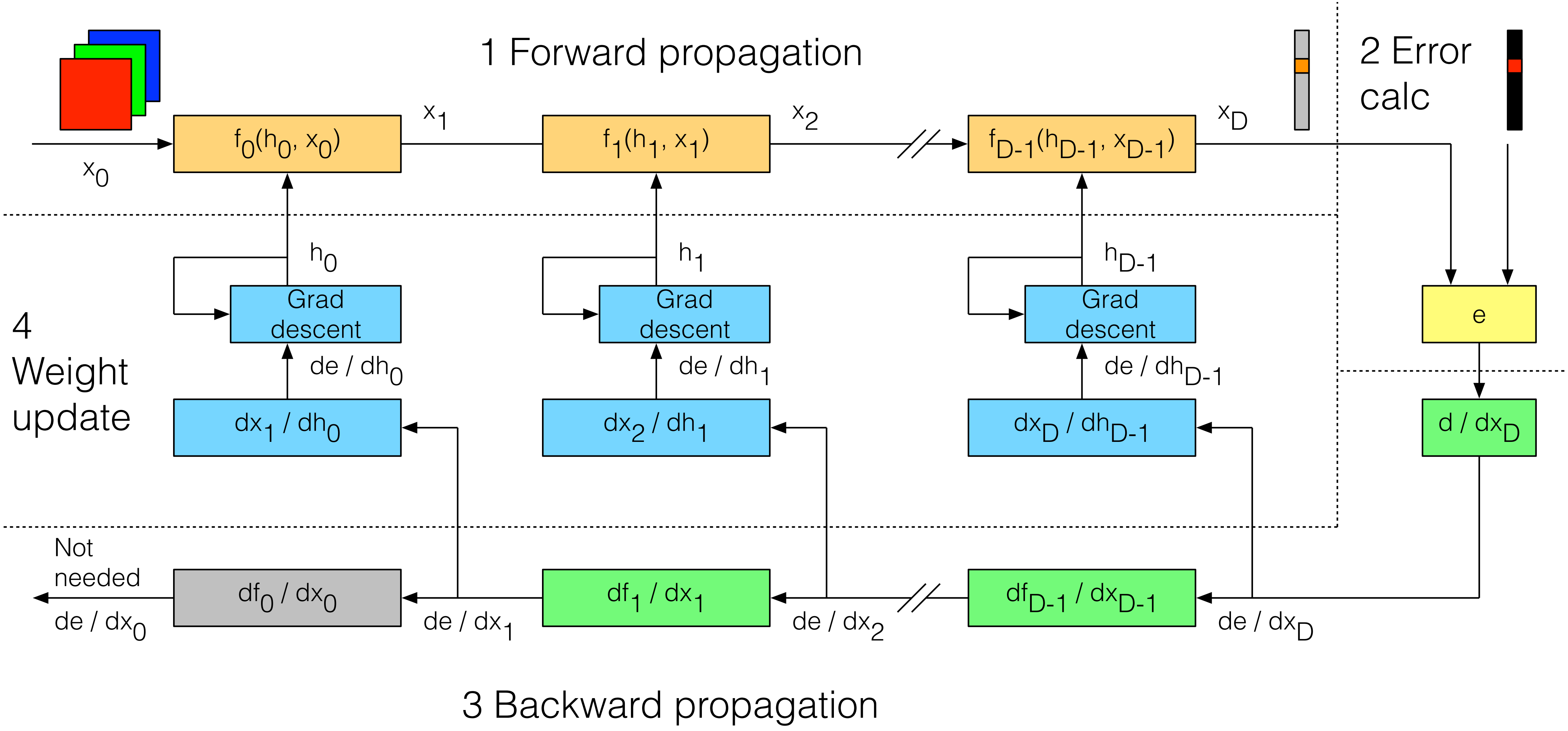
User And Software Specified Components

- [User] Specify the mapping from data to network output
- [User] Specify an error computation method and weight update method
- [Software] Use automatic differentiation with reverse mode accumulation to create a graph to propagate the sensitivity of the error with respect to the feature map from the output to the input of all nodes
- [Software] Use the multivariable chain rule to create graph nodes that map the sensitivity of the error with respect to the output feature map to the sensitivity of the error with respect to the parameters for all nodes with parameters
- [Software] Create graph nodes to update the parameters based on the sensitivity of the error with respect to the parameters based on the user's specified weight update method

User And Software Specified Components

- [User] Specify the mapping from data to network output
- [User] Specify an error computation method and weight update method
- [Software] Use automatic differentiation with reverse mode accumulation to create a graph to propagate the sensitivity of the error with respect to the feature map from the output to the input of all nodes
- [Software] Use the multivariable chain rule to create graph nodes that map the sensitivity of the error with respect to the output feature map to the sensitivity of the error with respect to the parameters for all nodes with parameters
- [Software] Create graph nodes to update the parameters based on the sensitivity of the error with respect to the parameters based on the user's specified weight update method

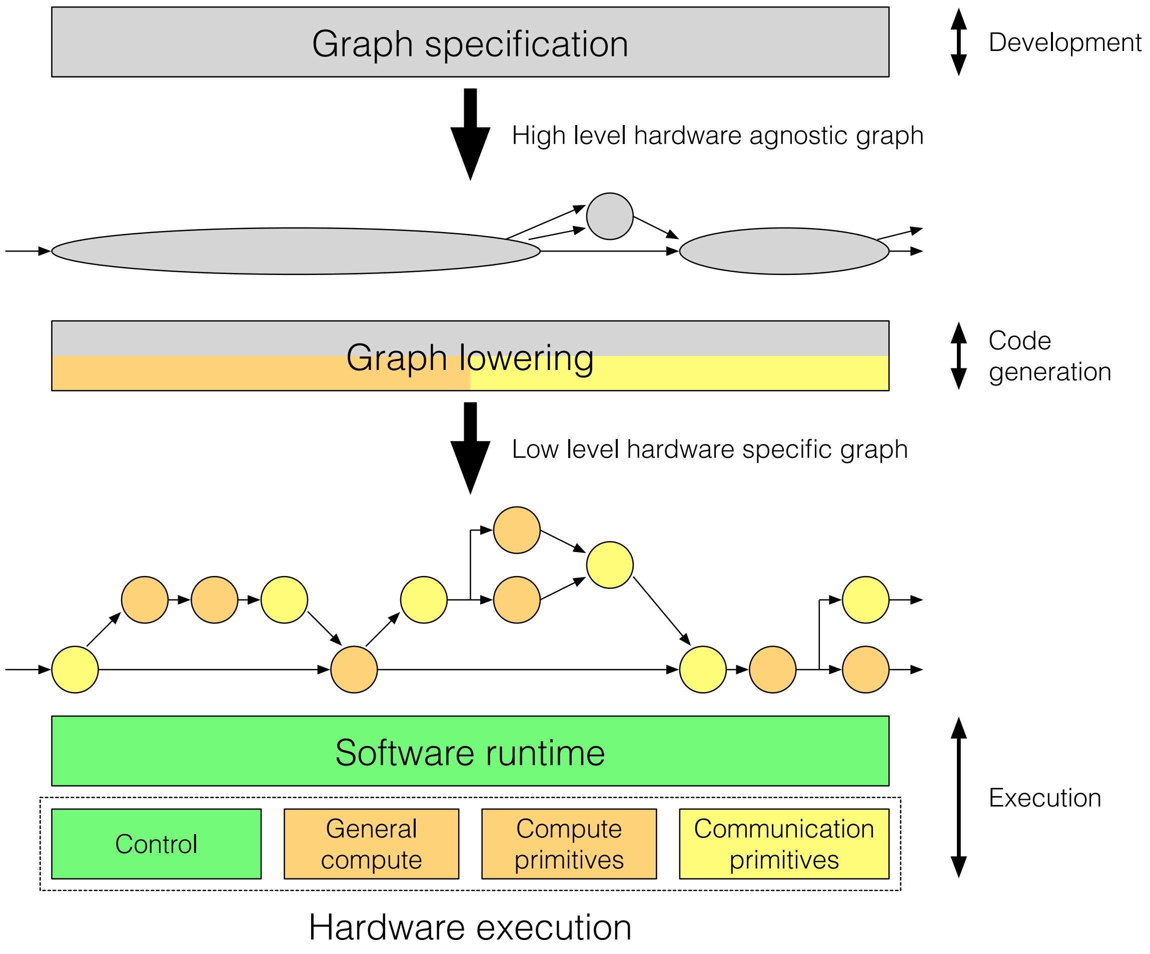
User And Software Specified Components



xNN Graph Lowering

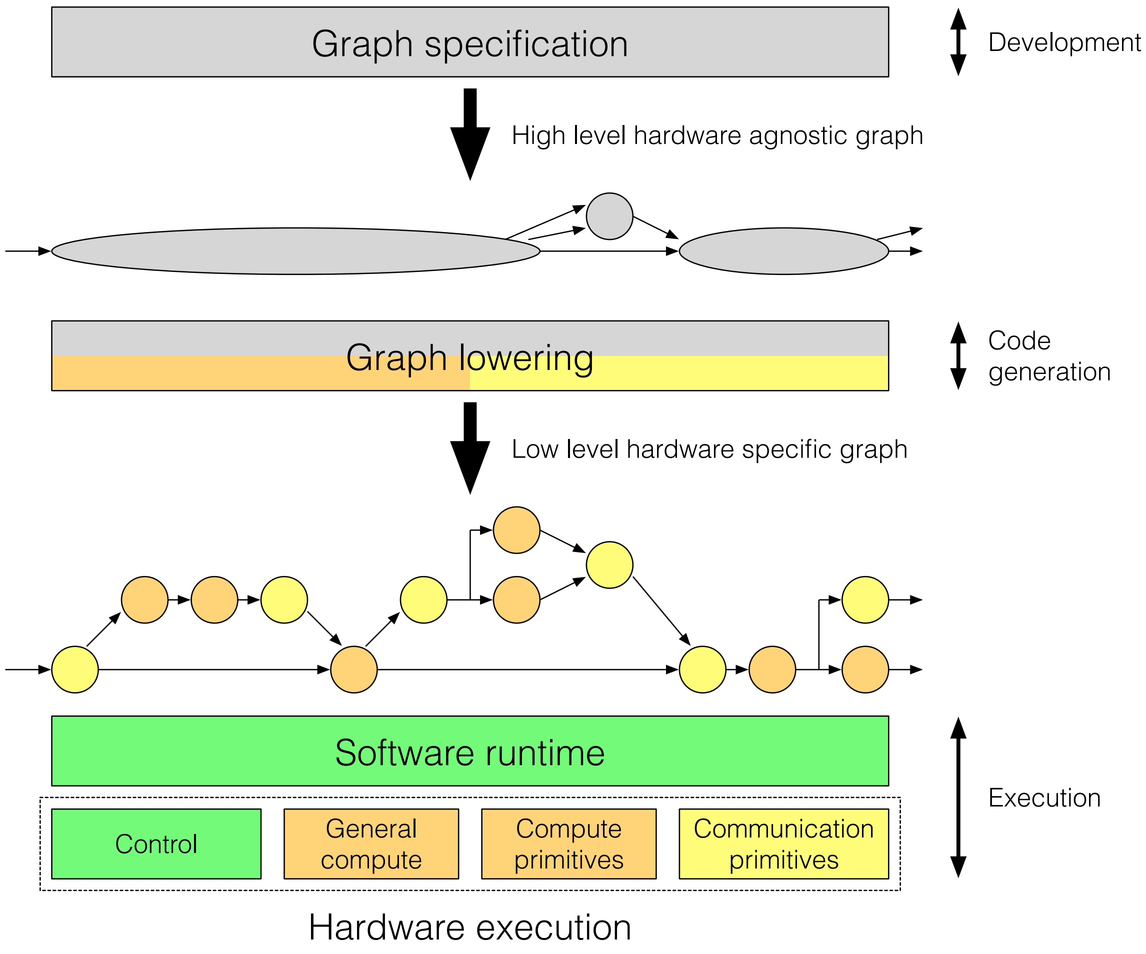
This is a non convex optimization problem, multiple iterations through this flow are common

- Domain agnostic hardware agnostic optimization
  - Remove unneeded edges and nodes required for specified input / output
  - Constant folding and constant propagation
- Domain specific hardware agnostic optimization
  - xNNs: remove dropout and scale associated weight layers
  - xNNs: absorb batch norm into convolution and create a bias term
- Domain specific hardware aware optimization
  - xNNs: transform data layouts (tensor ordering)
  - xNNs: node fusion, tiling and grouping
  - xNNs: post training quantization
- Domain agnostic hardware specific code generation
  - Memory planning for all tensors
  - Data movement and compute strategy selection for each node
  - Code generation for selected strategy



xNN Graph Execution (Software)

- This is the role of the software runtime
- Initialization phase: tie addresses for dynamic tensors into the graph
  - Ex: input and output memory locations for the specific input image
  - Maybe a few other setup operations
- Execution phase: cycle through nodes
  - Making sure that all dependencies are satisfied
  - Running the node on the appropriate general compute, computational primitive or communication primitive



xNN Hardware

Question

- What is the best hardware design?
  - The brain is an existence proof of what can be computed with 20 W of power and 3 lbs of material
  - But it’s not a limit of what can be built
- Coding theorists were lucky (???) and Shannon gave them a limit (though it took a really long time to get there)
- Approximate limits are known for some small pieces of hardware design
  - Comparators
  - DRAM bit cells
  - Individual multipliers
- But it’s a more difficult question to answer in the context of a full system with many variables
  - However, we should always have this question in the back of our minds when designing hardware
  - Take a principled approach that intersects algorithm requirements with physics

Moore’s Law

- The number of transistors in an integrated circuit doubles every ~ 2 years at a constant cost
  - Previously a little faster
  - Now a little slower

- Note
  - Doesn’t say anything about speed
  - Doesn’t say anything about power

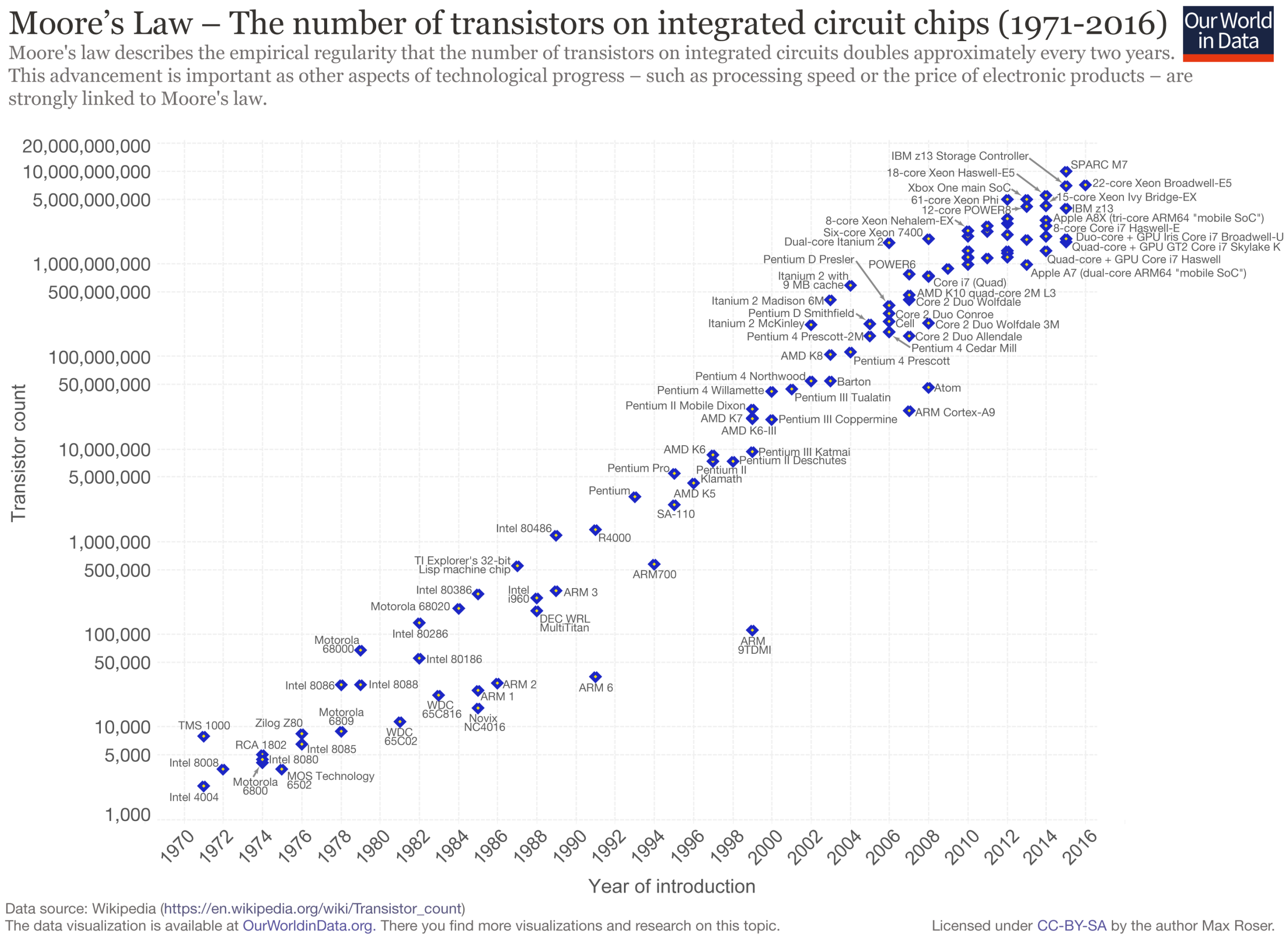


Figure from https://en.wikipedia.org/wiki/Moore%27s_law

Dennard Scaling

- Transistor power density used to be proportional to area but no longer is
  - It was from ~ 1974 – 2006 when energy was dominated by switching frequency (Dennard scaling)
  - But it no longer is (sadness)
  - The problem is that at smaller transistor sizes the threshold voltage and current leakage limits voltage scaling
  - Prior to ~ 2006 improvements in scaling feature sizes and voltage overwhelmed everything else
  - Now need better architecture designs to advance performance

Approximate physics

- L = transistor feature size
- V = voltage
- C = capacitance per transistor ($\propto$ L)
- D = area density ($\propto 1/L^2$)
- E = energy per transistor use ($\propto CV^2$)
- f = frequency ($\propto$ 1/L)
- P = power per area ($\propto$ DEf)

Approximate process technology

- In 1 generation L is scaled by ~ 0.7
- In 2 generations L is scaled by ~ 0.7*0.7 = 0.49 ≈ 1/2

2 gens with voltage scaling: L' = L/2, V' = V/2 and same area

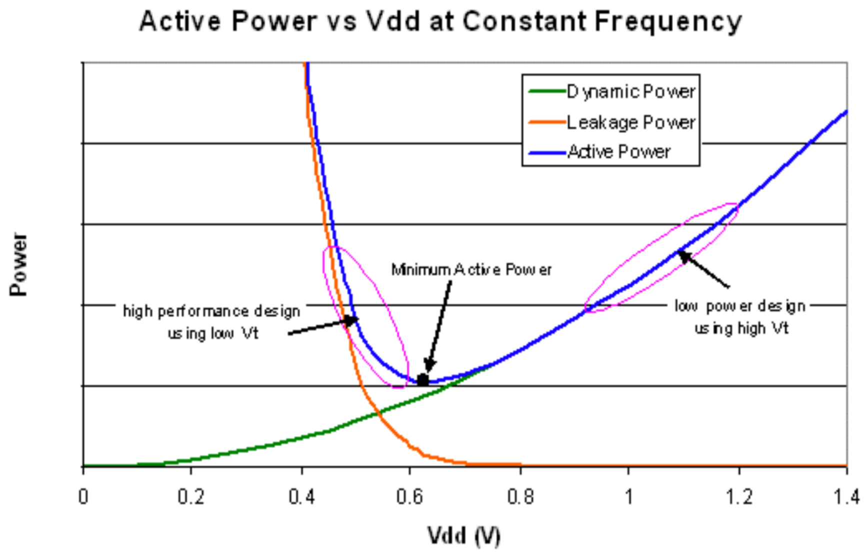
• C' = C/2	
• D' = 4D	4x transistors
• E' = E/8	
• f' = 2f	2x frequency
• P' = P	1x power

2 gens without voltage scaling: L' = L/2, V' = V and same area

• C' = C/2	
• D' = 4D	4x transistors
• E' = E/2	
• f' = 2f	2x frequency
• P' = 4P	4x power (**bad**)

Dark Silicon And Dark Memory

- Dark silicon
  - A consequence of the end of Dennard scaling
  - Only a fraction of a device can be active at one time because of increased energy per unit area vs power dissipation limits
  - This gets worse as process geometries continue to shrink
  - The result is that more and more of the device is off at any given time
  - Consequence: design accelerators to be as efficient as possible for key tasks



- Dark memory
  - A consequence of the end of Dennard scaling
  - Only a fraction of DRAM and local device memory can be active at one time because of increased energy per unit area vs power dissipation limits
  - This gets worse as process geometries continue to shrink
  - The result is that more and more of the memory is idle at any given time
  - Consequence: maximize data locality to minimize memory and data movement

Figure from https://www.design-reuse.com/articles/20296/power-management-leakage-control-process-compensation.html

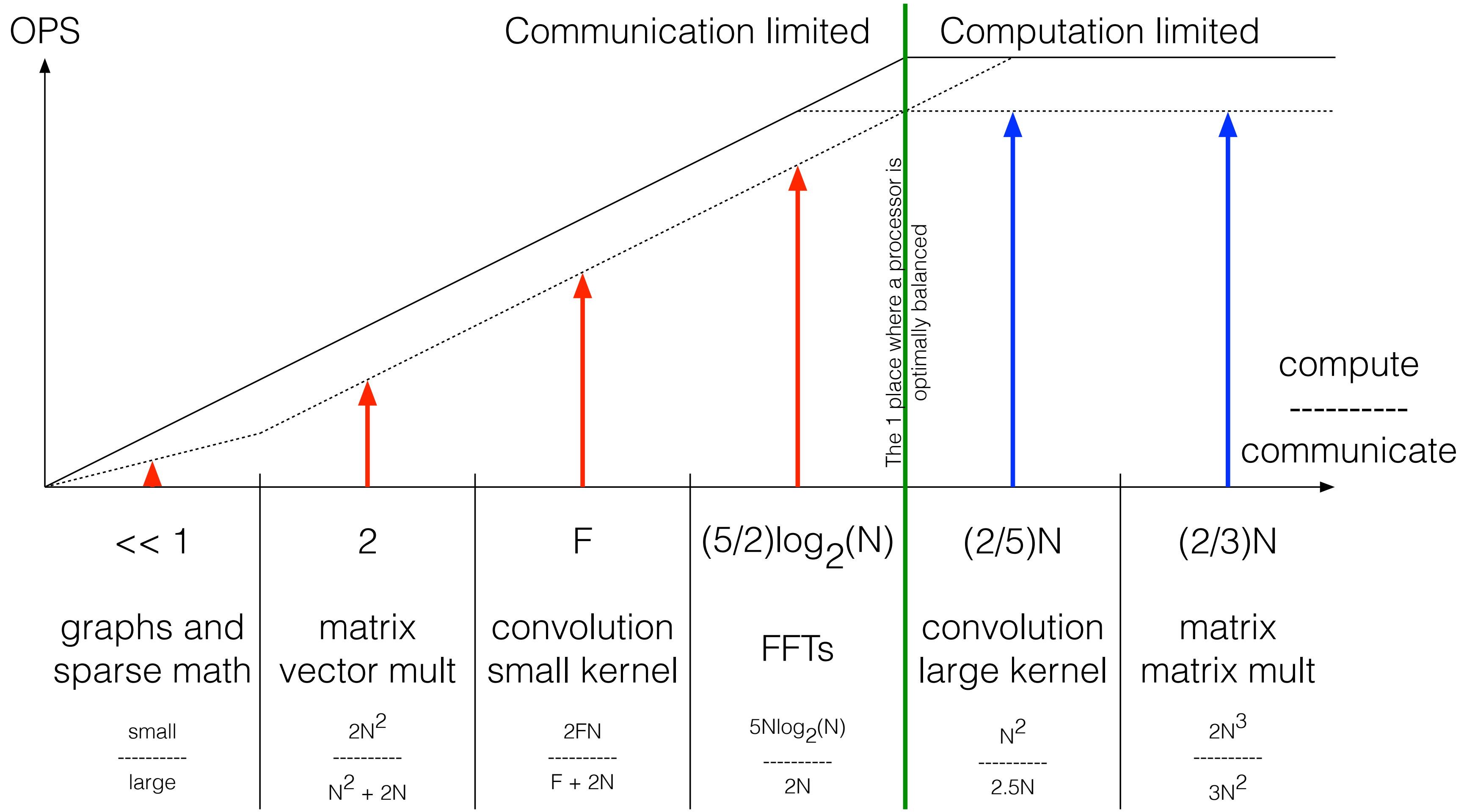
Power Is The Problem

- A list of what consumes power from most to least
  - Physical movement
  - Long distance communication
  - Off device communication
  - On device communication
  - Computation

Example power estimates in 28 nm (from a public presentation from ARM)

16 bit integer MAC	1	pJ
32 bit integer MAC	4	pJ
32 bit float MAC	6	pJ
64 bit float MAC	20	pJ
Read from on chip SRAM	1.5	pJ/B
Read from off chip DRAM	250	pJ/B
Wires 20 mm 50% transitions	7	pJ/B
Chip to chip parallel link	25	pJ/B
Chip to chip serial link	50	pJ/B

Roofline Model



Putting 1 And 1 And 1 And 1 Together

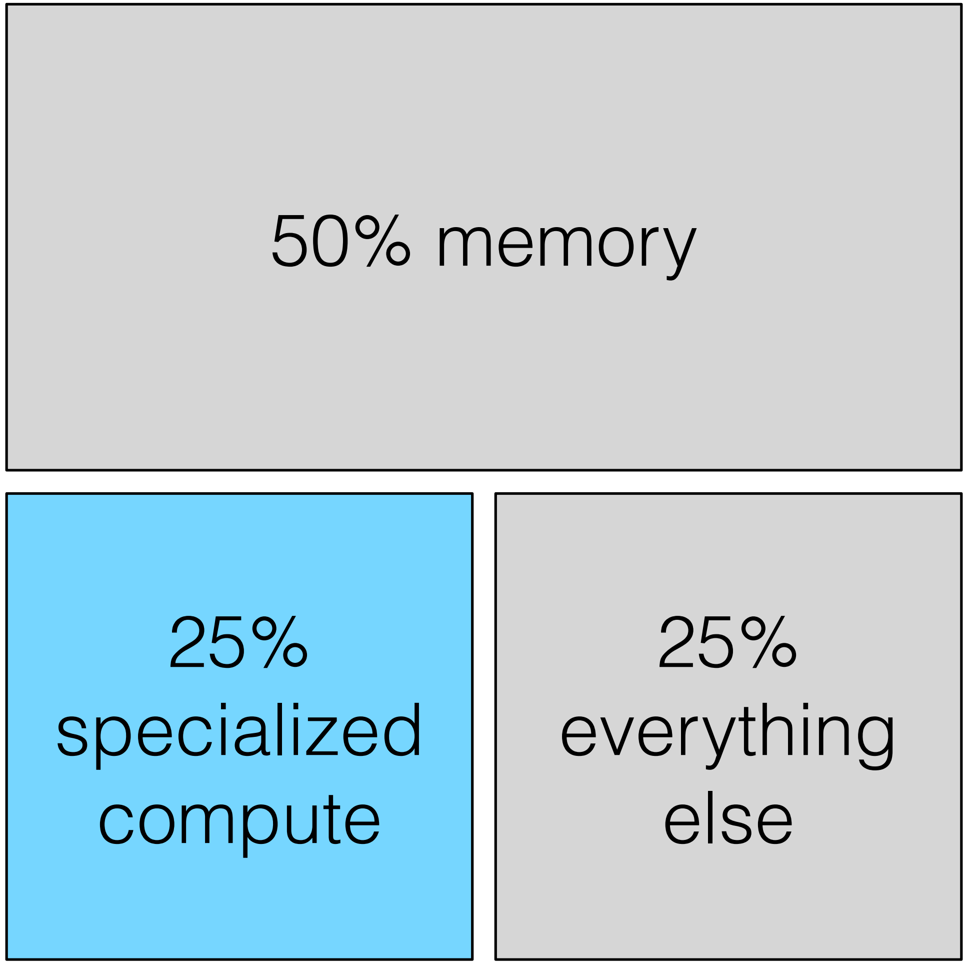
- Power is the problem, only part of the device can be on at a time, moving data takes the most power and compute is limited by data movement

- The implication of this thought chain with respect to how to design optimal hardware
    - Minimize off device data movement
      How: include sufficient on device memory
    - Minimize on device data movement
      How: data locality and accelerator reuse of data
    - Optimize compute
      How: exactly matched to algorithm, parallel to and sized for data movement

Big Compute SoC Trends

- Big compute device trends ≈
  - 50% memory
    - If off device data movement suddenly became very low power and very high throughput then this number will reduce
  - 25% optimized compute
  - 25% everything else

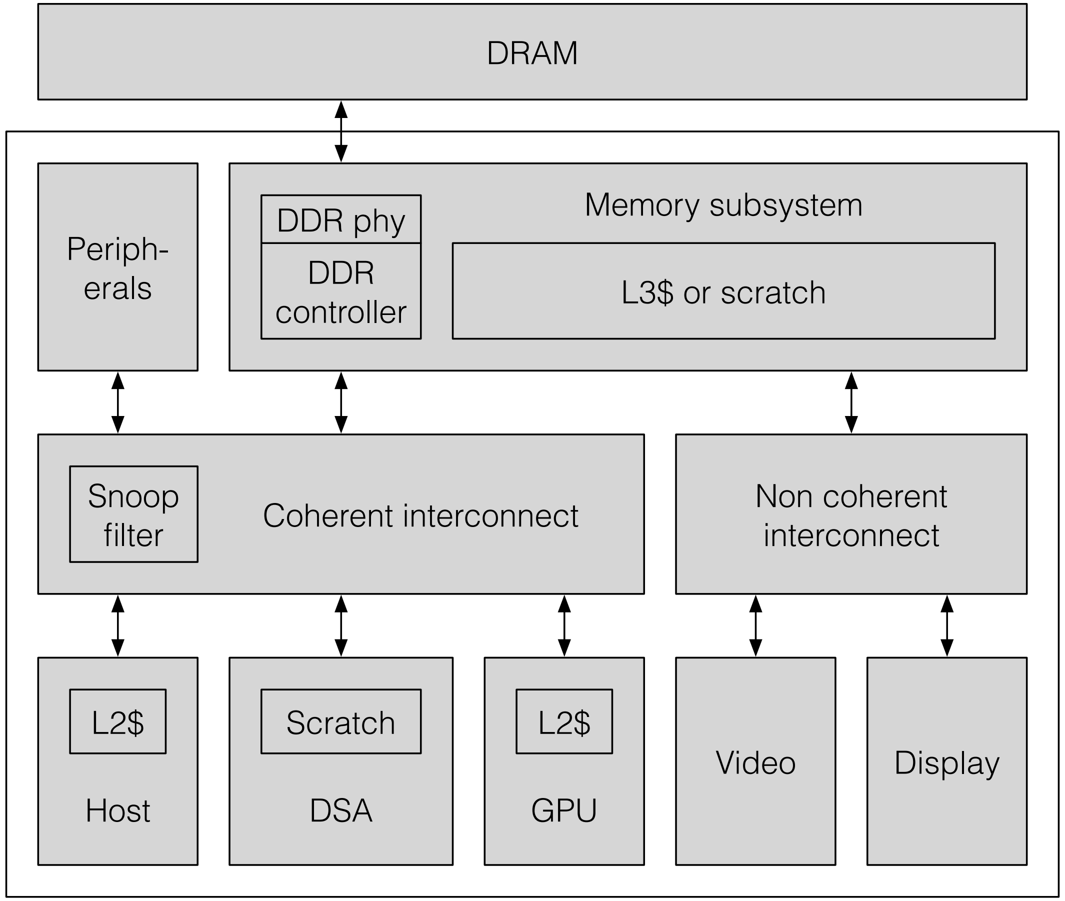
- Want the network designer to design the easiest networks to run as possible
  - But at the end of the day need to run the network
  - So how to support generality to future proof while still providing optimized compute



A Generic SoC Fabric Split Based On Memory

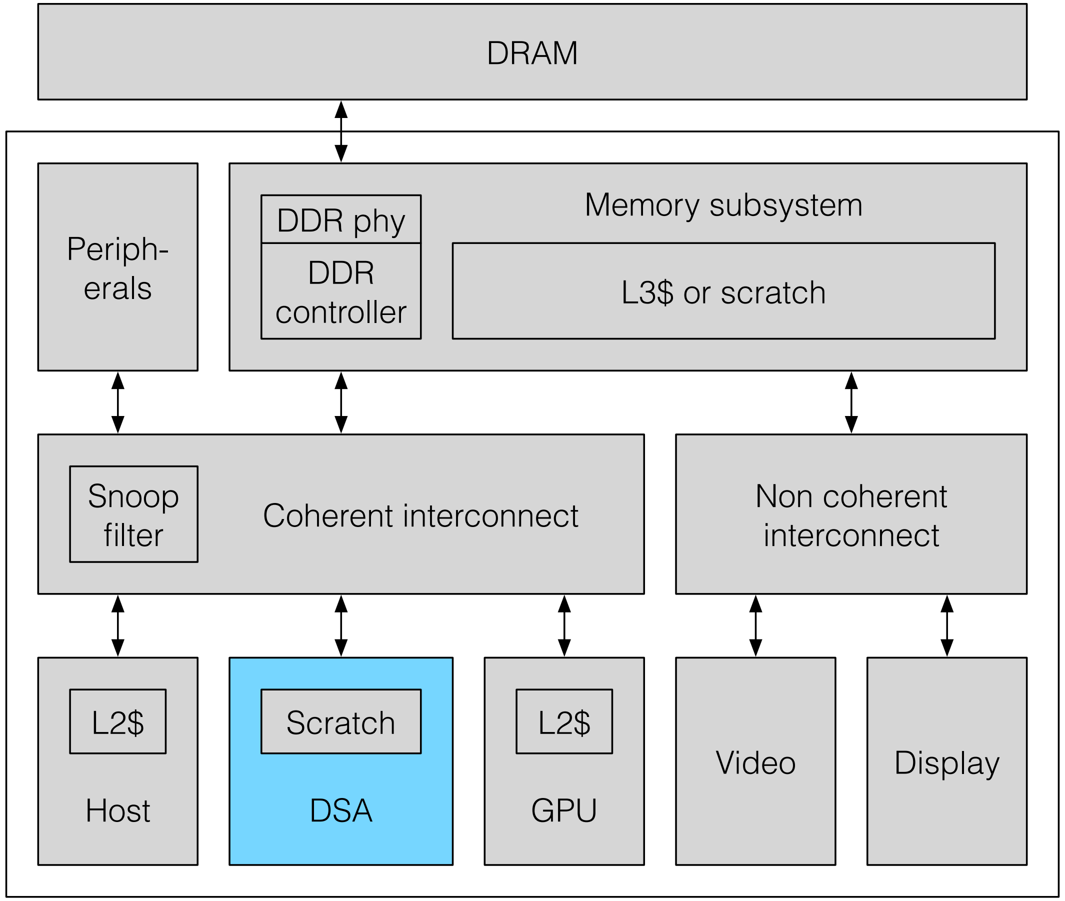
- Coherent
    - Host L2$
    - GPU L2$
    - L3$
- IO coherent
    - Peripherals
    - DSA
- Non coherent
    - Video
    - Display
    - L3 scratch

Note that there are other reasonable ways to split up a SoC fabric



Domain Specific Architecture

- Domain specific architecture
  - Hardware optimized for a specific application
  - Includes memory, control, comm and compute
  - Can potentially give the same benefits as multiple generations of Moore's Law
- Can include on the coherent or non coherent interconnect depending on the application
- Question: What is the optimal DSA for xNNs?
  - What type of memory is needed?
  - What type of control is needed?
  - What type of communication is needed?
  - What type of computation is needed?



Domain Specific Vs CPU Architecture

- A traditional CPU also has memory, control, computation and communication → But there are significant differences vs a DSA

- CPU
  - Latency optimized
  - Intelligence in hardware at run time
  - Generic communication, computation and memory

- DSA
  - Throughput optimized (for this domain)
  - Intelligence in software at compile time
  - Domain optimized communication, computation and memory

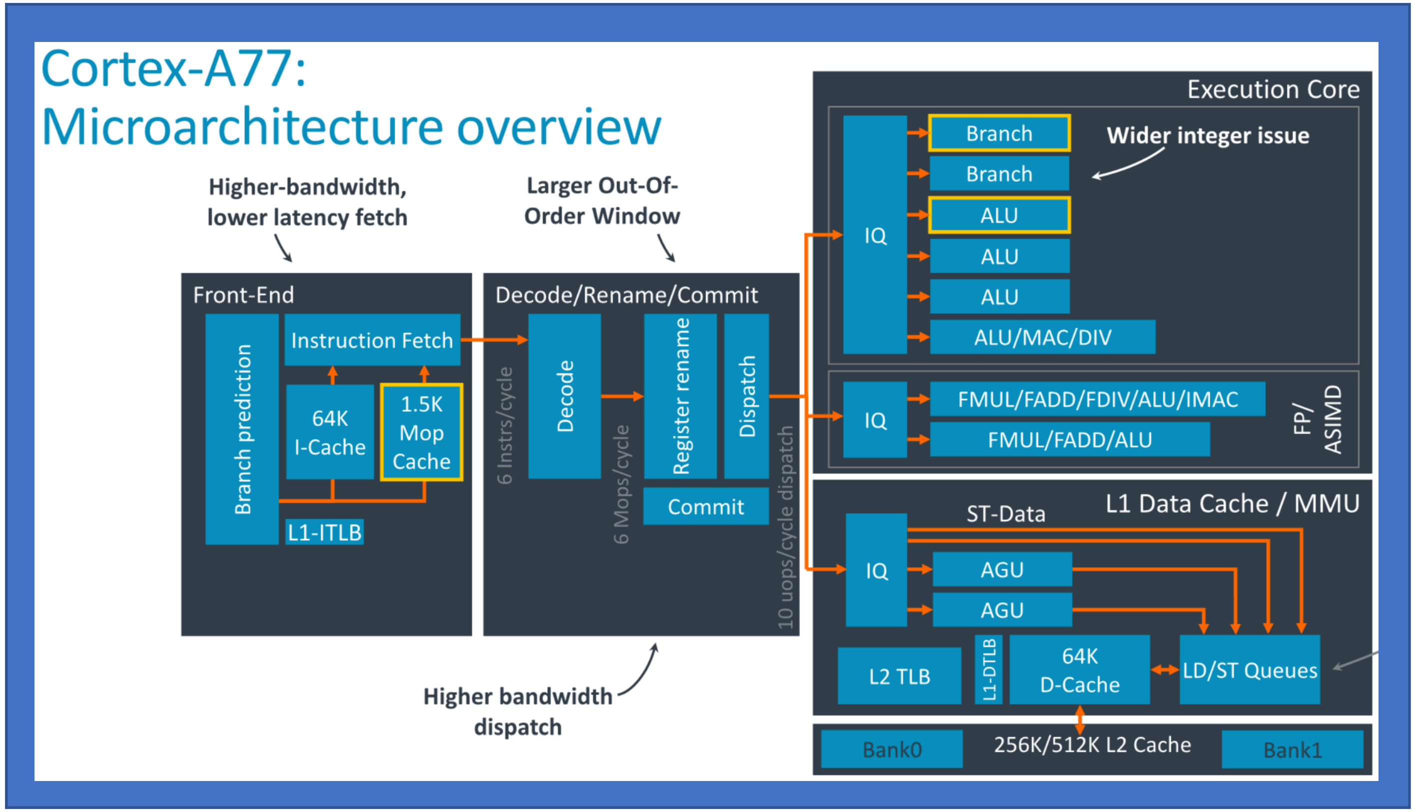


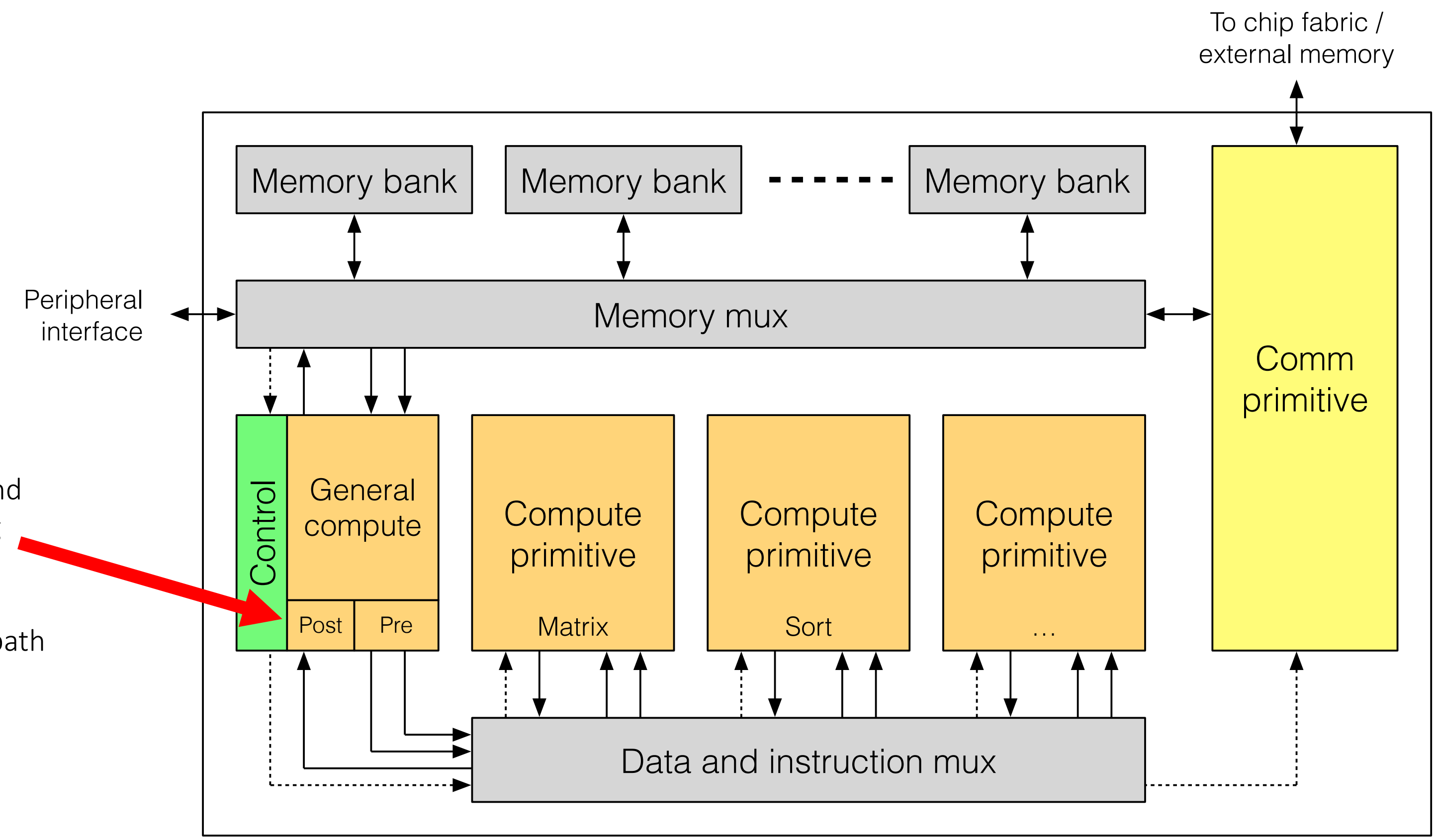
Figure from https://www.anandtech.com/print/14384/arm-announces-cortexa77-cpu-ip

Primitive Defined Domains

This is the better way to build a DSA

- Goal: keep domain specific optimality while allowing a high amount of generality
- Strategy: define the domain in terms of fundamental math, not an application
- Components
  - Memory
  - Control
  - Communication
  - Computation
- Control, communication and computation operate in parallel

With general compute pre and post processing part of the computational primitive data path



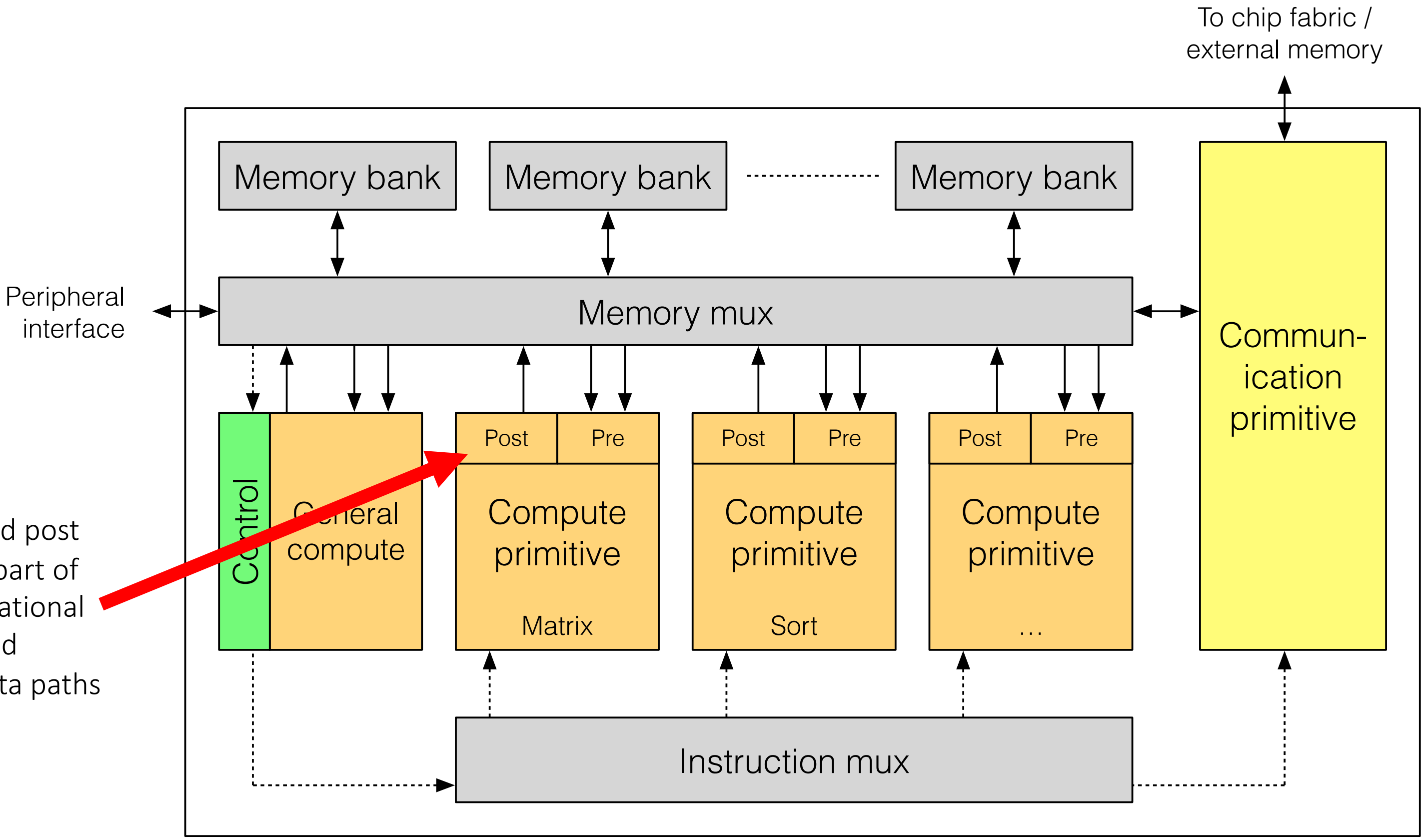
Note that there are many different configuration options

Primitive Defined Domains

This is the better way to build a DSA

- Goal: keep domain specific optimality while allowing a high amount of generality
- Strategy: define the domain in terms of fundamental math, not an application
- Components
  - Memory
  - Control
  - Communication
  - Computation
- Control, communication and computation operate in parallel

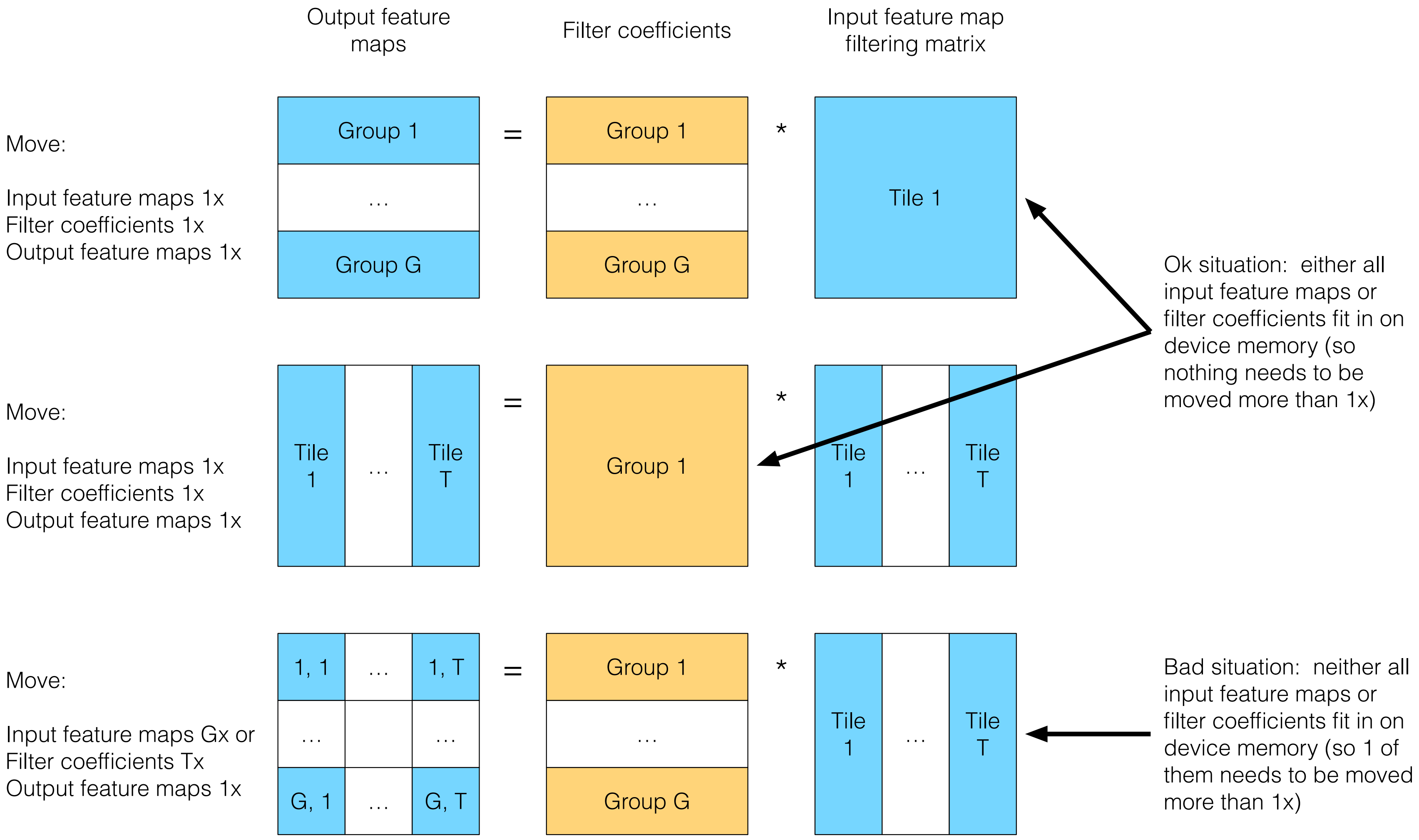
With pre and post processing part of the computational primitive and separate data paths



Note that there are many different configuration options

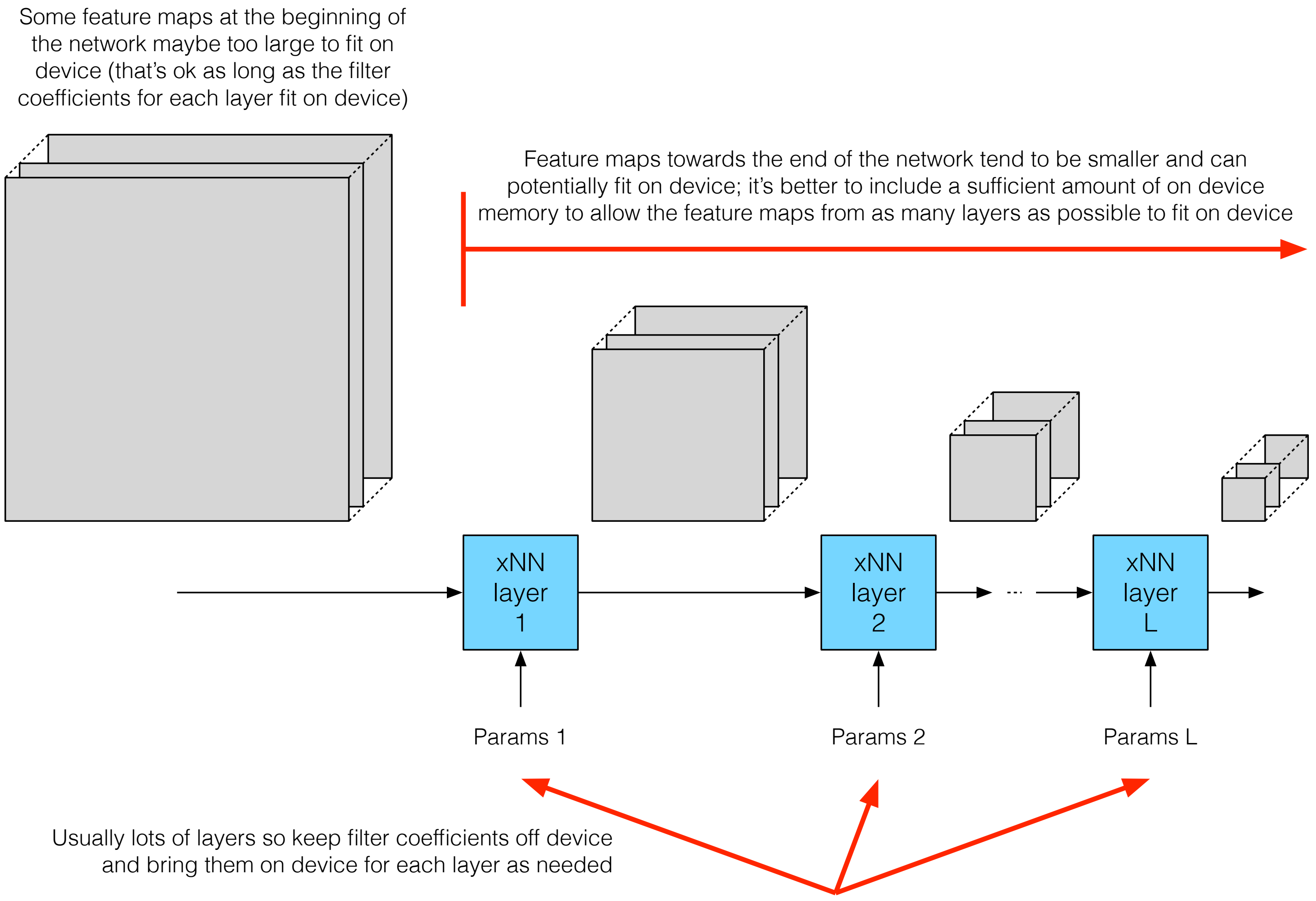
On Device Memory – How Much Is Ok?

Answer: sufficient on device memory such that for each layer (considered individually) either all feature maps or all filter coefficients fit on device



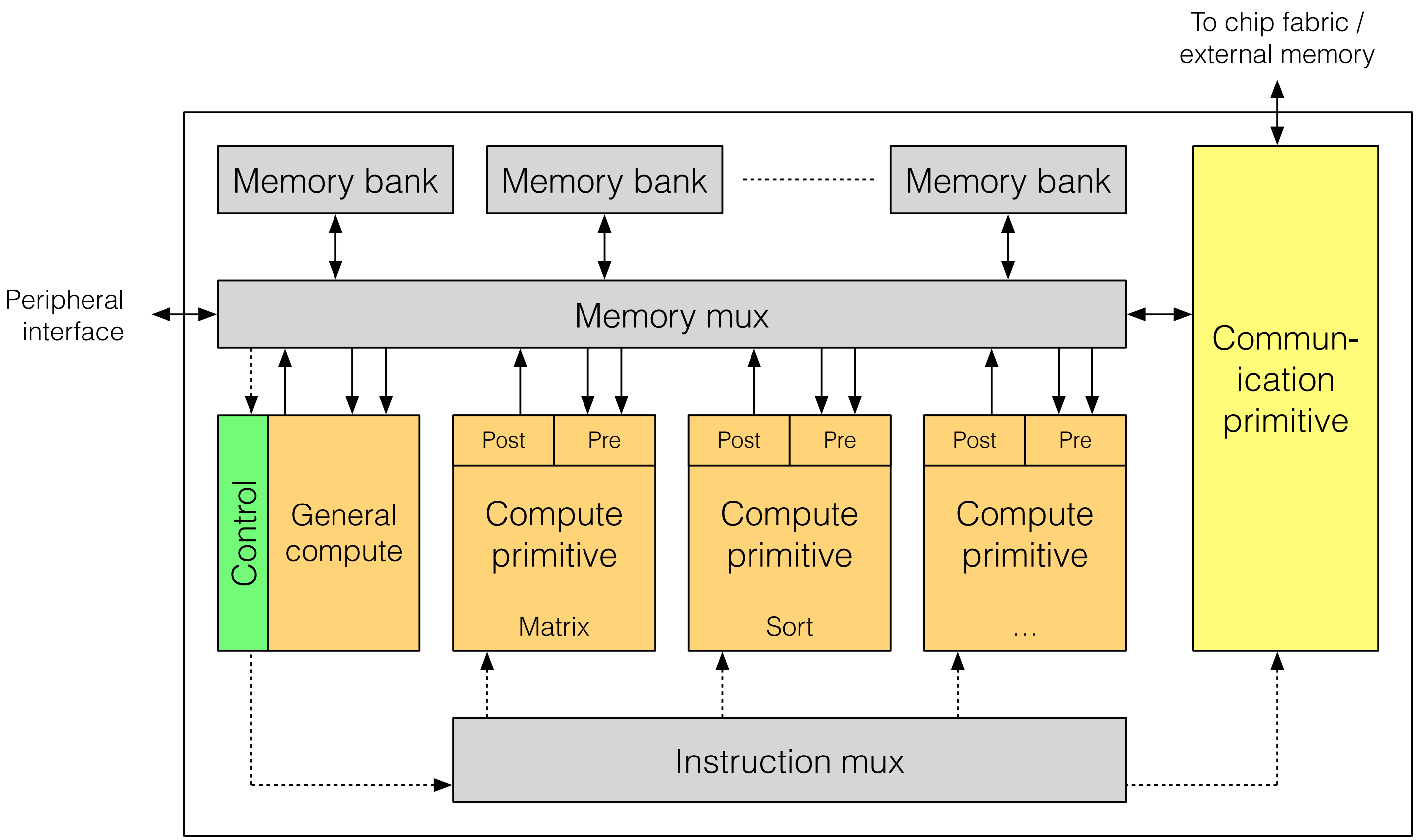
On Device Memory – How Much Is Better?

Answer: enough to keep feature maps fully on device such that on a per layer basis only weights need to be moved on device



Control

- Control cycles through nodes on the low level graph
- Passes instructions to the appropriate compute or communication primitive to execute supported nodes
- Typically part of a general compute resource that executes all non primitive supported nodes

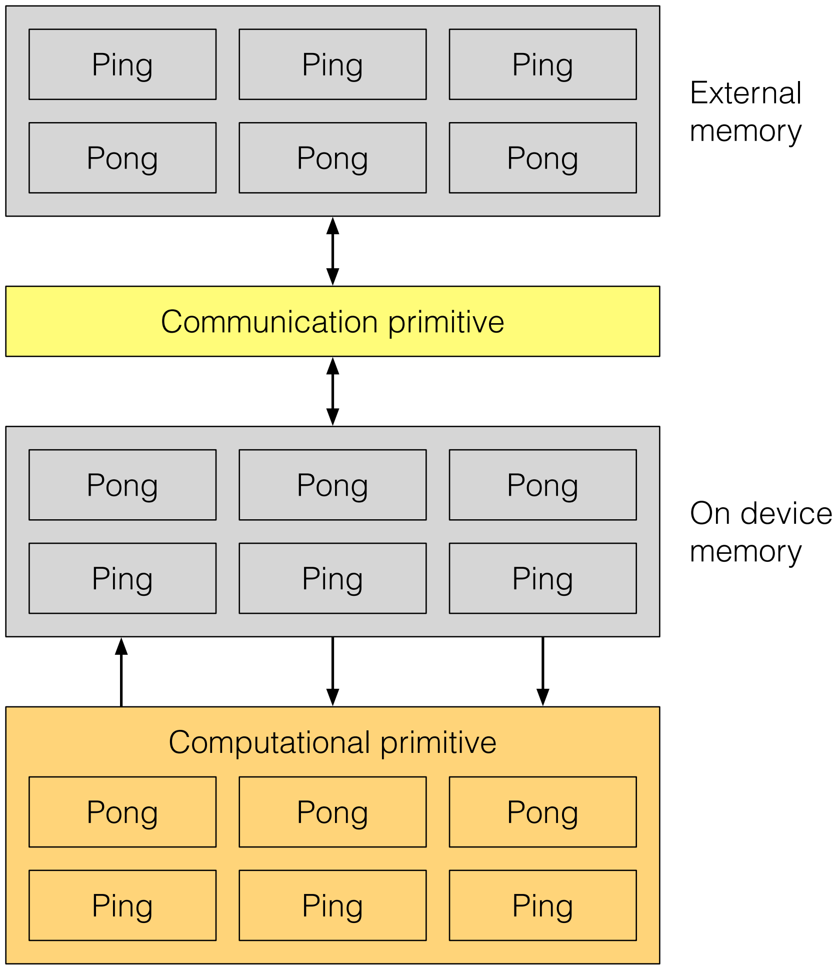


Communication Strategy

- Create ping pong buffers in DRAM and local memory if needed to allow continual compute in parallel with communication

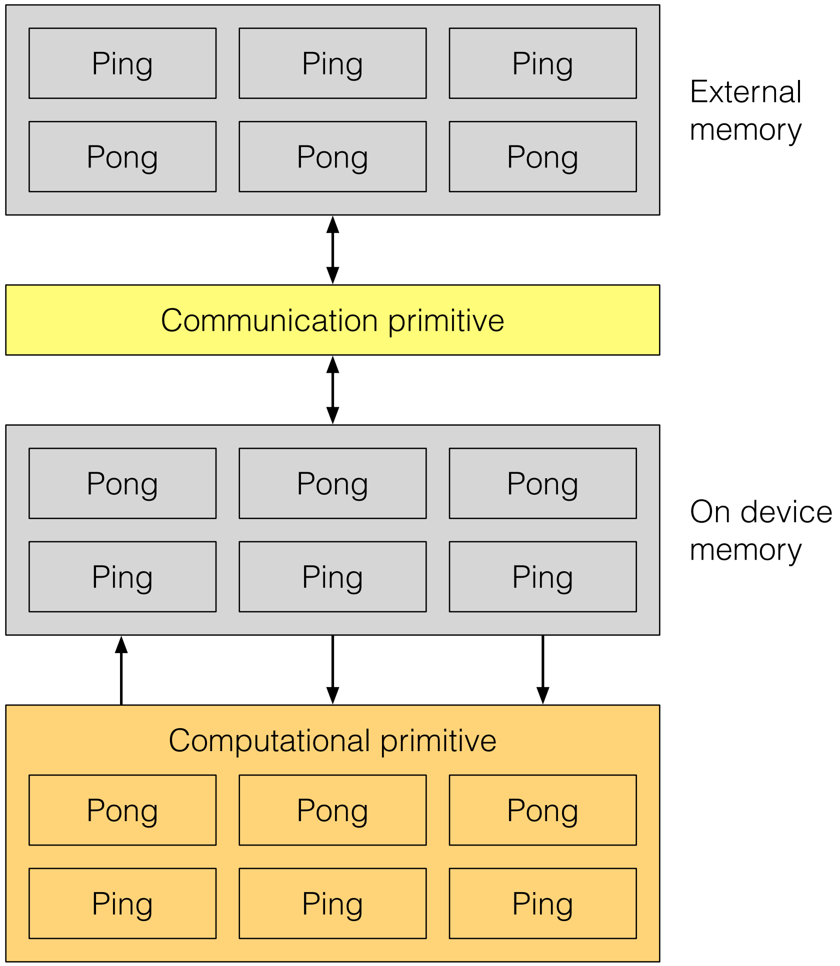
- Attempt to keep feature maps on device and bring in filter coefficients as needed per layer
  - As mentioned on the memory slide
  - This removes the need for feature maps to be involved in the ping pong scheme
  - Great if all filter coefficients fit on device too, but this is usually not the case

- Do the following in parallel
  - External to local memory data movement
  - Local memory to compute data movement
  - Compute



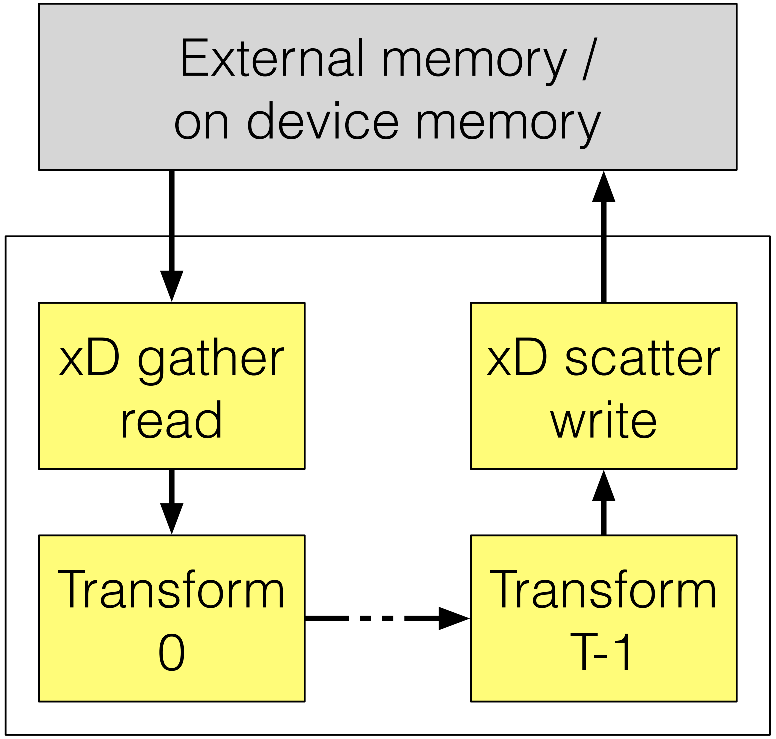
Communication Strategy

- Typically, external memory bandwidth is much less than internal memory bandwidth
- This implies 2 options for efficiency
  - Need to keep some fraction of data on device
  - Need to have a high level of data reuse once on device



Communication Primitive

- Really a transform primitive within a communication framework
- Data flow
  - xD gather: vector read from on device / DRAM mem
  - Ex transform: compress / decompress
  - Ex transform: encrypt / decrypt
  - xD scatter: vector write to DRAM / on device mem
- Note: many other transform primitives are possible
  - Structuring the communication framework this way allows new transform primitives to be added in a convenient way



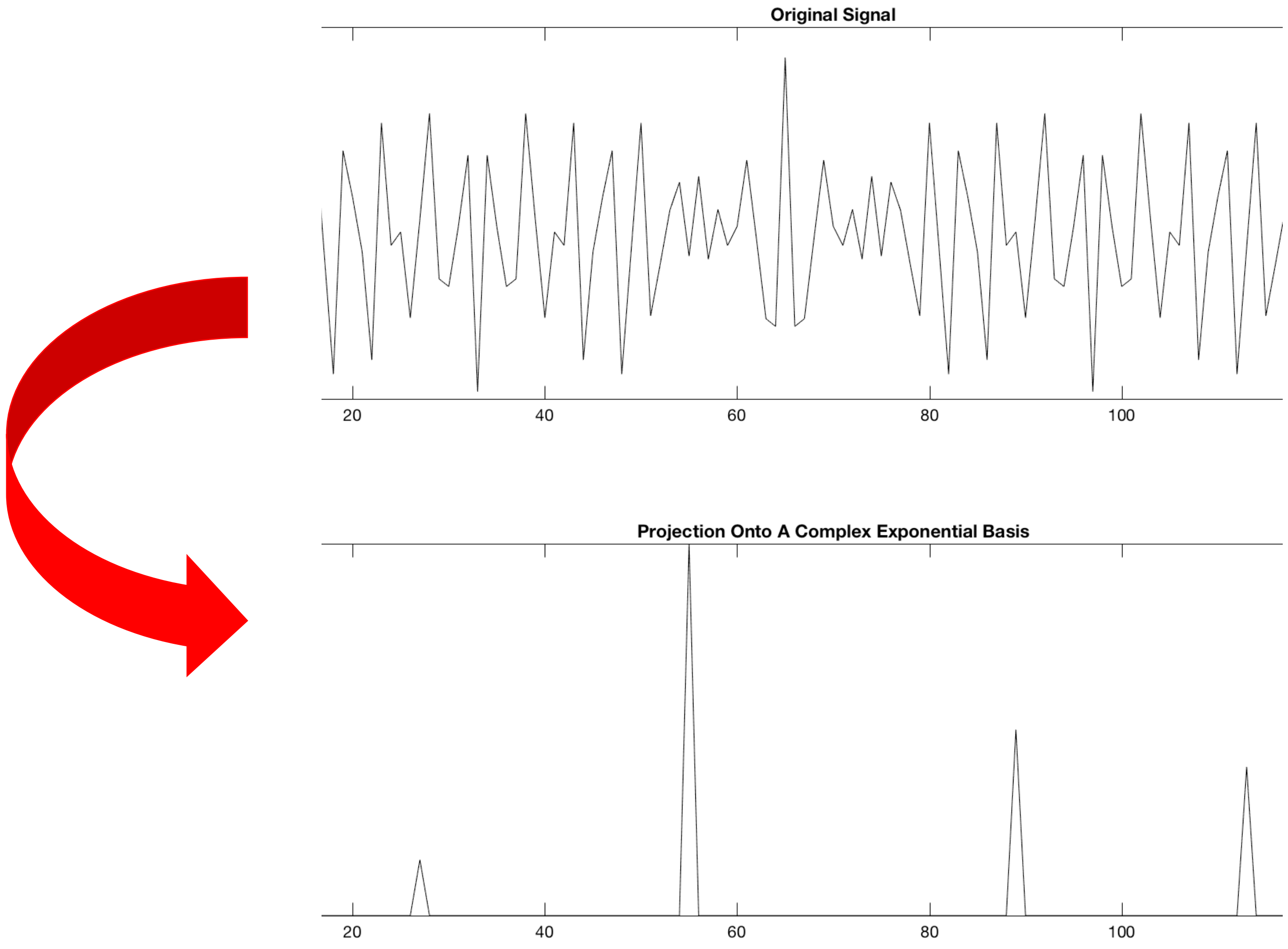
Computation Strategy

- Don’t put high level algorithms in gates
- Unless they’re part of the fixed portion of a standard
  - The transmitter part of a communication standard
  - The decoder part of a compression standard
  - ...
- Algorithm designers can change their minds like you change your socks (on a near daily basis)
- Hardware designers create silicon at a much much slower and more expensive pace
- The question: how do you get the efficiency of a dedicated accelerator but still enable generality with respect to high level algorithms?

An Analogy For Thinking About Computation

A small detour: a FFT projects a signal onto a complex exponential basis and tells you how much of each basis component was in the original signal

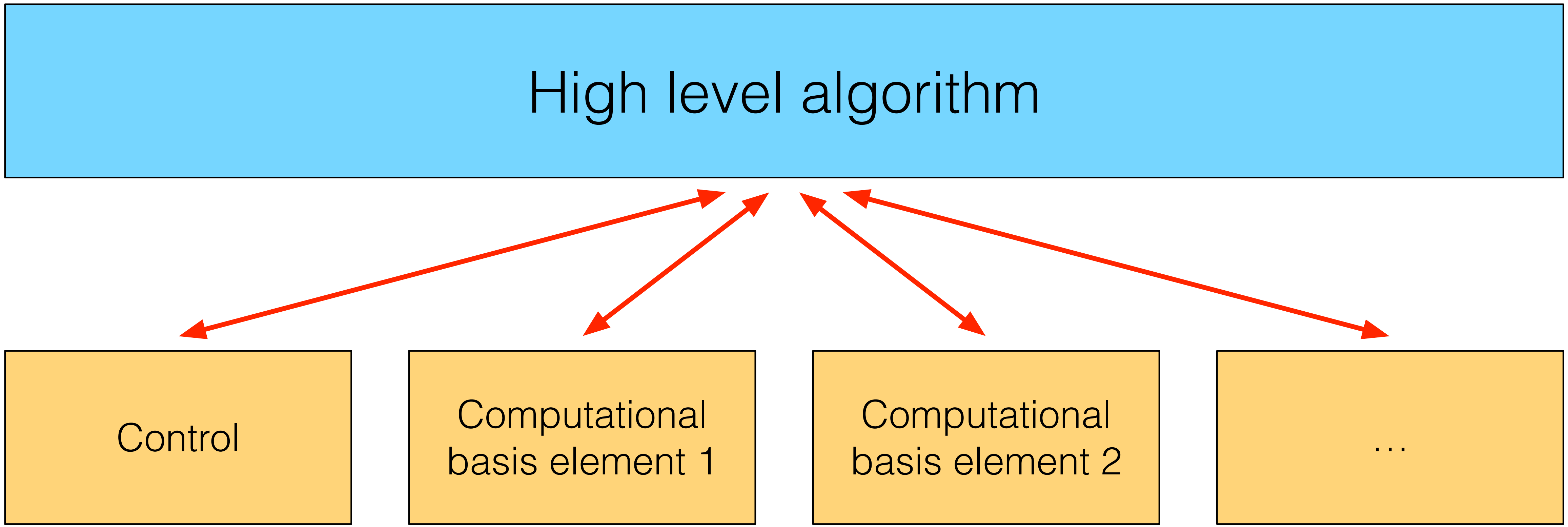
FFT



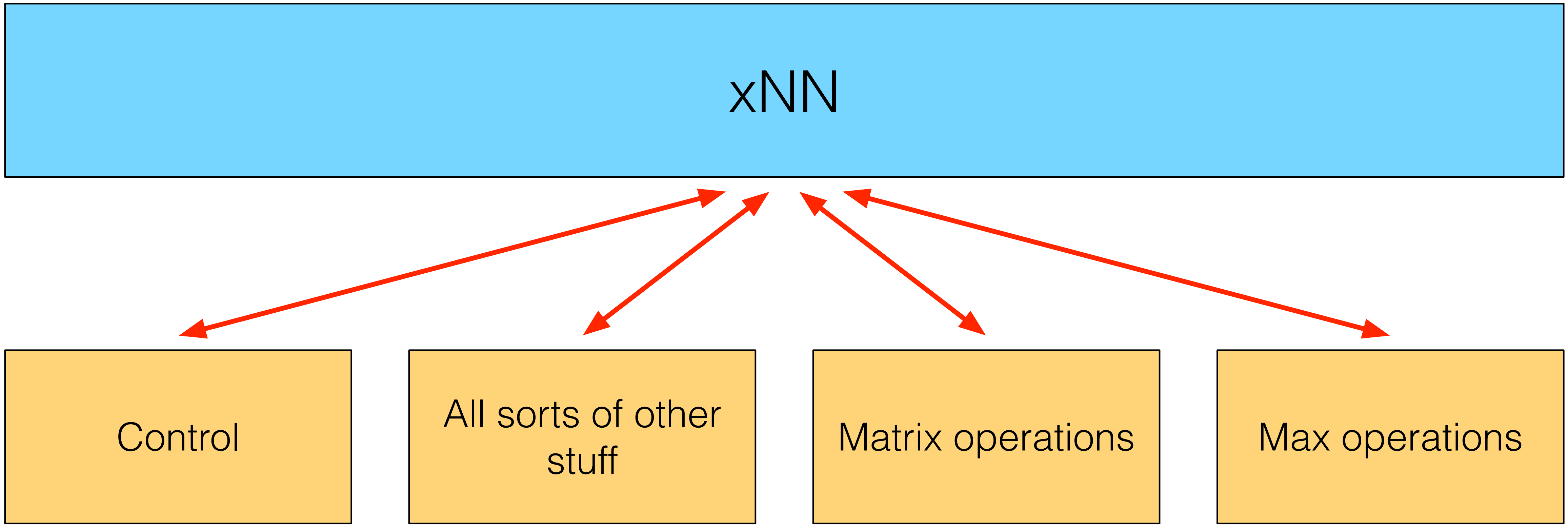
Dear observant person: yes the top figure is a bit of a cheat (just the abs of the sequence) because I was too lazy when making figures to construct a real even signal; but the point of the slide is valid

A Computational Basis For Algorithms

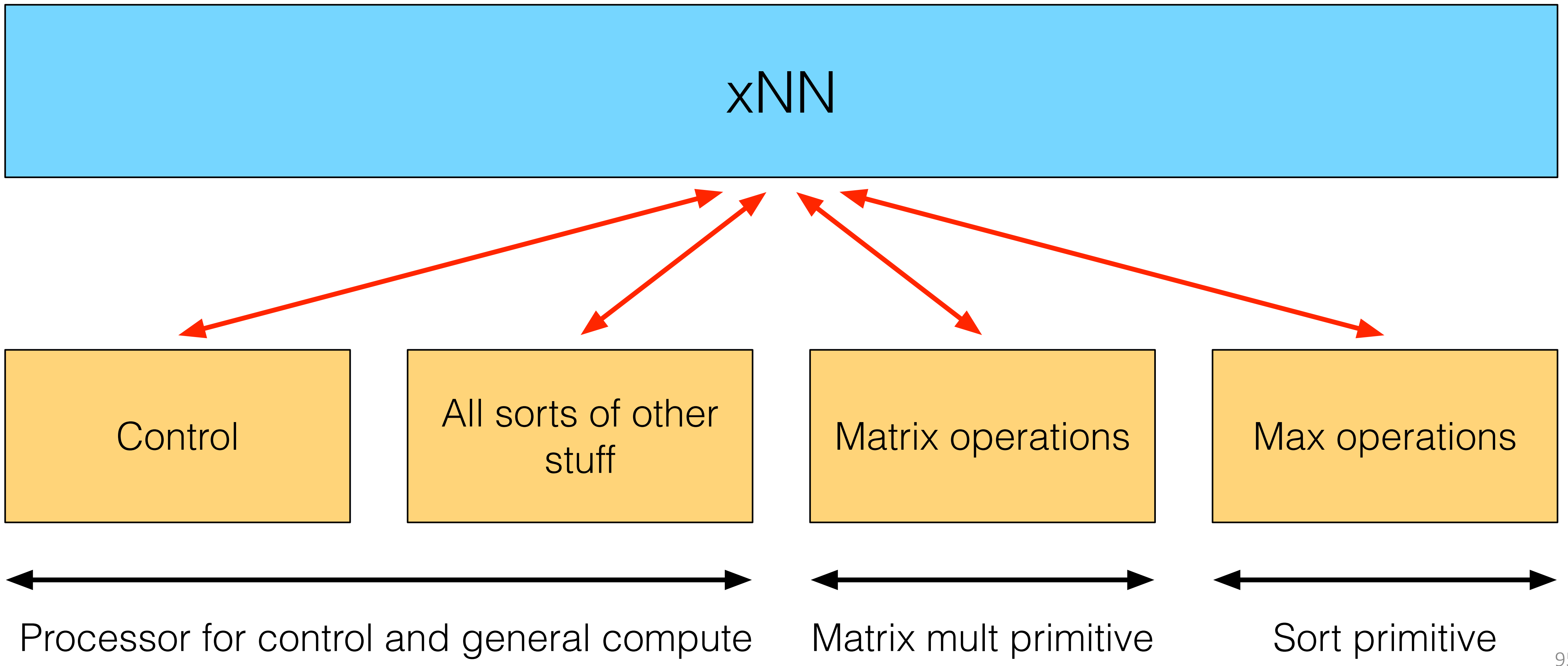
Use a similar strategy for designing hardware: decompose high level algorithms onto a computational basis and provide optimized implementations of key computational basis elements (get ASIC efficiency while maintaining algorithmic generality)



Projecting A xNN Onto A Computational Basis

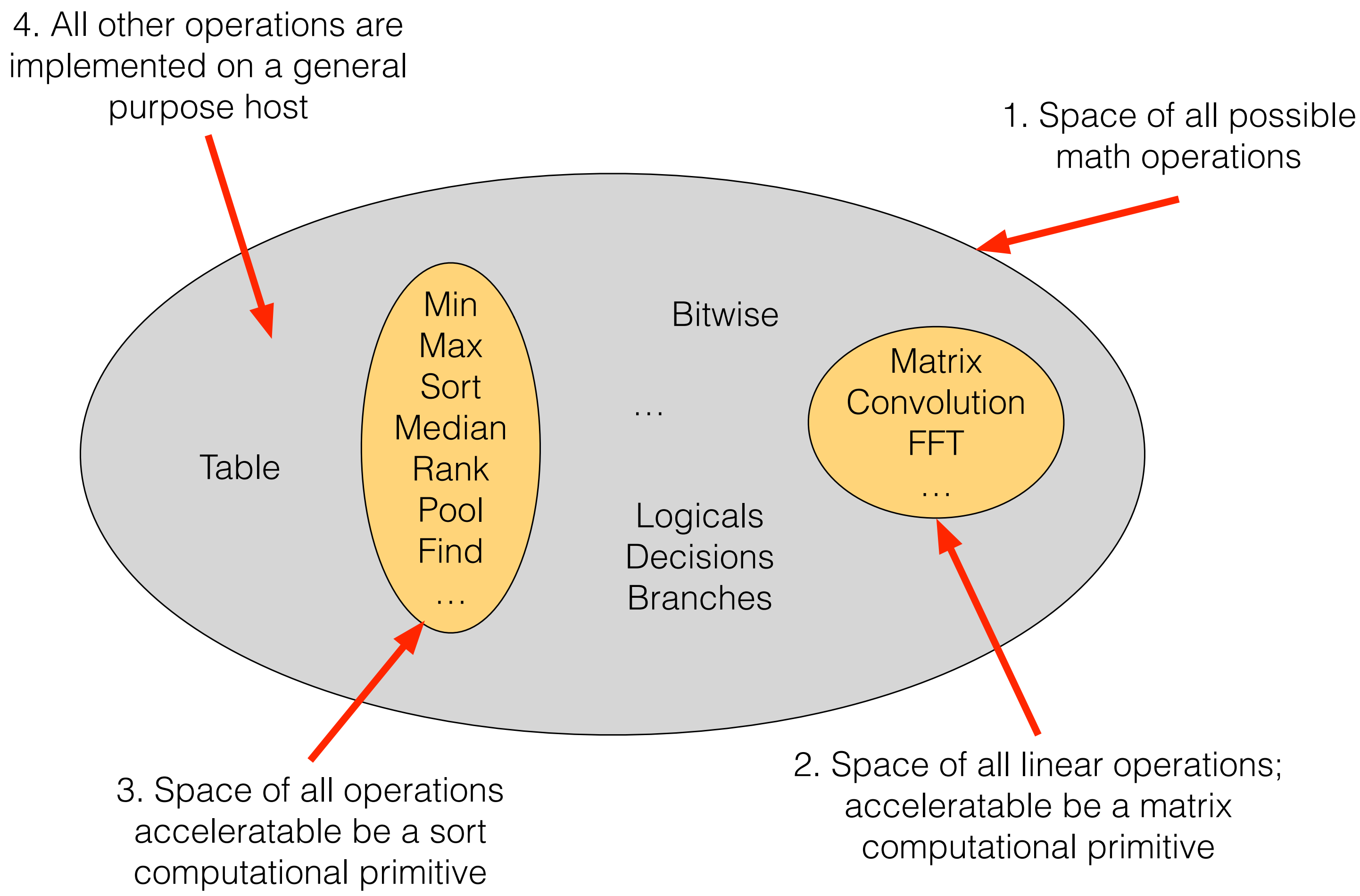


Projecting A xNN Onto A Computational Basis



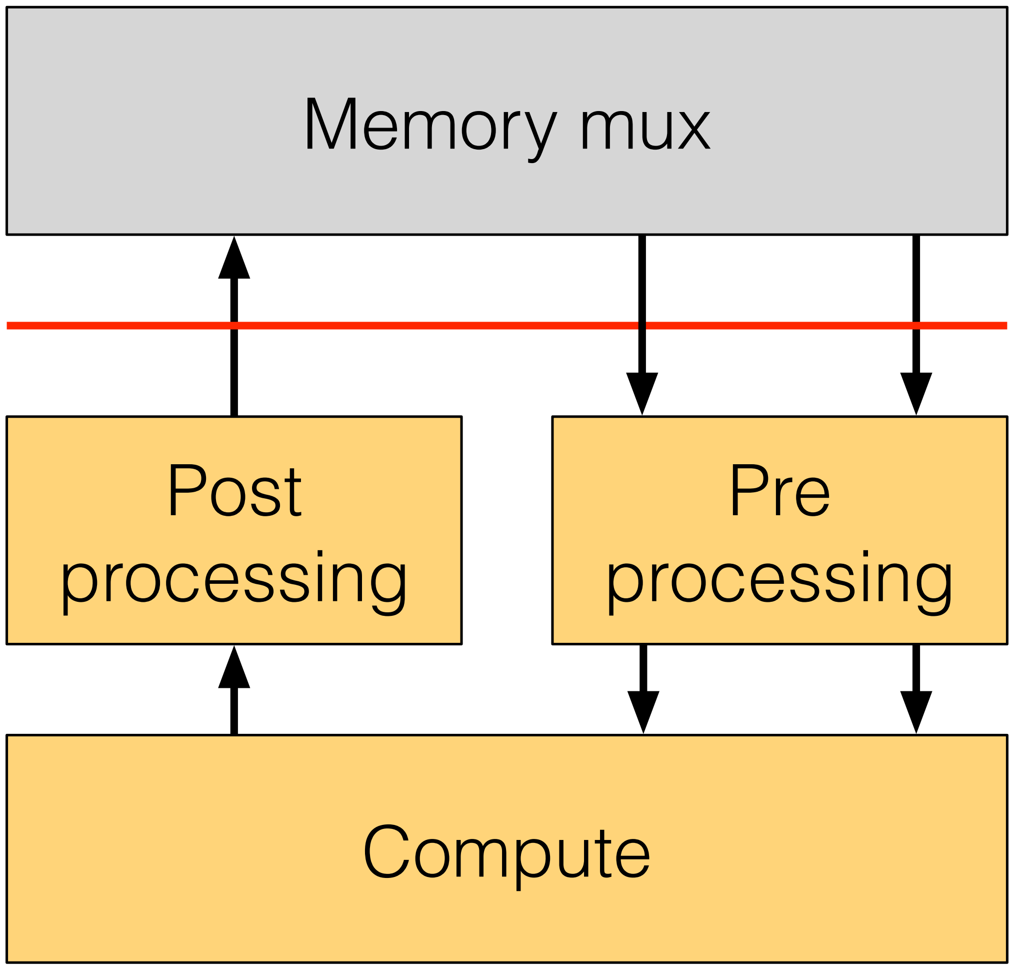
Computational Primitive Candidates

- Matrix multiplication is a given
- Sort seems appropriate
  - Or maybe a general divide and conquer primitive
- After that it's more open ...
  - Perhaps something for an ISP if vision focused
  - Perhaps some if machine for MCTS optimization if RL focused
  - ...



Computational Primitive Template

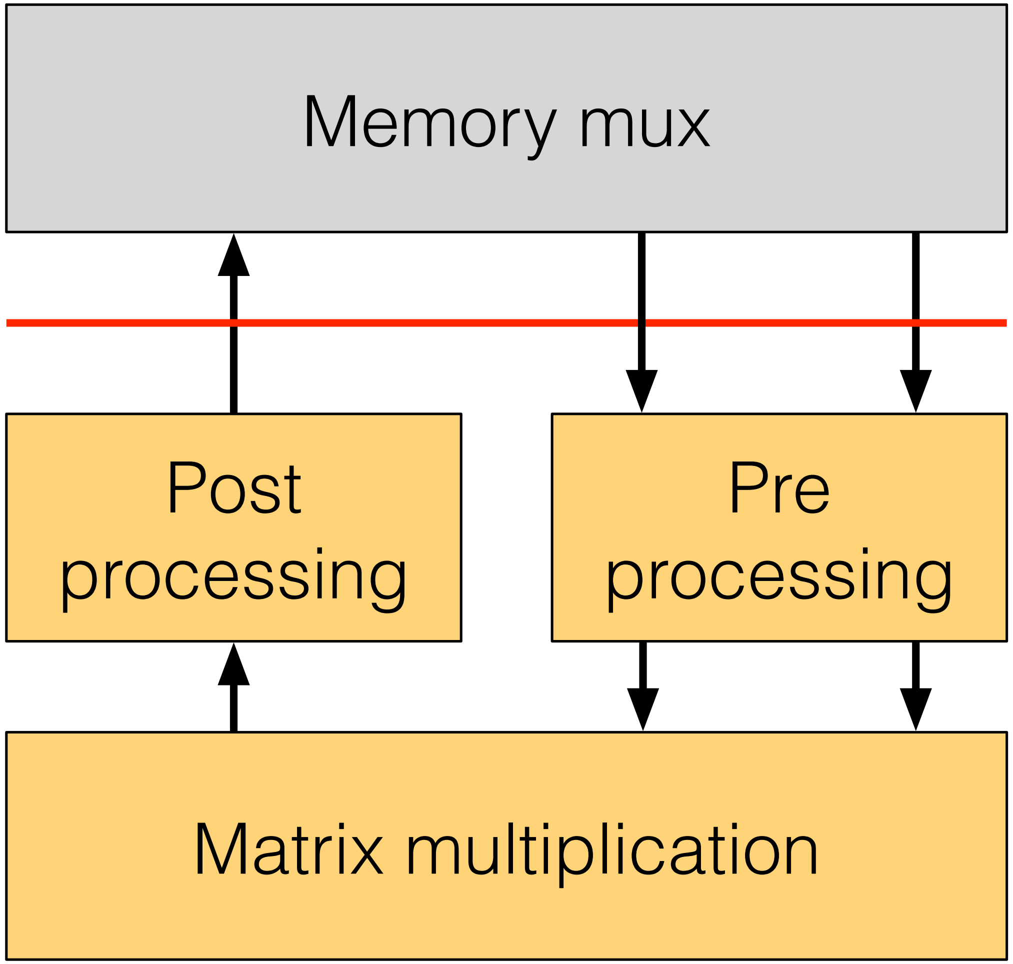
- Pre processing
  - Simple limited set of data transformations
  - Allows optimized regular compute structure
  - Enable generality within the primitive class
- Compute
  - Computational primitive computation
- Post processing
  - Simple limited set of data transformations
  - Allows optimized regular compute structure
  - Enable generality within the primitive class



Matrix Multiplication Primitive

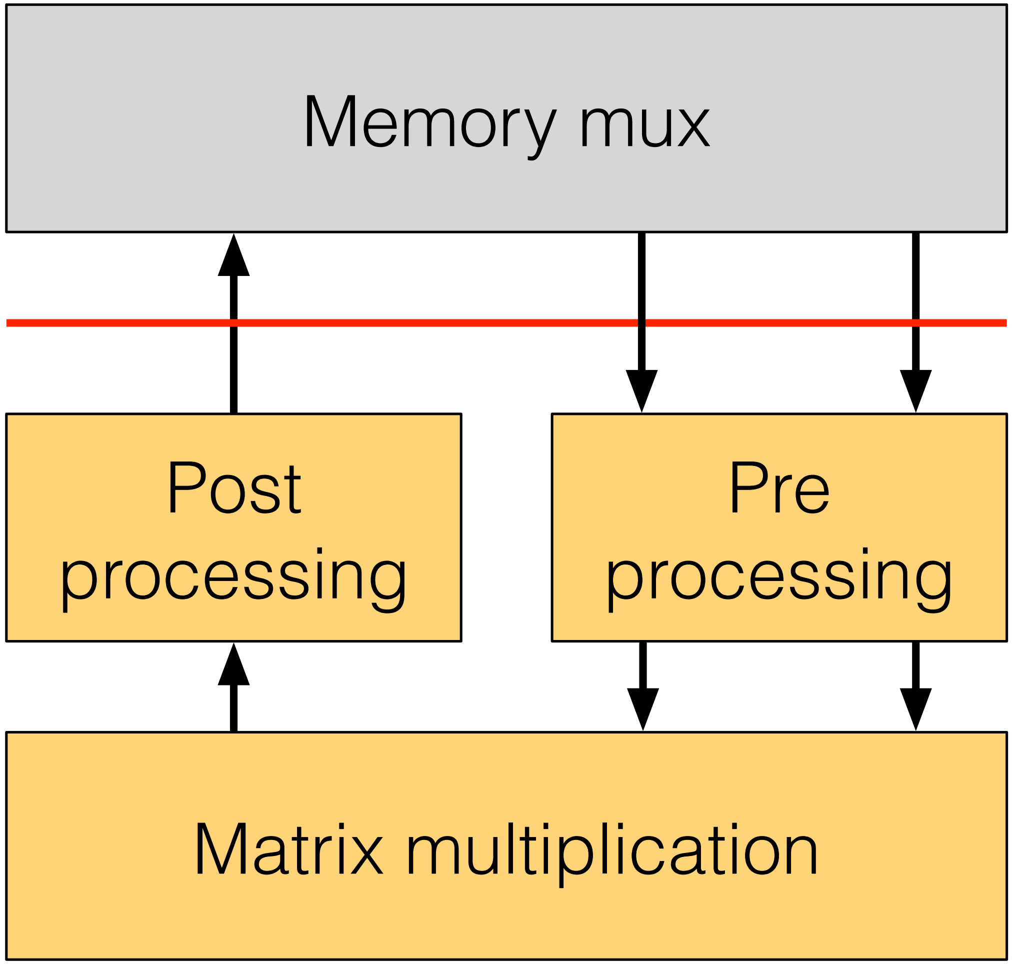
- Flow
  - Read two N x N matrices in N cycles
    - Ex: $H_{tile}(m, k)$ and $X_{tile}(k, n)$
  - Use pre processing to do formatting for the input
    - Ex: $H_{tile}(m, k) \rightarrow A$ and $X_{tile}(k, n) \rightarrow X_{filter}(k, n) \rightarrow B$
  - Compute N x N matrix multiplication in N cycles
    - C += A*B
  - Use post processing to do formatting for the output
    - $C \rightarrow Y_{tile}(m, n)$
  - Write a N x N matrix in N cycles

- Compute to data movement ratio
  - N^3 MACs in N cycles or N^2 MACs / cycle
  - A maximum of 3N pieces of data read or written to local memory per cycle



Matrix Multiplication Primitive

- Comments
  - Reads and writes are address aligned to maximize bandwidth efficiency
  - Pre and post processing formats data to transform a compatible problem into matrix multiplication and adapt the problem size (e.g., using block matrix multiplication)
  - Compute uses ping pong registers to hold matrices as necessary based on the selected matrix multiplication method to allow continual compute in parallel with data movement
  - Different precisions are supported via scaling the matrix size keeping bandwidth constant
  - For fixed point compute
    - Accumulation is typically at 4x the number of bits of the input operands, scale round clip to bring the output back to the number of bits of the input
    - Supporting 8, 16 and 32 bit precision can be accomplished with the same bandwidth, memory and compute via appropriate multiplier design and using primitive sizes of 1x, 1/2x and 1/4x, respectively

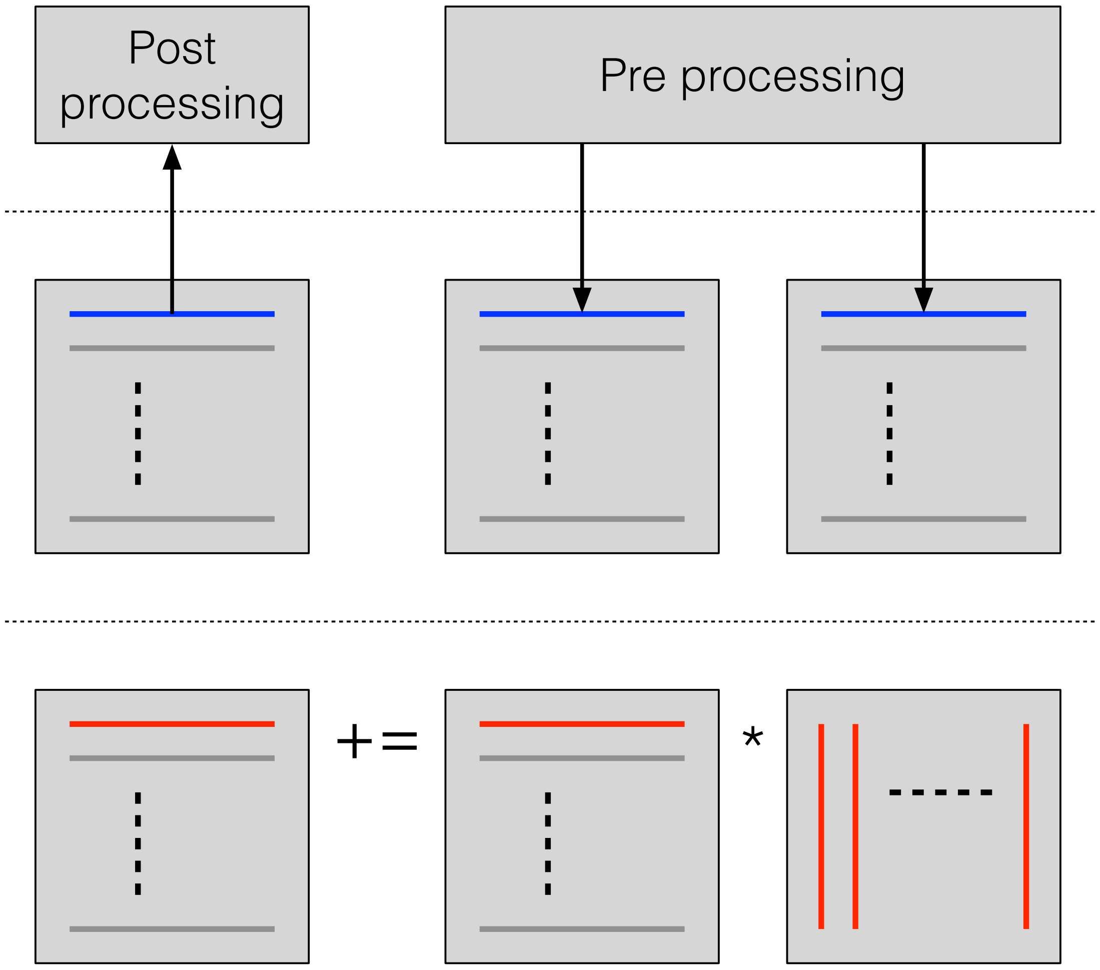


Inner Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is in linear order
  - B is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via background load
- Per cycle transfer data
  - Read A(m, :) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute 1 output row C(m, :)

```
C = C0                   // e.g., bias matrix
For m = 0 to M - 1       // m = 0
  For n = 0 to N - 1
    For k = 0 to K - 1
      C(m, n) += A(m, k) B(k, n)
    End
  End
End
```

Parallel

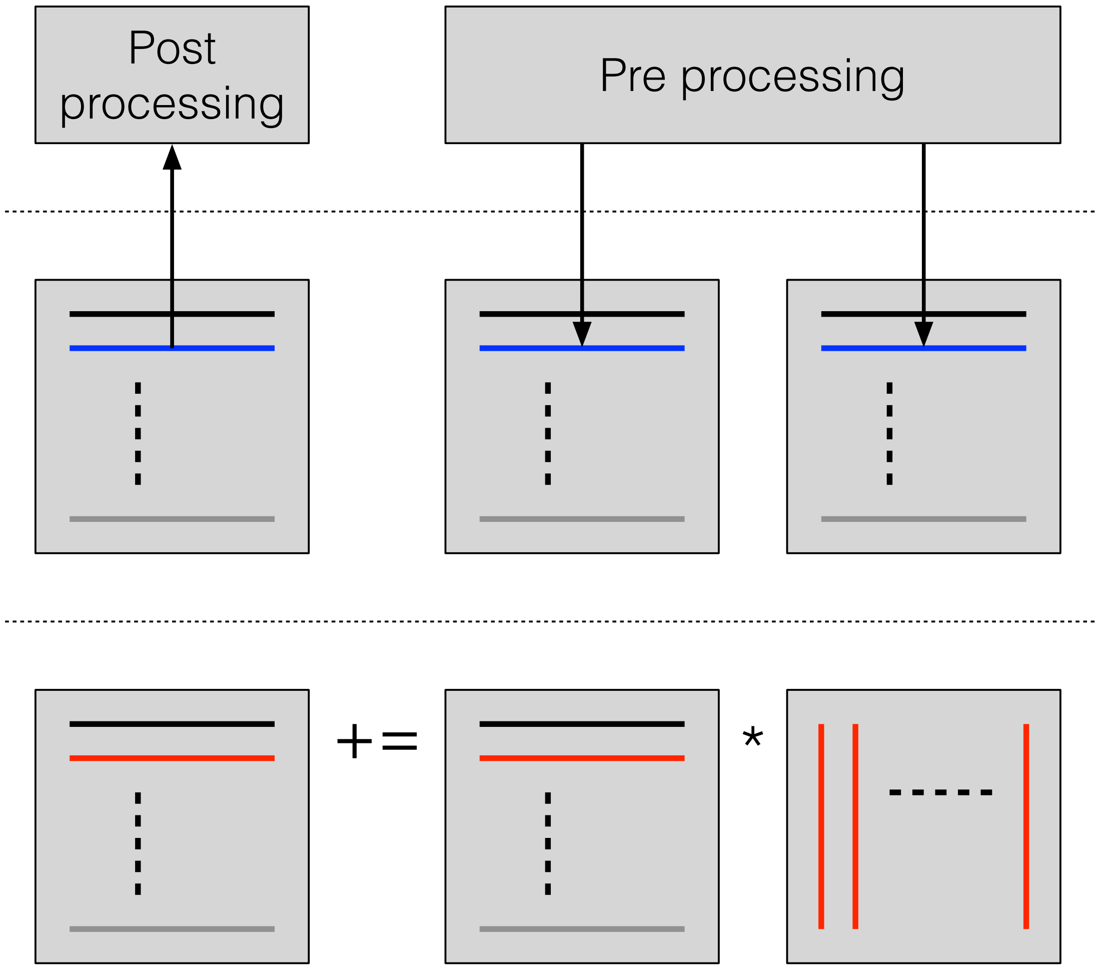


Inner Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is in linear order
  - B is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via background load
- Per cycle transfer data
  - Read A(m, :) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute 1 output row C(m, :)

```
C = C0                  // e.g., bias matrix
For m = 0 to M − 1      // m = 1
  For n = 0 to N − 1
    For k = 0 to K − 1
      C(m, n) += A(m, k) B(k, n)
    End
  End
End
```

Parallel

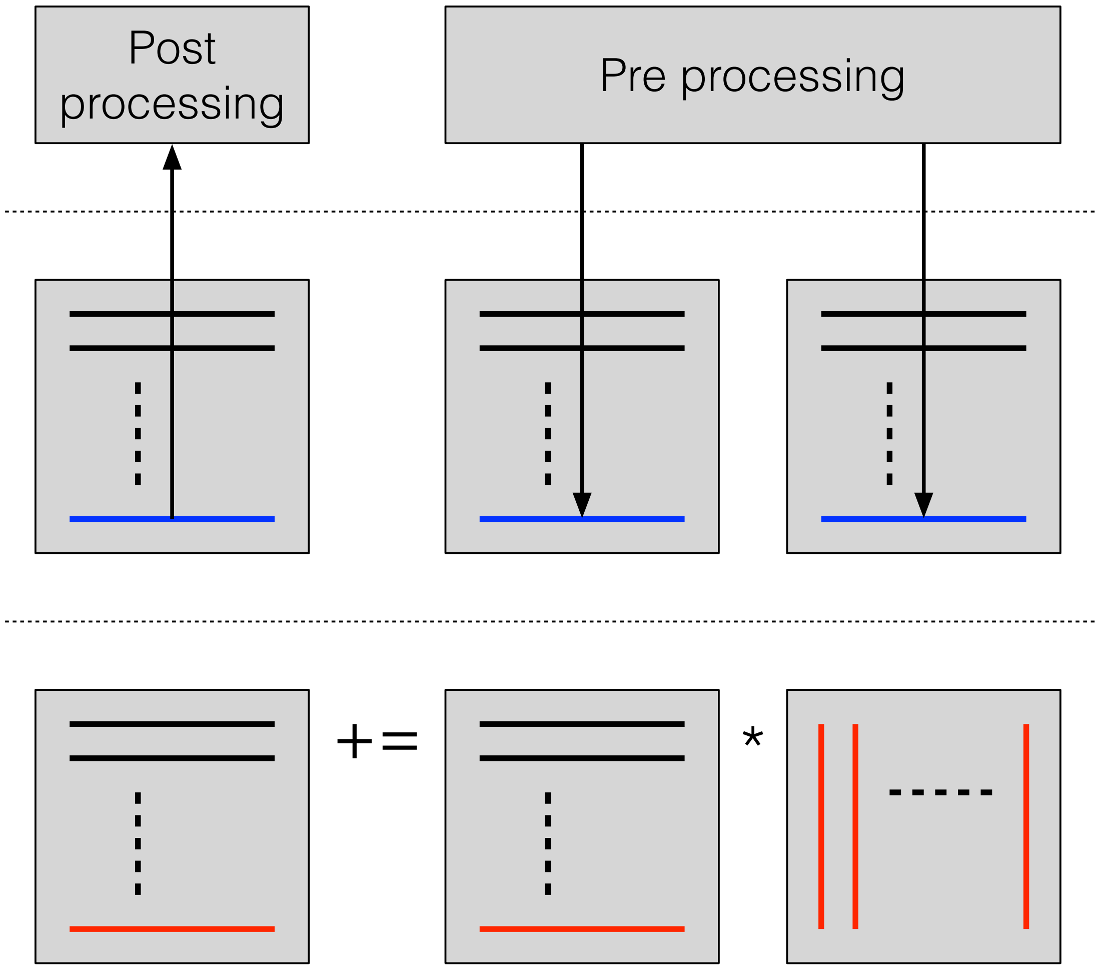


Inner Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is in linear order
  - B is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via background load
- Per cycle transfer data
  - Read A(m, :) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute 1 output row C(m, :)

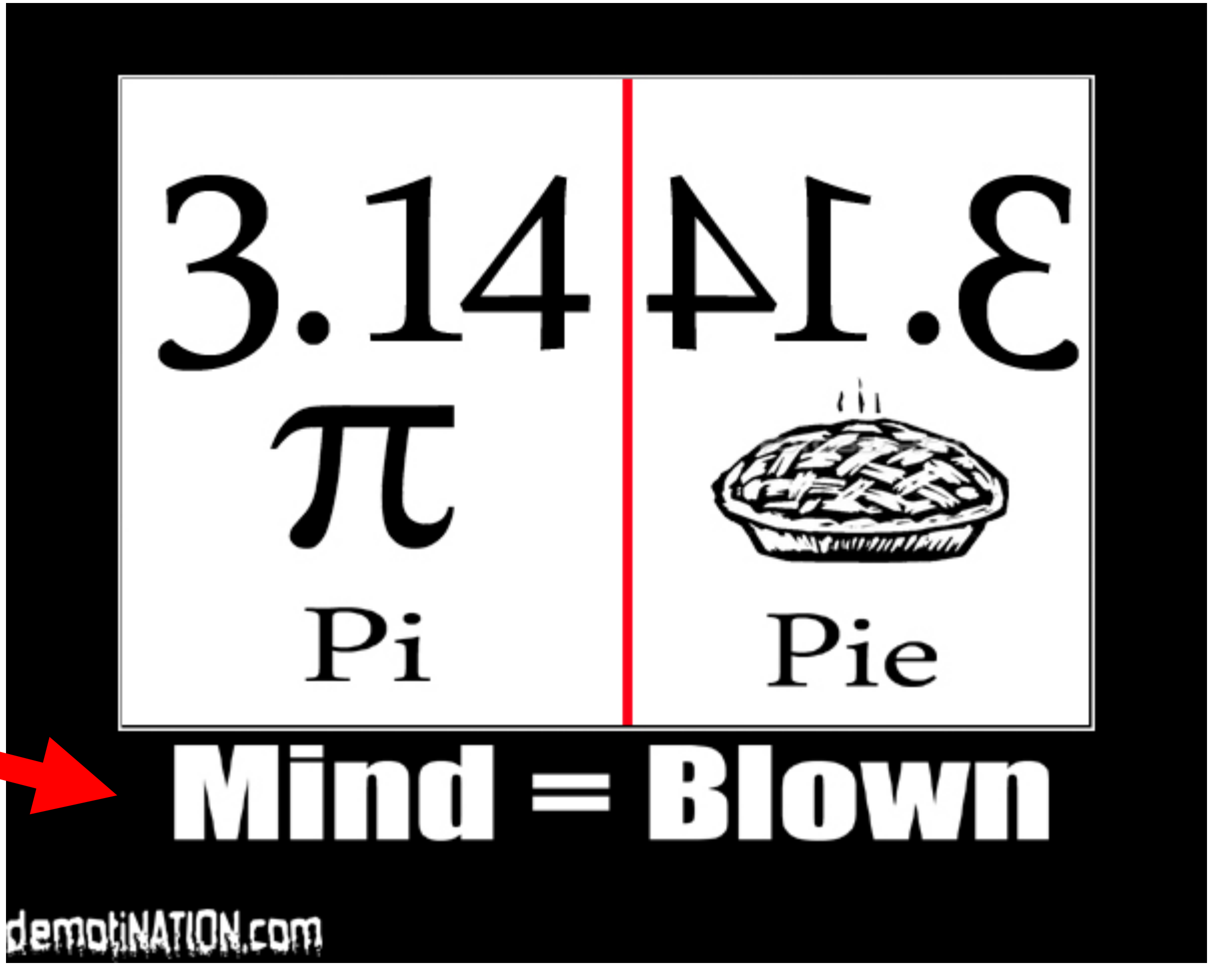
```
C = C0                  // e.g., bias matrix
For m = 0 to M − 1  // m = M − 1
    For n = 0 to N − 1
        For k = 0 to K − 1
            C(m, n) += A(m, k) B(k, n)
        End
    End
End
```

Parallel



Multiplying Matrices With < N^3 MACs

- The above described methods for matrix matrix multiplication require N^3 MACs for M = N = K square matrices
- N^3 is a large number, it would be nice to have a smaller exponent
- It's possible to multiply 2 matrices with less than N^3 MACs
- But there are tradeoff of additions, sequential operations, memory movement, ...
  - In practice this makes it somewhat questionable for many cases when applied to optimal architectures



How To Reduce The Cost Of A Calculation

- Need the operations that make up the calculation to have different costs so you can tradeoff 1 for the other
  - Example: multiplies cost more than adds
- Generally need to create intermediate terms that will be re used
  - This is used in many places, either implicitly or explicitly
- Sometimes the intermediate term strategy is recursively applied

Figure from https://www.giantbomb.com/articles/hey-its-a-premium-membership-sale/1100-5747/

Example Applications Of This By Gauss

- Example 1: computing FFTs via a power of 2 matrix decomposition
- Example 2: multiplying 2 complex numbers
  - Standard = 4 real multiplies and 2 real adds

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

  - Gauss trick = 3 real multiplies and 5 real adds
    - Partial terms (note sequential dependency)

$$u = ac,\ v = bd,\ x = a + b,\ y = c + d,\ z = xy$$

    - Final result

$$ac - bd = u - v$$
$$ad + bc = z - u - v$$

Strassen's Algorithm For Matrix Multiplication

- Strassen's algorithm for reducing the number of multiplications in matrix matrix multiplication
  - Multiplies two block 2 x 2 matrices using 7 block multiplies vs the standard of 8
  - Recursively apply
  - Reduces multiplications to ~ $O(N^{\log\_2(7)}) = O(N^{2.81})$

- Strassen mechanics for the multiplication of N x N matrices C = A B

$$\begin{bmatrix} C(0,0) & C(0,1) \\ C(1,0) & C(1,1) \end{bmatrix} = \begin{bmatrix} A(0,0) & A(0,1) \\ A(1,0) & A(1,1) \end{bmatrix} \begin{bmatrix} B(0,0) & B(0,1) \\ B(1,0) & B(1,1) \end{bmatrix}$$

Strassen's Algorithm For Matrix Multiplication

- Define the following 7 partial terms

New "A"s size N/2	New "B"s size N/2	
S_1 = (A(0,0) + A(1,1))	(B(0,0) + B(1,1))	// 1 mult, 2 add
S_2 = (A(1,0) + A(1,1))	(B(0,0))	// 1 mult, 1 add
S_3 = (A(0,0))	(B(0,1) – B(1,1))	// 1 mult, 1 add
S_4 = (A(1,1))	(B(1,0) – B(0,0))	// 1 mult, 1 add
S_5 = (A(0,0) + A(0,1))	(B(1,1))	// 1 mult, 1 add
S_6 = (A(1,0) – A(0,0))	(B(0,0) + B(0,1)))	// 1 mult, 2 add
S_7 = (A(0,1) – A(1,1))	(B(1,0) + B(1,1))	// 1 mult, 2 add

Strassen's Algorithm For Matrix Multiplication

- Note that

$C(0,0) = S_1 + S_4 - S_5 + S_7$ // 0 mult, 3 add
$C(0,1) = S_3 + S_5$ // 0 mult, 1 add
$C(1,0) = S_2 + S_4$ // 0 mult, 1 add
$C(1,1) = S_1 - S_2 + S_3 + S_6$ // 0 mult, 3 add

Strassen total // 7 mult, 18 add
Traditional total // 8 mult, 4 add

- Now recursively apply the same decomposition to each of the 7 new "A" and "B" matrix pairs

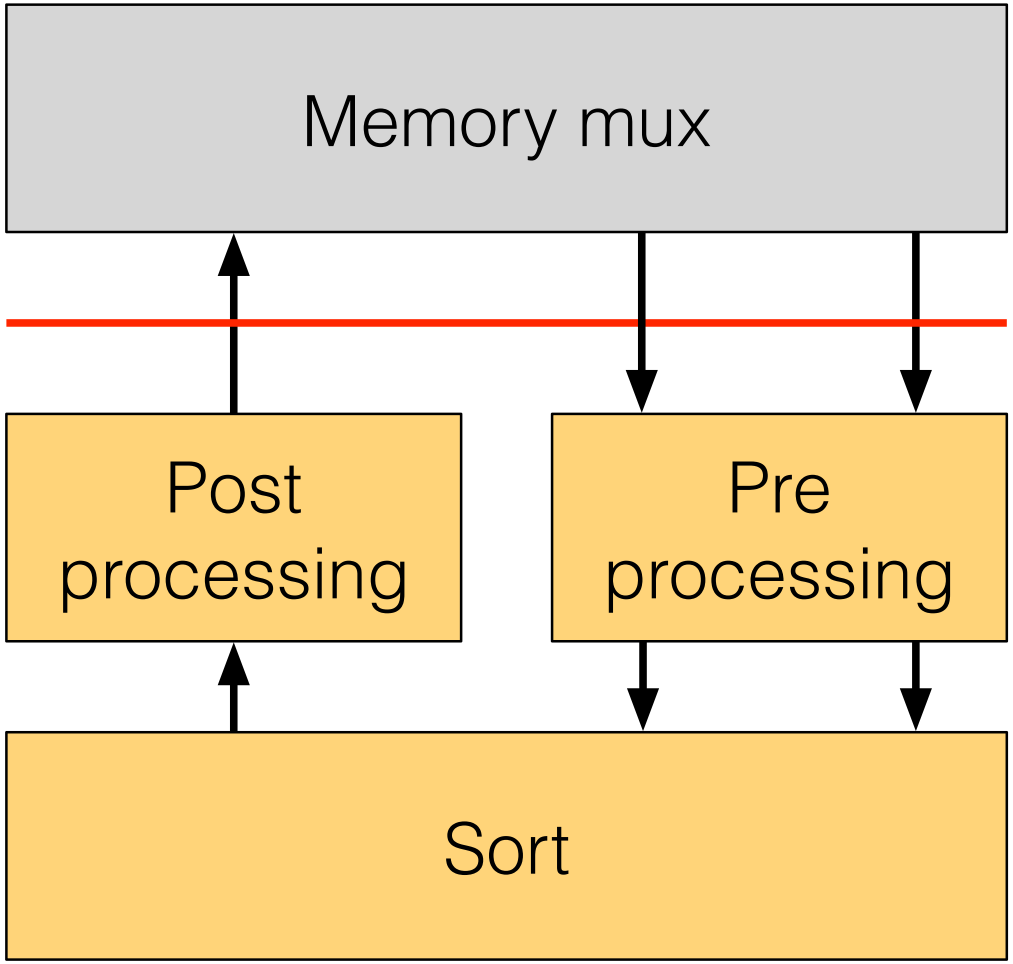
Example Multiplication Primitive Uses

Enabled via appropriate pre and post processing

- Dense linear algebra
  - Matrix multiplication and addition
  - Matrix pointwise multiplication and addition
- Convolution
  - CNN style 2D convolution + ReLU
  - Standard 1D and 2D convolution
- Transforms
  - DFT, FFT and DCT
- Some things you don't expect
  - Clamp
  - Transcendental functions (via series approximations)

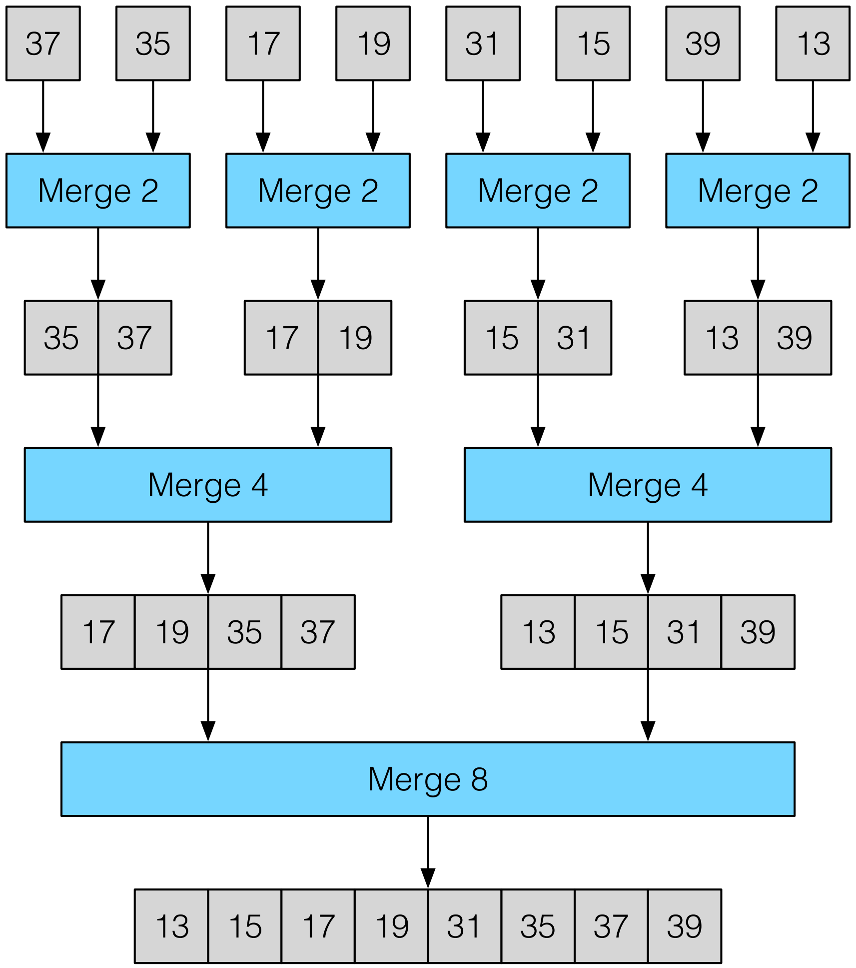
Sort Primitive

- Flow
  - Read two N x 1 vectors each cycle
    - Ex: two new rows for 3 x 3 / 2 max pooling
  - Use pre processing to do formatting for the input
    - Ex: align 3 to 4 via repetition for common sorting
  - Merge sort for a specified number number of stage
    - Ex: two to sort 4 items
  - Use post processing to do formatting for the output
    - Ex: accumulator comparison to sort across rows
    - Ex: keeping max out of each 4 columns
  - Write a N x 1 vector each cycle
    - Ex: maxes in 1 cycle

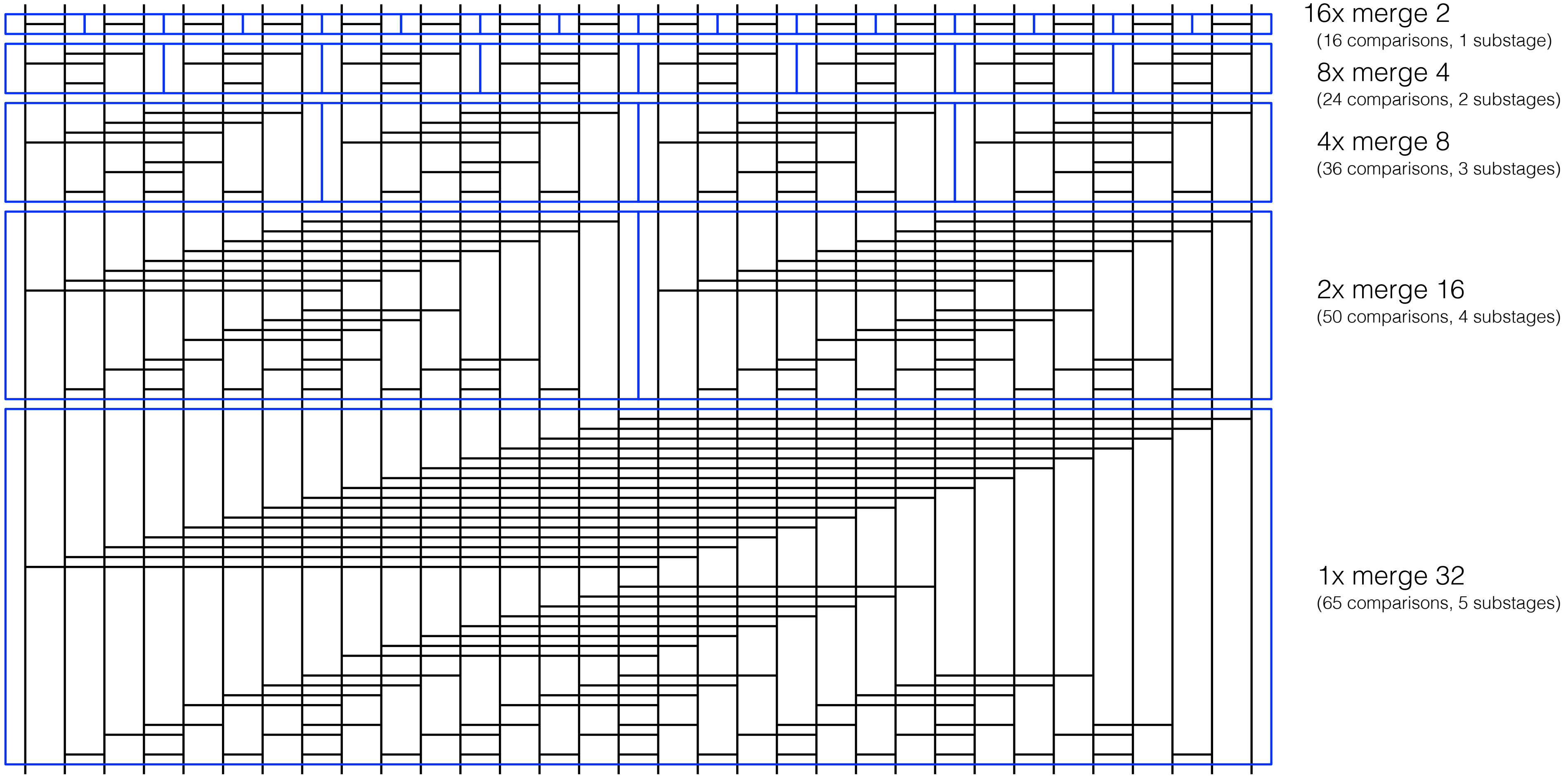


Sort Primitive

- Comments
  - Performance of proposed sorting algorithm is matched to data movement limit
  - Supporting 8, 16 and 32 bit precision can be accomplished with the same bandwidth, memory and compute via appropriate comparator design and using primitive sizes of 1x, 1/2x and 1/4x, respectively



Parallel Merge Sort Style Sorting Network



Example Sort Primitive Uses

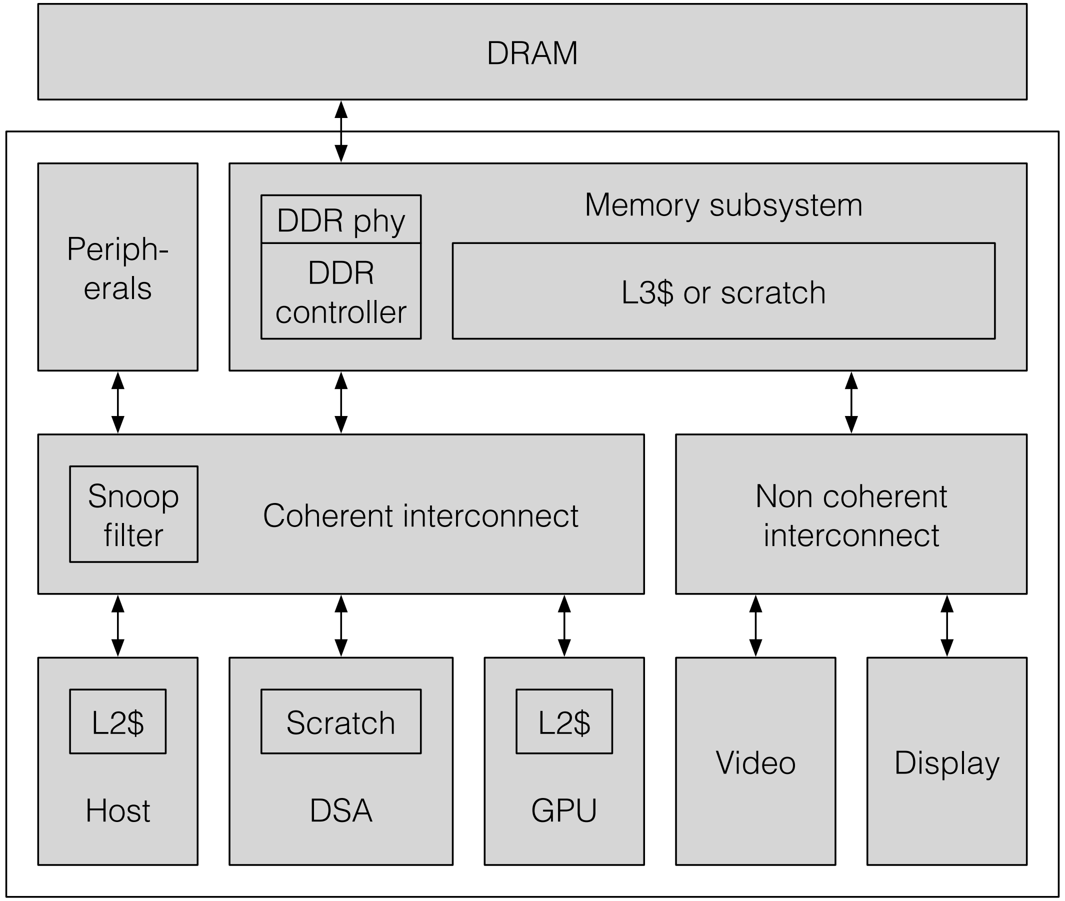
Enabled via appropriate pre and post processing

- Sorting
  - Full, partial
  - 1 vector with another
  - 1D and 2D
- Min and max
- Rank order filter
  - Median and arbitrary
- Pooling
  - Max

xNN Configurations

Individual Device

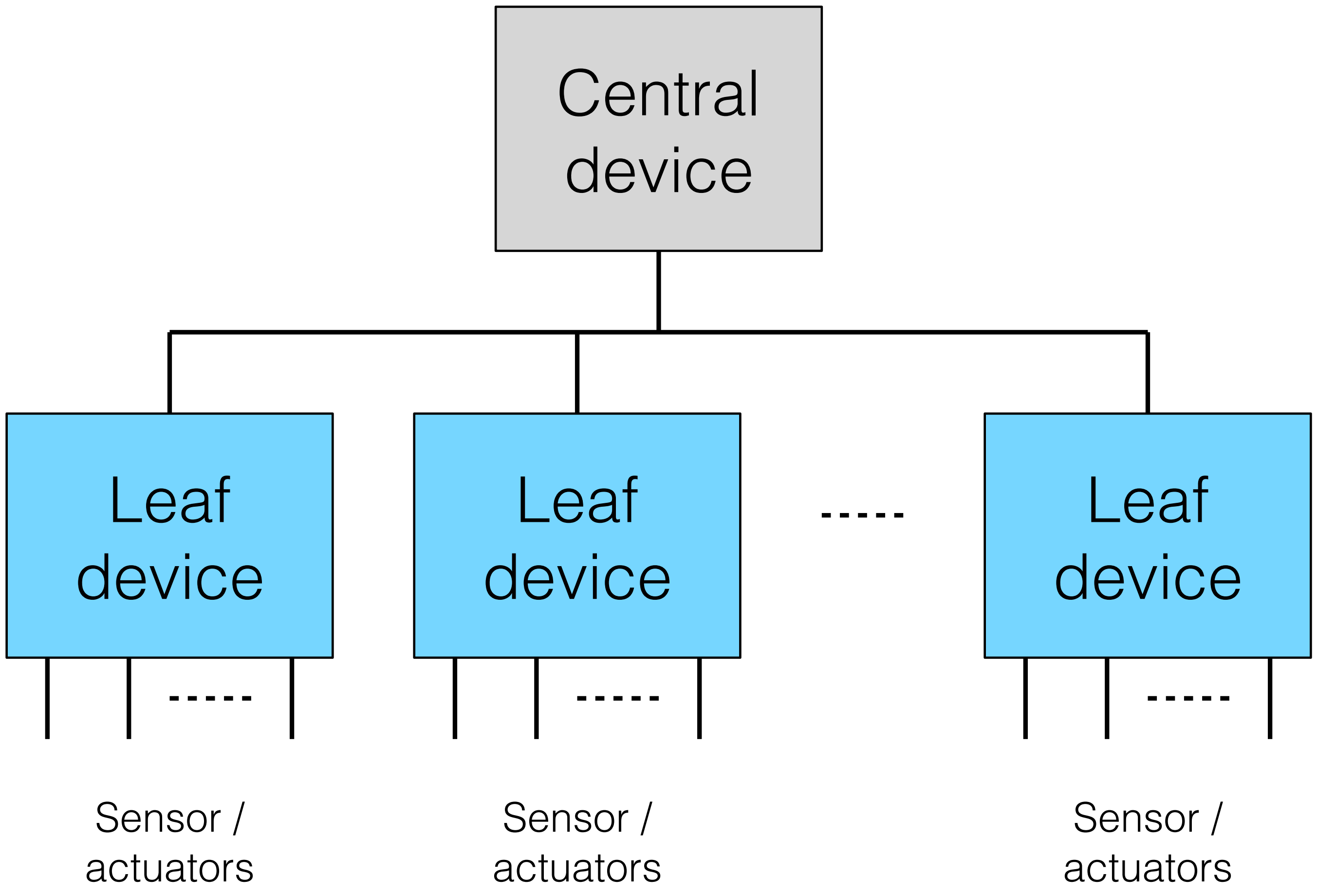
- This is what we've been discussing so far
  - Consider scaling up or down based on the particular application space



Multiple Devices In A Tree Configuration

- Example sensor / actuator use
  - Disjoint sensor processing via xNNs in edge devices
  - Higher level fusion in a central device leading to decisions; possibly xNN based, possibly shallow combination
  - Disjoint actuation in edge devices based on central device decisions

- Example training use
  - Synchronous training

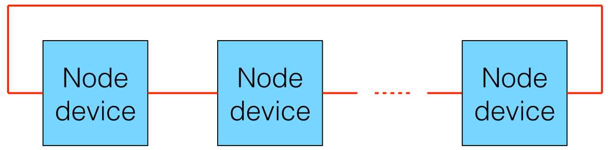


Multiple Devices In A Torus Configuration

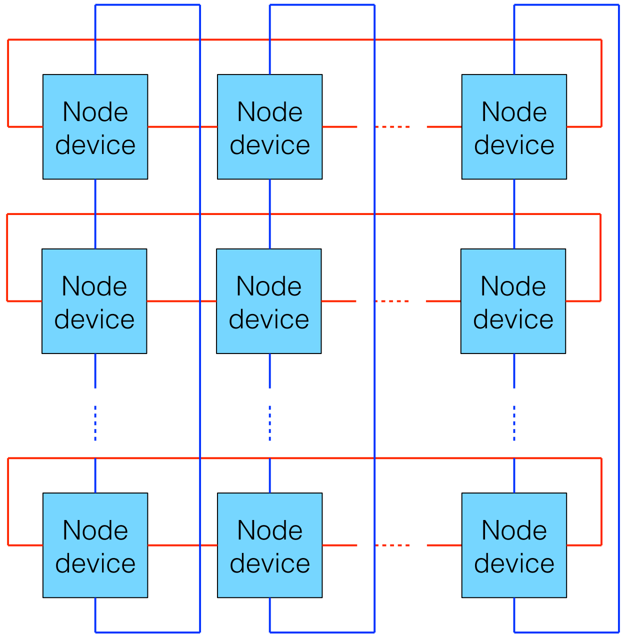
- Devices with 2N network connections in a ND torus

- Example use
    - Larger compute problems

- Note
    - Beyond tree and torus configurations, many other configurations are possible



1D torus



2D torus

xNN Performance

Performance Prediction Vs Benchmarking

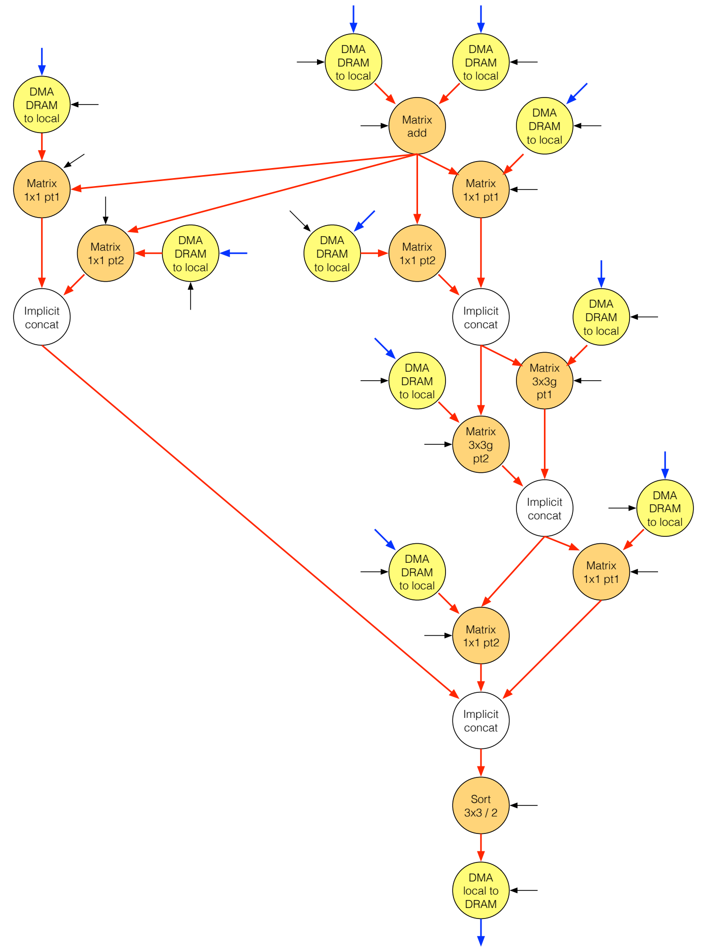
- Performance prediction == **estimating** network / software implementation / hardware implementation performance

- Benchmarking == **measuring** network / software implementation / hardware implementation performance

- Architecture decisions affect performance
  - Not an especially profound statement
  - But perhaps slightly more subtle, memory, data movement and compute choices mean certain systems can perform better / more efficiently in some cases and others in other cases

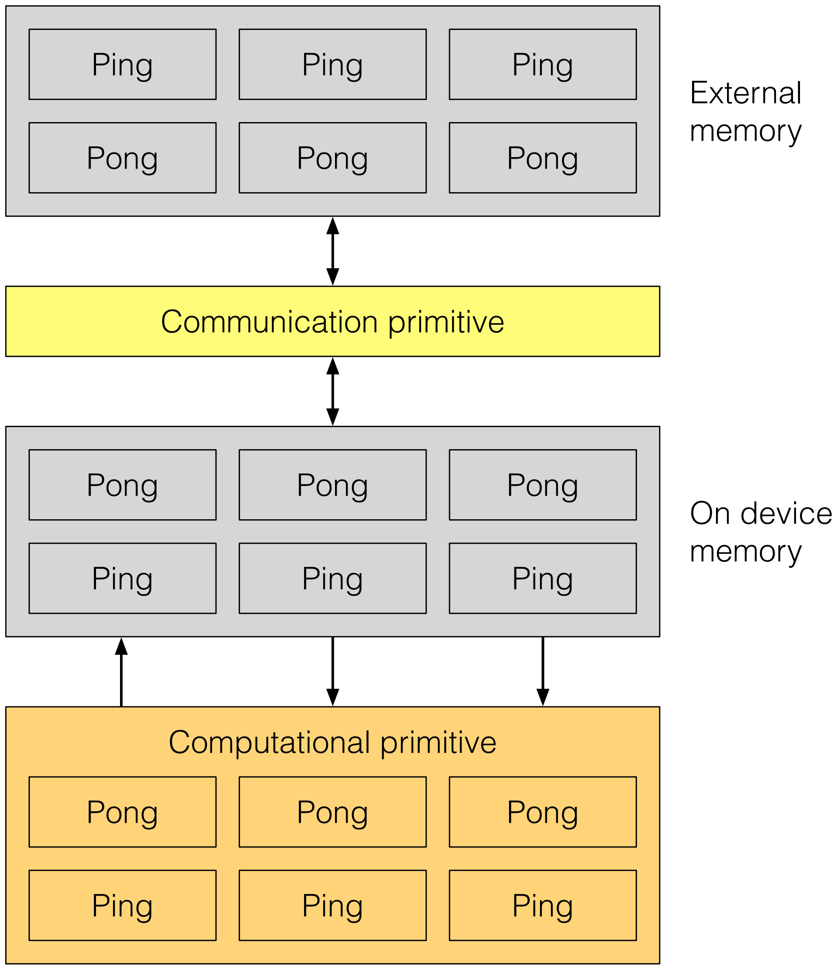
Performance Prediction

- Known from the low level graph
  - Exact order, parallelism and time for each node



An Approximation To Give Intuition

- Choose memory locations
  - Feature maps in local memory if they fit, else external memory
  - Filter coefficients in external memory
- Calculate communication time
  - Input feature maps, filter coefficients, output feature maps
- Calculate computation time
  - Scalar, vector and matrix
- Bound total time per layer
  - Serial bound: communication + computation
  - Parallel bound: max(communication, computation)
- Bound total time for the network
  - Serial bound: sum of serial time for each layer
  - Parallel bound: sum of parallel time for each layer



Backup

Backup – General

Common Data Formats

• 32 bit float	24.8	(IEEE 754 single)	// default for training
• 16 bit float	11.5	(IEEE 754 half)	// less good because of less range
• 16 bit float	8.8	(bfloat16)	// can typically quantize to this as same range
• 8 bit fixed	8.0	(signed and unsigned)	// common for CNN inference
• 4 bit fixed	4.0		// in some HW, not especially interesting
• 1.5 bit fixed	{-1, 0, 1}		// sometimes possible for CNN inference
• 1 bit fixed	{-1, 1}		// maximally compact

There are additional block floating point representations that save storage space and simplify multiplier design but using a common exponent for various tensor sizes

Items That Can Be Quantized

- All memory elements
  - Feature maps
  - Filter coefficients and other parameters
  - Gradients and associated training terms
    - Possible useful for training
    - But also make training more difficult and it's already a hassle
    - So skip for now and focus on quantizing memory used in testing
    - Note that will still cover quantized training
    - Quantized values will be in the forward path though

Why is high precision not always needed in xNNs?

- A linear layer is ~ doing template matching or drawing boundaries
- A loss of precision implies an imperfect template or boundary
- If template matches or classes are sufficiently separated then a bit of noise can be ok
- The question is how much is tolerable
- For different places in the network? Can network modifications be made to make it more tolerable?

- Feature maps and filter coefficients can use different (but compatible) quantization choices
  - Ex: 4 bit signed fixed point filter coefficients and 8 bit unsigned fixed point feature maps

- Need to understand the implications of the reduction in different values that a memory element can take
  - Appropriately balance reduction in range and precision
  - Goal is to maximize efficiency gain and minimize accuracy loss

Fx Pt Quantized CNN Style 2D Convolution

- Start from what can be calculated with fixed point hardware
  - Y_q = clip(round(s_c (H_q X_q + V_q)))
  - Y_q is the fixed point quantized 2D matrix created via re arranging the 3D tensor of output feature maps
  - H_q is the fixed point quantized 2D matrix created via re arranging the 4D tensor of filter coefficients
  - X_q is the fixed point quantized 2D matrix created via a Toeplitz style arrangement of the 3D tensor of input feature maps
  - V_q is the fixed point quantized 2D bias matrix created via an outer product of a bias vector and 1s row vector
- Compute scale s_c
  - s_c is a scale selected to constrain the output fixed point range to the target number of bits
  - Can select a static s_c based on training data
  - Can select a dynamic s_c based on monitoring accumulator values to maximize dynamic range individually for each input (though this requires some downstream adjustments where additions occur)
- Note
  - H_q = clip(round(H / s_H)) ≈ H / s_H static, typically selected to maximize range
  - X_q = clip(round(X / s_X)) ≈ X / s_X static or dynamic, from the previous layer
  - V_q = clip(round(V / s_V)) ≈ V / (s_H s_X) static or dynamic, dependent on the filter and input scale; $s_V = s_H s_X$ to align bias (additive) scale with combined filter and input (multiplicative) scale

Fx Pt Quantized CNN Style 2D Convolution

- Substituting in and ignoring clipping and rounding
  - $Y_q \approx s_c\,((H / s_H)(X / s_X) + (V / (s_H\, s_X)))$
    $\approx (s_c / (s_H\, s_X))\,(H\,X + V)$

- Let $s_Y = (s_H\, s_X) / s_c$ and $Y = s_Y\, Y_q$ then
  - $s_Y Y_q \approx H\,X + V$
  - $Y \approx H\,X + V$

- This implies we can approximate floating point CNN style 2D convolution $Y = H\,X + V$ with fixed point CNN style 2D convolution $Y_q = \text{clip}(\text{round}(s_c\,(H_q\, X_q + V_q)))$ and the following constraints
  - Bias scale $s_V = s_H\, s_X$ is dependent on the filter and input scale; as such, the bias typically uses 2x – 4x the number of bits vs multiplicative parameters
  - Output feature map scale $s_Y = (s_H\, s_X) / s_c$ is a function of the filter, input and compute scales; for convenience of implementation the compute scale s_c may have some constraints (e.g., only powers of 2)
  - Note the coupling of scales from 1 layer to the next
  - Note that typically do accumulation at 4x input scale before compute scale

Fx Pt Quantized Pooling

- Not a big deal
  - For max pooling: the set of possible output values come from the set of input values
  - For avg pooling: the set of possible output values is bound by the range of input values

31	21	33	34	5	2	15
10	29	32	6	27	16	13
7	4	28	20	24	30	26
25	18	14	35	22	1	3
17	23	12	8	19	9	11

Max pool ↓ 3x3 / 2

33	34	30
28	35	30

32	36	68	56	72	48
8	40	64	84	80	12
28	96	92	76	16	4
52	88	44	20	60	24

Avg pool ↓ 2x2 / 2

29	68	53
66	58	26

Fx Pt Quantized N Input 1 Output Operations

With additive (definitely) or concatenative (probably) operations

- A bit of a hassle
  - Need to align scales of inputs
  - A simple way to do this is to us the largest scale as a common scale to bring all other scales to, then perform the operation

- Examples
  - Element wise addition (as in ResNet)
  - Concatenation (as in GoogLeNet / Inception, DenseNet, ...)
  - After grouped convolutions (if different scales are used for different groups)

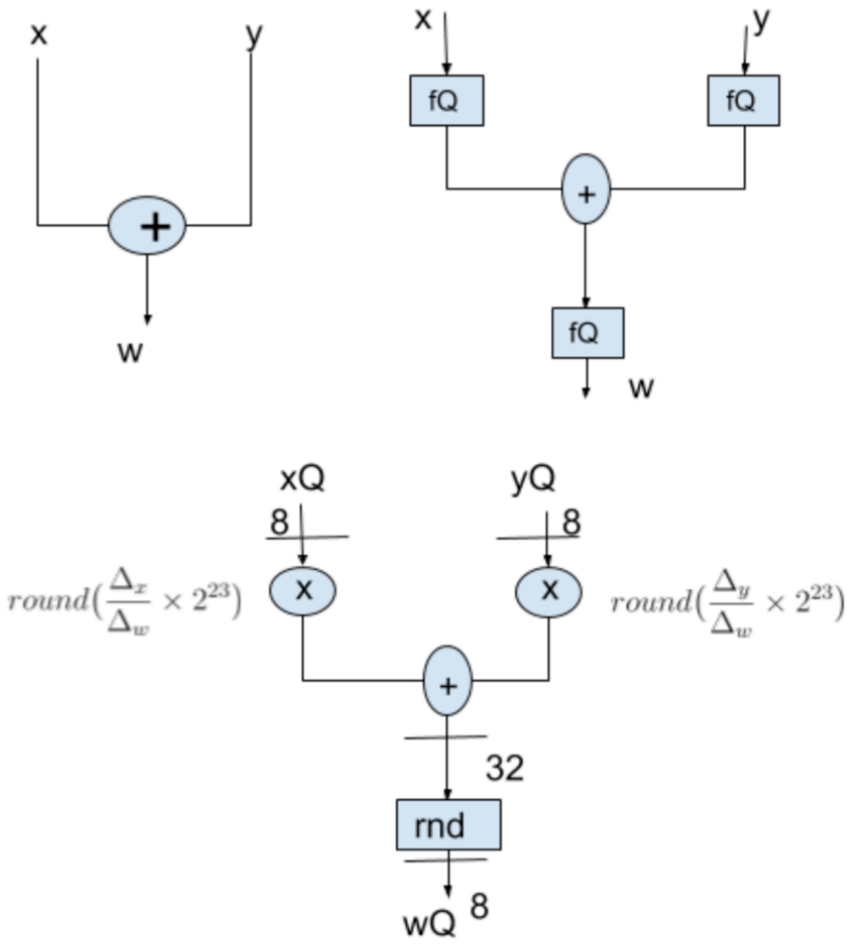


Figure from https://arxiv.org/abs/1806.08342

Fx Pt Quantized Batch Norm

- A bit of a hassle
  - Because it's data dependent
  - But on the upside it's only needed during training
  - Typically do via 2 passes through convolution

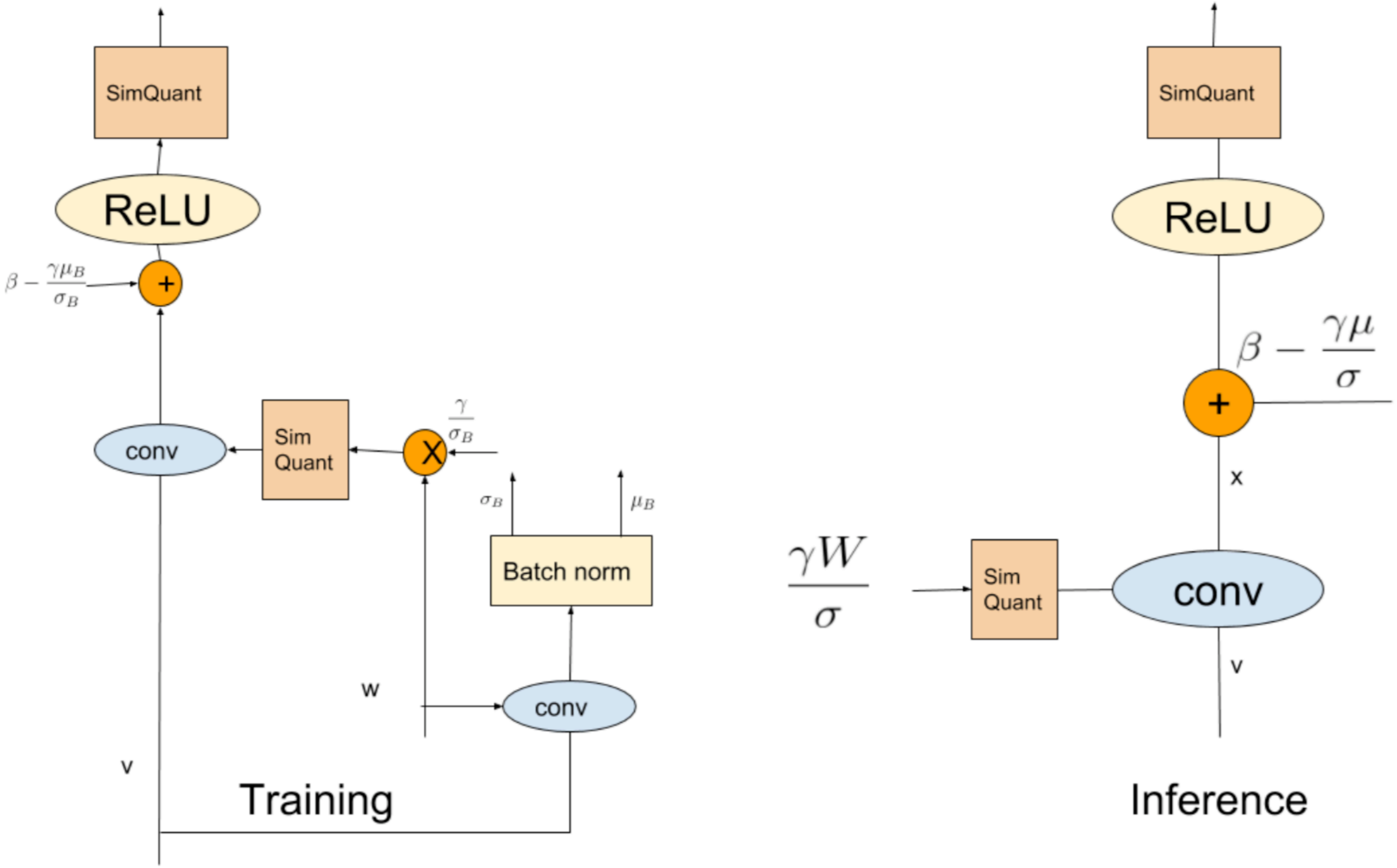


Figure from https://arxiv.org/abs/1806.08342

Range Options

- Previous examples selected scales such that we don't clip but that's not the only option
  - Clipping becomes more useful for static compute scale selection

- No clipping
  - Signed
    - -MaxAbs to MaxAbs (parameters and feature maps in general)
    - Min to Max (implies extra operations, get at most 1 extra bit)
  - Unsigned
    - 0 to Max (feature maps after ReLU)
    - Min to Max (implies extra operations)

- Clipping
  - For traditional signal processing set clip value to maximize SNR but it's different for CNNs
  - Data space is usually smooth but feature space is spikey with many small values
  - SNR as an evaluation metric is the wrong criteria by itself
  - Really care about information and final accuracy

Simulating Fx Pt In Floating Pt Hardware

- A common trick is to introduce quantize – un quantize blocks into the graph
  - Quantize X_q = round(X / s_X) a tensor to introduce loss of information
  - Un quantize the tensor $X \approx s_X X_q$ to bring back to original range
  - Perform operation
  - Take care to properly handle N input 1 output and batch norm operations

xNN Graph Specification

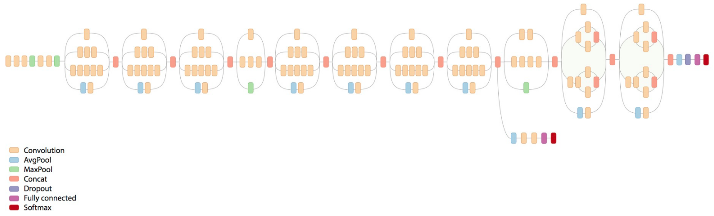
- Graph specification
  - The network is specified as a high level hardware agnostic directed acyclic graph
  - For training a data source, data path, error and update method are specified
    - The tool adds nodes to create the error gradient path and weight updates
  - For testing a data source, data path and output are specified
  - See the backup – software slides for more information on this, specifically with respect to the details of various popular frameworks
- Different execution methods
  - Graph execution: initial software had separate steps for graph specification, graph compilation and graph execution; this is optimal from a performance perspective but counterintuitive to typical development expectations in an interpreted language
  - Eager execution: later software packages followed the expectations of an interpreted language and executed code as written; this is better from a development perspective but sub optimal from a performance perspective
  - Recent frameworks are moving towards supporting both styles

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```



Figures from https://www.tensorflow.org/tutorials/ and https://ai.googleblog.com/2016/08/improving-inception-and-image.html

xNN Graph Specification

- On the topic of memory
  - The locations of tensors can be static or dynamic
  - The contents of tensors can be static or dynamic
  - The management of tensors can be internal or external
- This information needs to be either available to or infer ably by the graph compiler for optimal performance
- Examples
  - Different input images (dynamic contents) during testing can be in different places in DRAM (dynamic location) determined by a host application (external management)
  - Trained weights (static contents) during testing can be in the same place in DRAM (static location) determined by the graph compiler (internal management) within an allocated memory pool

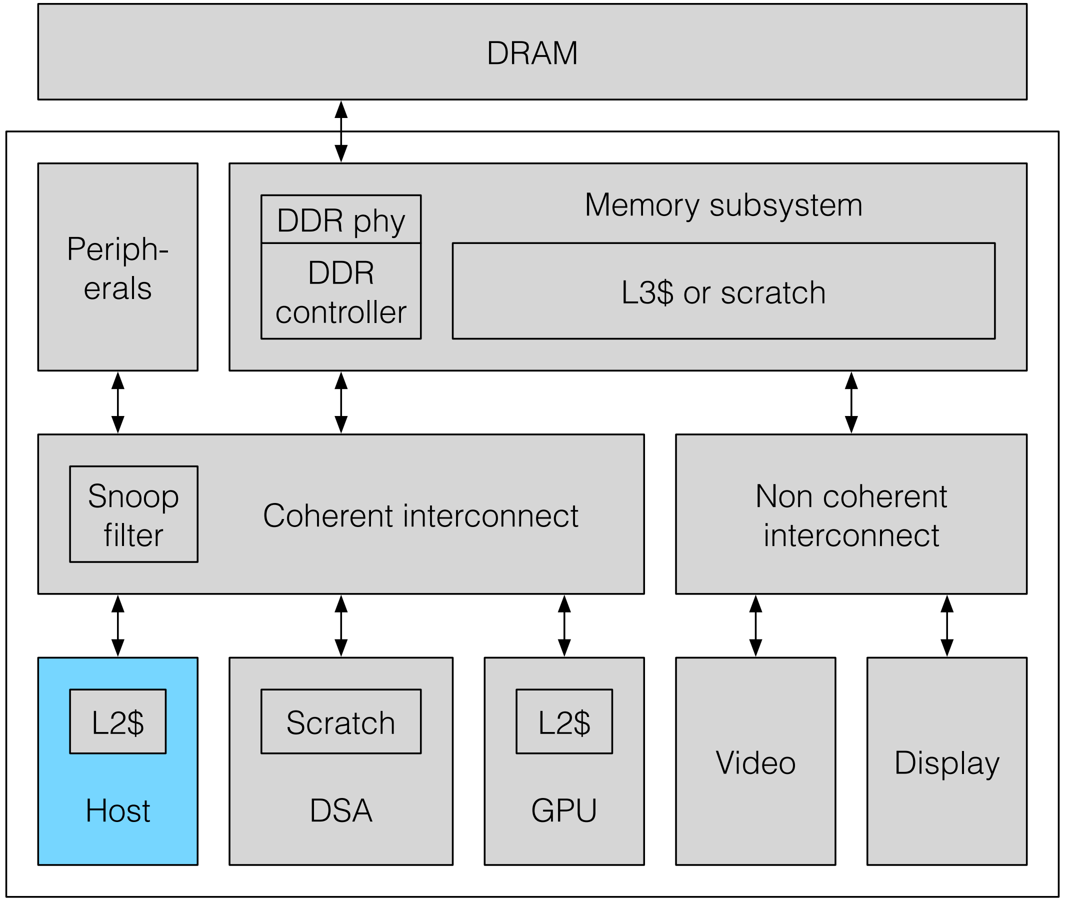
Model Export

- Frameworks work with models in different formats
  - During training this is typically an in memory representation
  - After training this is typically thought of as a static graph
  - Use the word conversion when it’s not the native format

- Sometimes some graph optimization is done during the export process

CPU

- General compute that can do everything
  - Example architectures: x86, ARM, RISC-V, ...
  - This is what you want for optimizing the performance of large amounts of runtime dynamic hardware agnostic code
  - This is not what you want for optimizing the performance of small amounts compile time static hardware specific code
  - This is 1 possible get out of jail free card in the event that some portion of a network design does not map to the DSA
- Intelligence in hardware == limited optimization horizon, extra power, extra area and lower frequency
  - Cache
  - Branch prediction
  - Out of order processing
  - Speculative execution



DSA Architecture Vs CPU Architecture

- CPU example internal memory
  - Registers
  - L1D$, L0I$, L1I$
- CPU example external memory
  - L2 cache, L3 / L3$, DRAM
- CPU example control
  - Front end
    - Fetch
    - Branch prediction
  - Decode / rename / commit
    - Decode, Rename, Dispatch

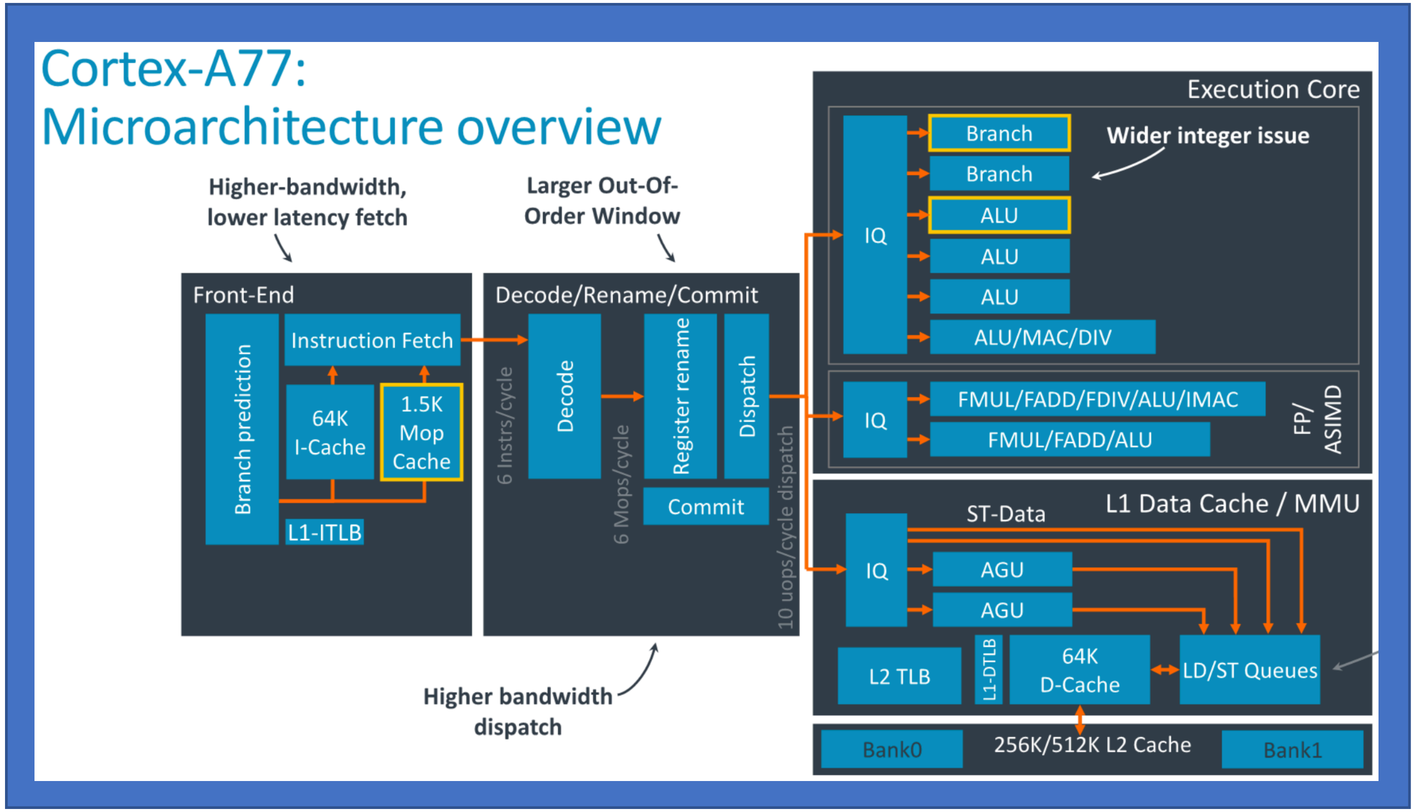


Figure from https://www.anandtech.com/print/14384/arm-announces-cortexa77-cpu-ip

DSA Architecture Vs CPU Architecture

- CPU example communication
  - MMU
    - Address generation
    - Load / store
- CPU example computation
  - Execution core
    - Integer branch
    - Integer ALU
    - Floating point

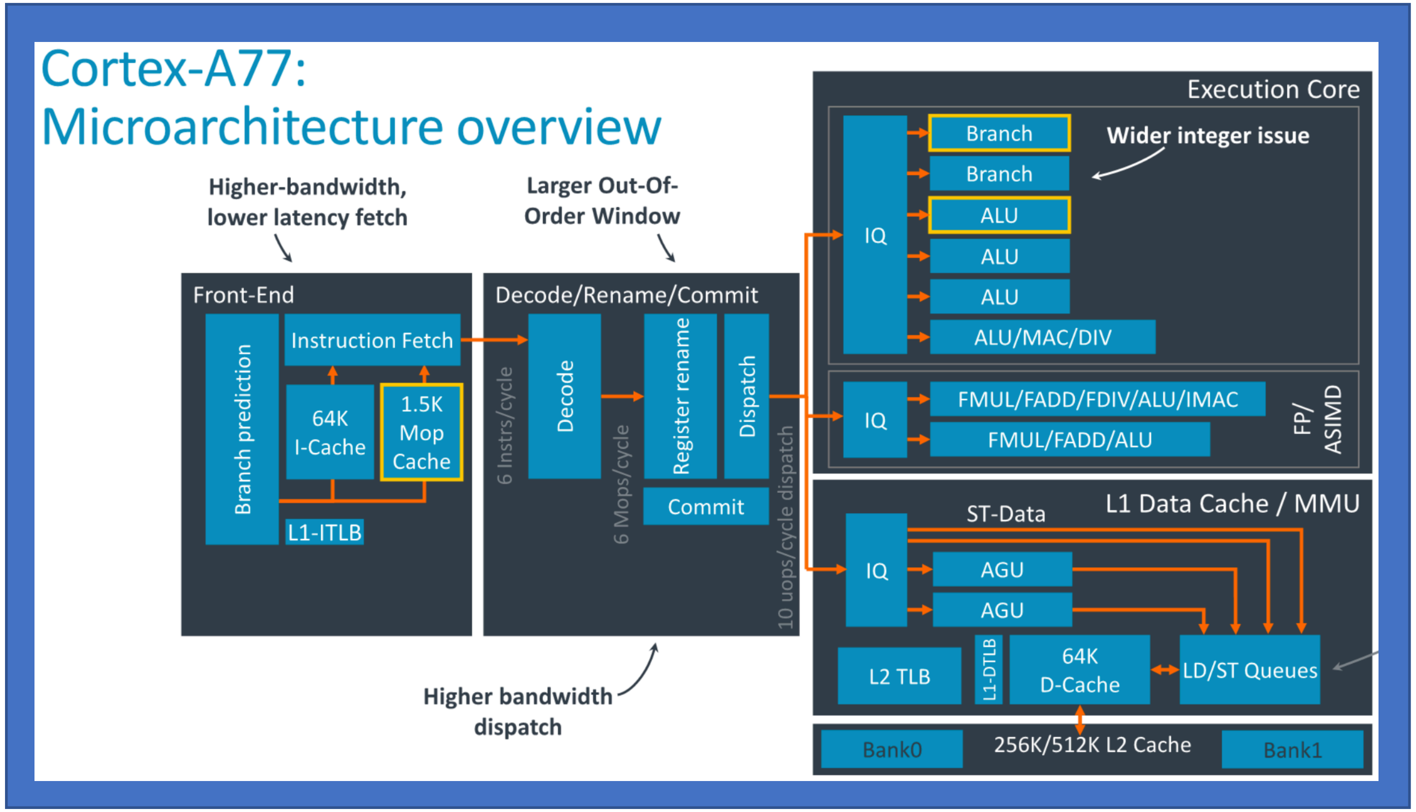
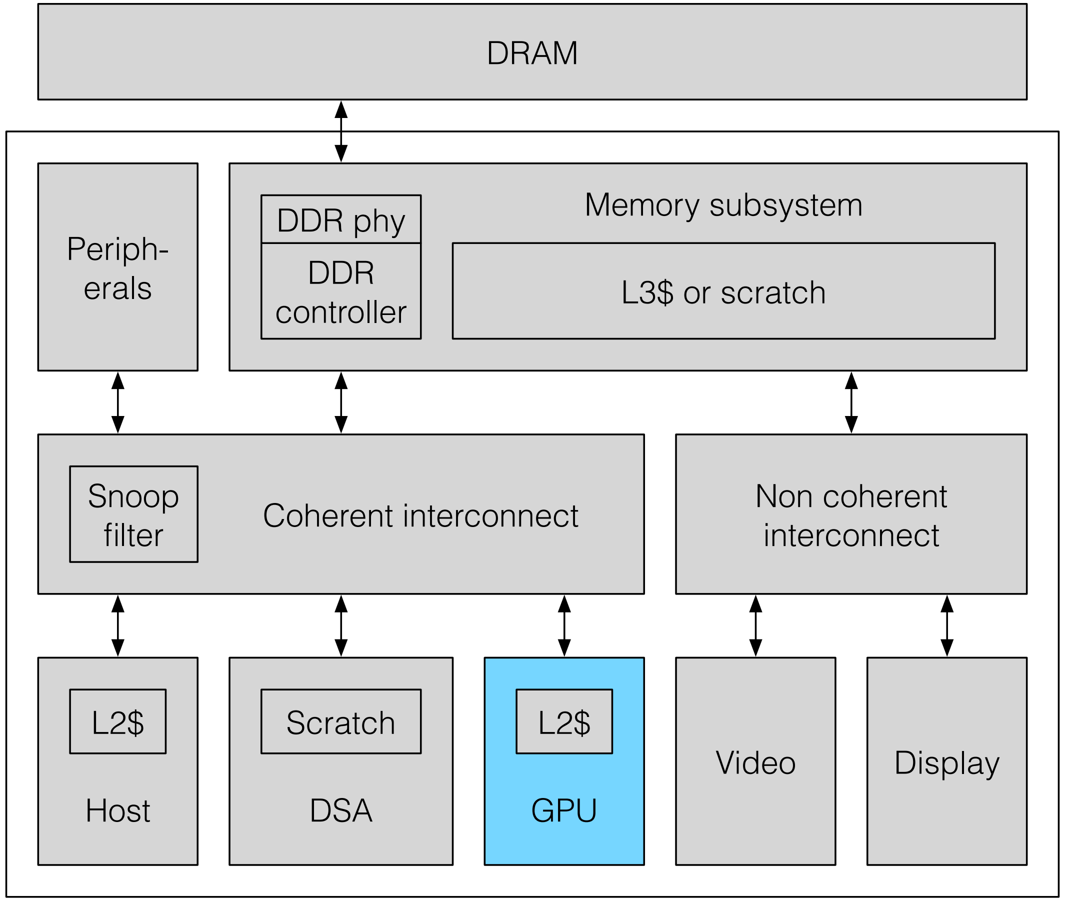


Figure from https://www.anandtech.com/print/14384/arm-announces-cortexa77-cpu-ip

GPU

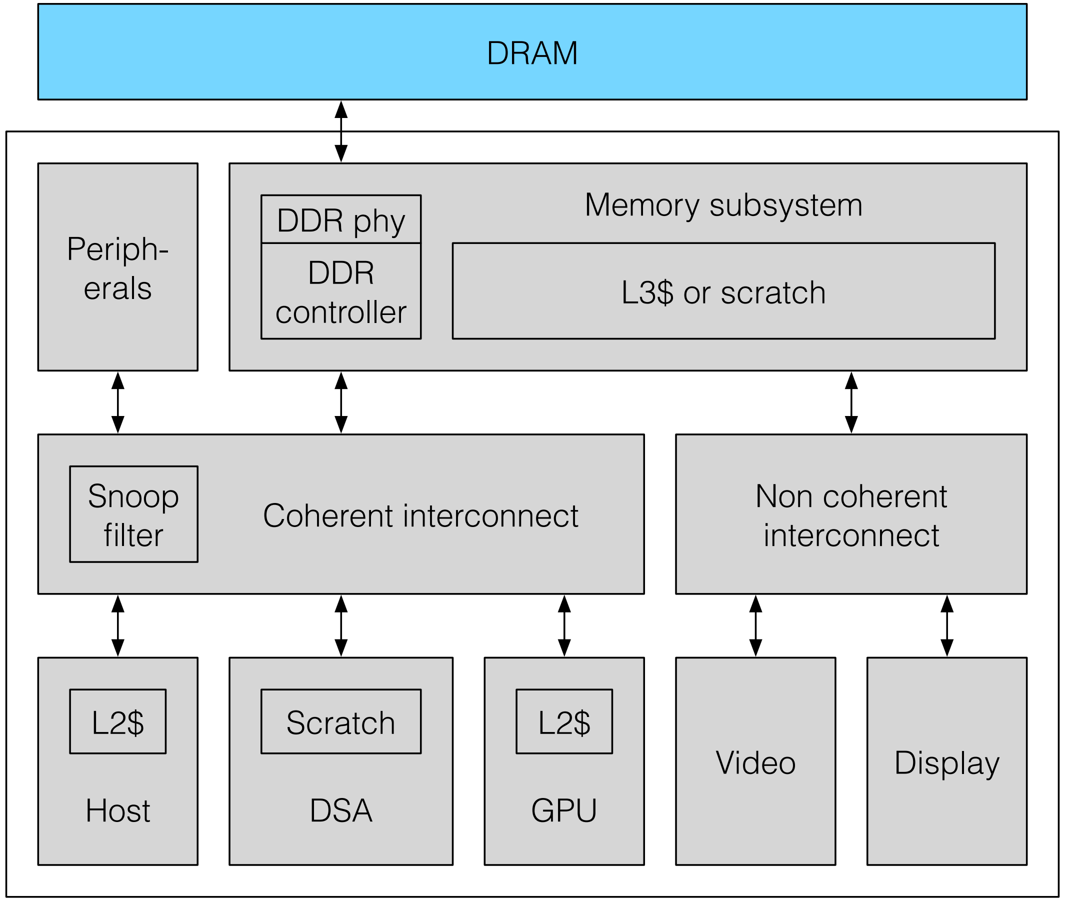
- A GPU is optimal for figuring out what pixels to put on a screen

- A GPU is not optimal for large matrix operations with static compile time graphs
  - Lots of parallel 3x3, 3x4 and 4x4 matrix multiplication is ok but suboptimal in terms of a dedicated architecture (re: N/3 compute to data movement ratio)
  - But in the absence of access to optimized hardware it's a convenient mechanism for training CNNs and it makes up for it architecture shortcomings with sheer size for applications that are less power constrained
  - And you can always use it to play video games when you're not training



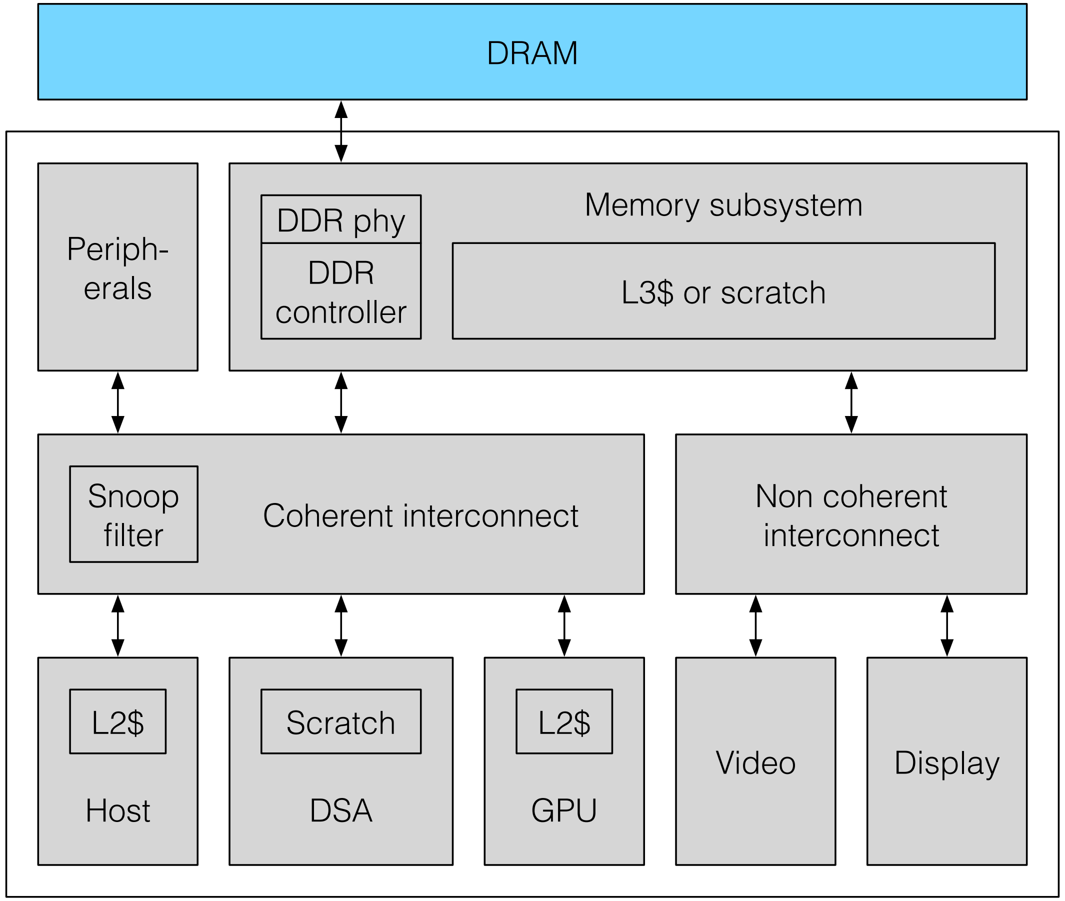
External Memory

- DRAM
  - Use for off device volatile data storage
  - Most common types are DDRx SDRAM
- Memory cell
  - Data is stored as charge on a capacitor representing a bit
  - Memory cells require 1 transistor and capacitor per bit
  - Because charge leaks from the capacitor DRAM needs an external circuit to continually refresh the data
- Organization
  - (Banks * rows * columns) x bits
  - Commonly most efficient with ~ 64 B alignment and multiple of 64 B accesses; specific alignment and access size is a function of the specific memory
  - This affects data arrangement and memory accesses



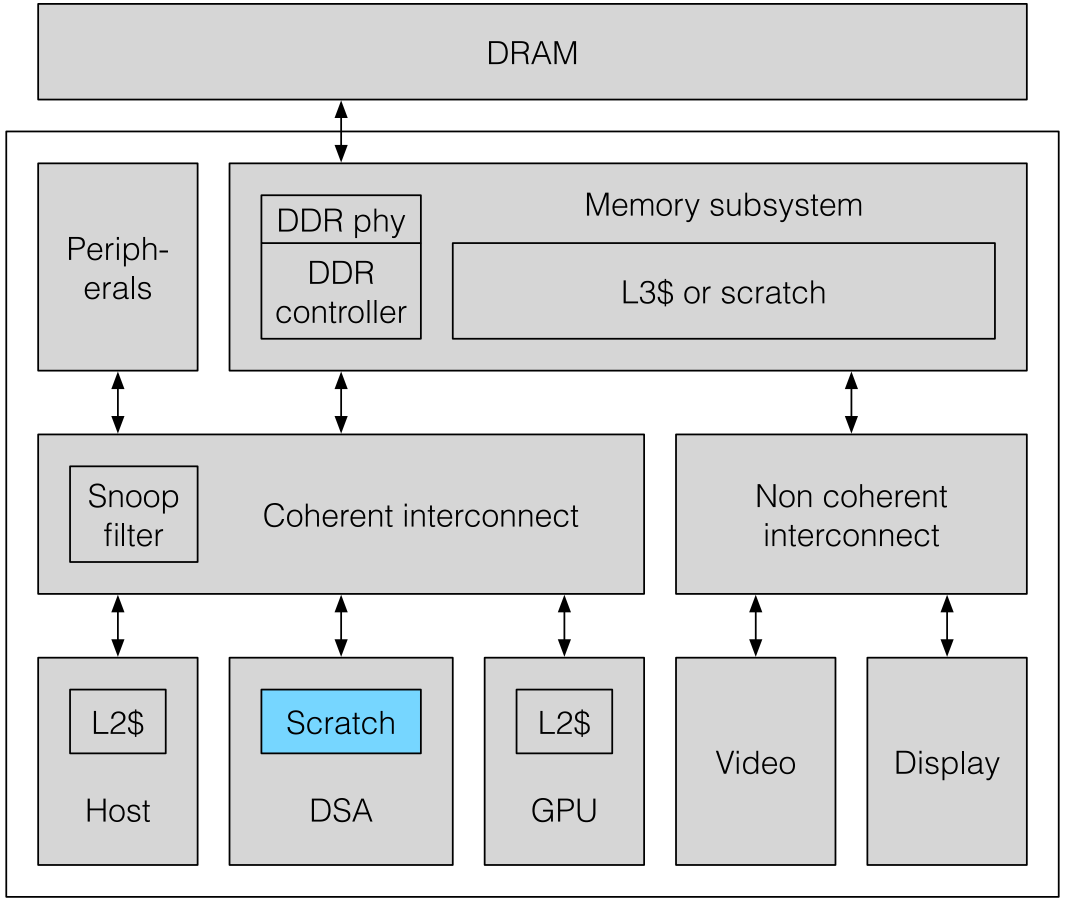
External Memory

- Comments
  - High off device latencies can usually be hidden in throughput optimized compute using on device memory
  - Cheaper per bit but slower than SRAM
- Typical uses of external memory for xNNs
  - For our purposes assumed to be ~ infinite (unless working with extremely large networks or extremely small systems)
  - Dynamic network inputs (unless coming from a direct peripheral interface or another linked graph in the session)
  - Dynamic network outputs (unless a linked graph in the session)
  - Filter coefficients (unless all very small)
- Occasionally uses of external memory for xNNs
  - A buffer for intermediate feature maps when they're too big to fit on device



Internal Memory

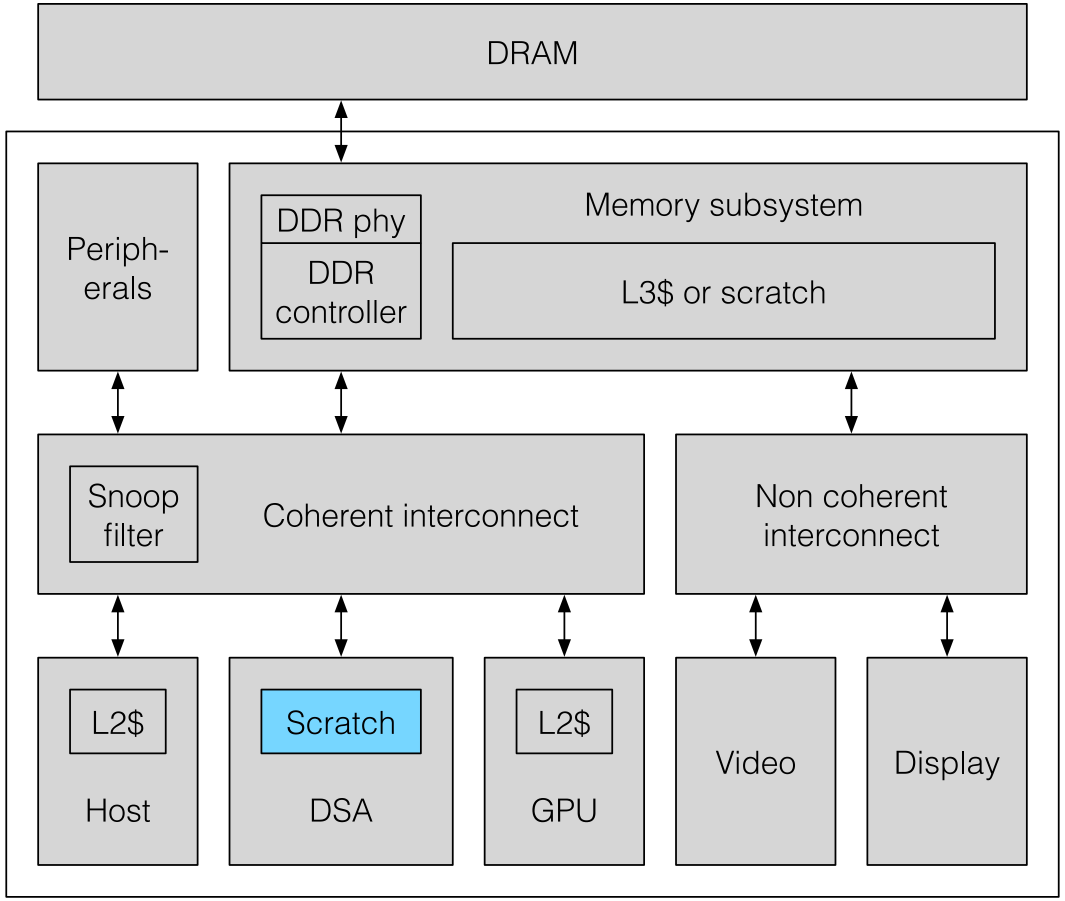
- SRAM
  - Use for on device volatile data storage
  - Compute out of SRAM (maybe 1 step removed)
- Memory cell
  - Data is stored in a flip flop representing a bit
  - Memory cells require 4 - 6 transistors per bit
- Organization
  - Divided into multiple banks where each bank can be thought of as a 2D array of bits / bytes
  - Access are most efficient that read a row at a time
  - Applications spread data across multiple banks for multiple simultaneous read / write operations
  - Either use bank randomization or coordinated memory arrangements to minimize delays caused by multiple simultaneous read / write operations to the same bank



Internal Memory

- Example
  - 2^4 = 16 banks of 2^10 = 1024 rows of 2^6 = 64 bytes
  - Total memory = 2^4 * 2^10 * 2^6 = 2^20 ~ 1 MB
  - Up to 16 parallel accesses are possible (for single port designs, though typically would design for many fewer simultaneous read / write access to avoid collisions and there are also implications wrt the memory mux / switch connecting memory banks to IPs)
  - Accesses are most efficient when the starting address is a multiple of 64 bytes and 64 bytes are read at a time

- Typical uses of internal memory for xNNs
  - Finite (though frequently occupying a large fraction of an optimal device)
  - Input and output feature maps for internal graph edges
  - Filter coefficients for the current layer



Amdahl's Law

- Define
  - p = the fraction of tasks in a program benefiting from acceleration
  - s_{task} = speedup of the task
  - $S_{program}$ = speedup of the whole program
- Amdahl's law
  - $S_{program} = 1 / ((1 - p) + (p/s_{task}))$
- xNNs have many layers
  - CNN style 2D convolution dominates the compute of CNNs and to a 1st order approximation you should do everything you can to make it run as fast as possible (give it most bandwidth)
  - But if you get really really good at making that go fast, it's possible for other operations to start to become a more significant part of the execution time
  - It's why we put control and communication in parallel
  - It's why you may want to have the option of allocating bandwidth to pool completed output feature maps while the matrix primitive is finishing up other output feature maps

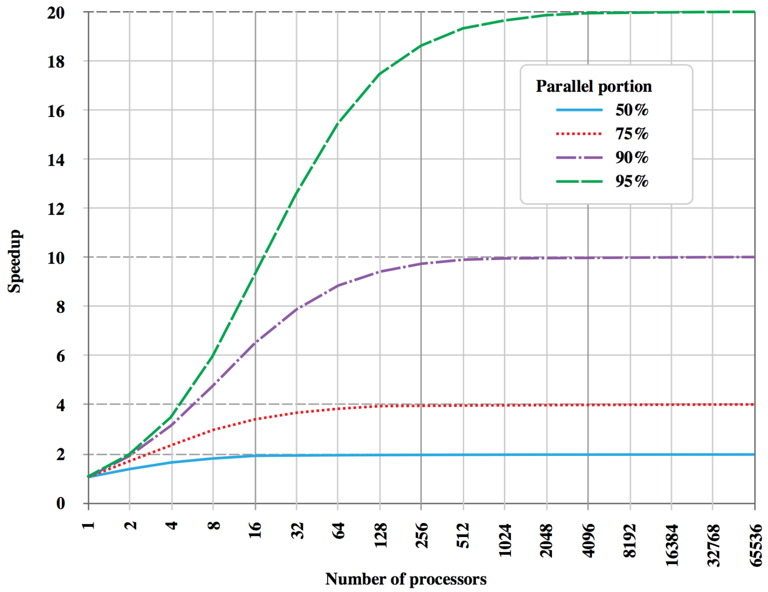


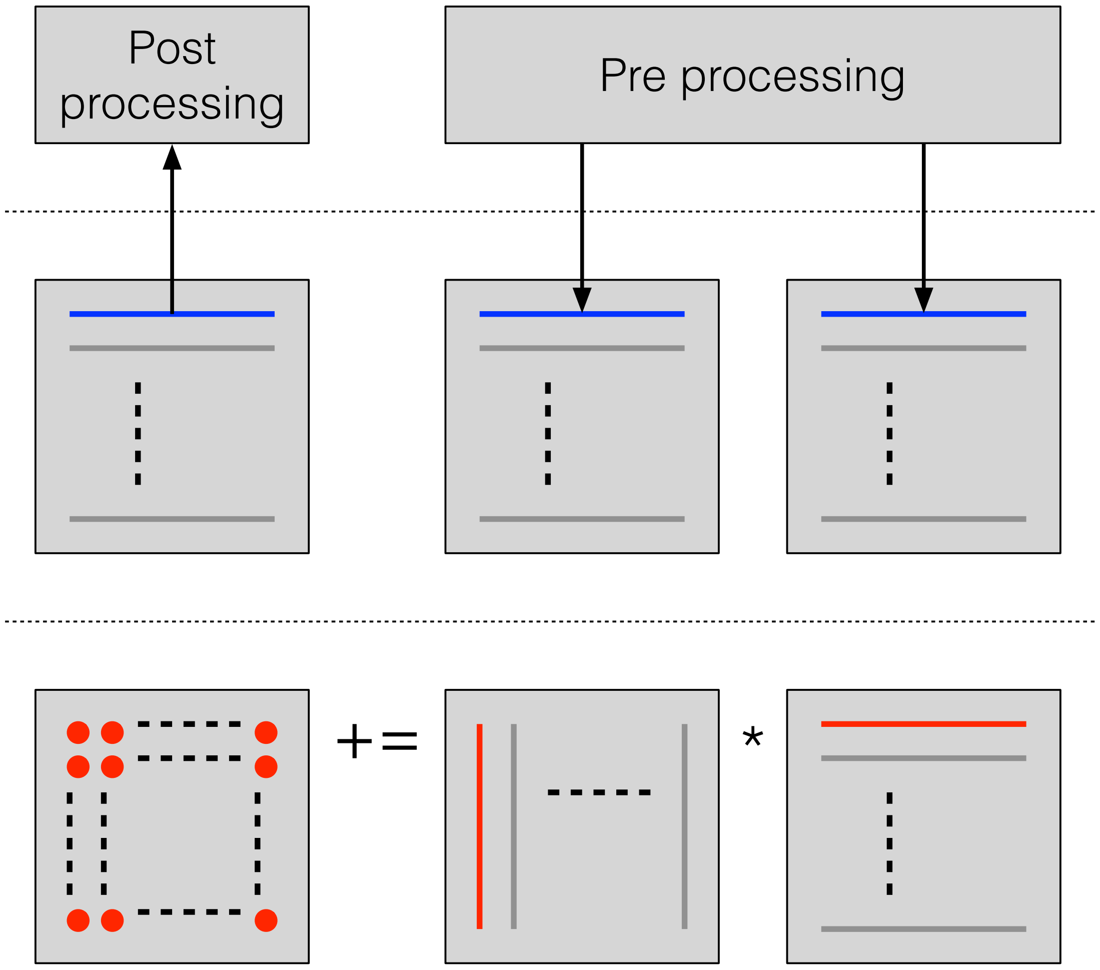
Figure from https://en.wikipedia.org/wiki/Amdahl%27s_law

Outer Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via store ordering of A or back
  - B is in linear order
- Per cycle transfer data
  - Read A(:, k) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute all partial outputs C(:, :)

```
C = C0                  // e.g., bias matrix
For k = 0 to K - 1      // k = 0
  For m = 0 to M - 1
    For n = 0 to N - 1
      C(m, n) += A(m, k) B(k, n)
    End
  End
End
```

Parallel

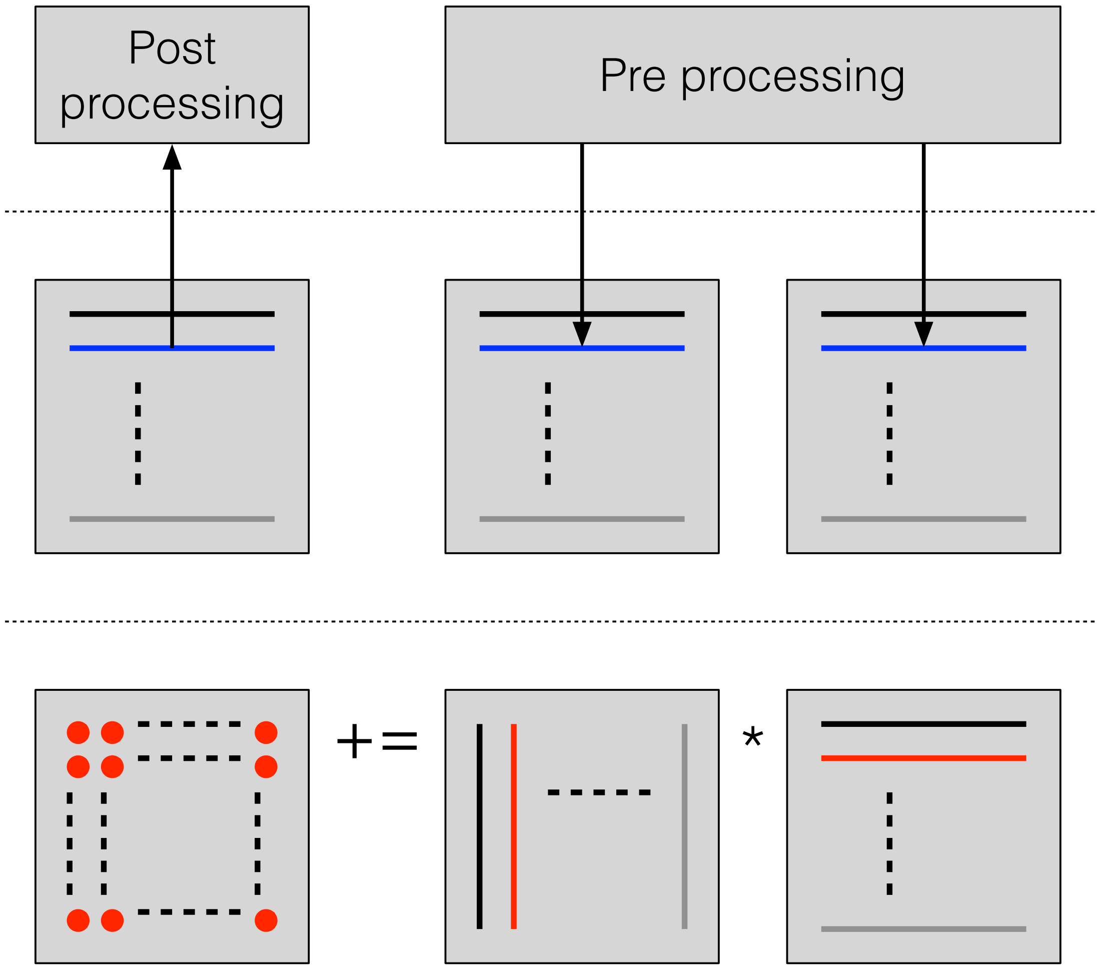


Outer Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via store ordering of A or back
  - B is in linear order
- Per cycle transfer data
  - Read A(:, k) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute all partial outputs C(:, :)

```
C = C0                  // e.g., bias matrix
For k = 0 to K - 1      // k = 1
  For m = 0 to M - 1
    For n = 0 to N - 1
      C(m, n) += A(m, k) B(k, n)
    End
  End
End
```

Parallel

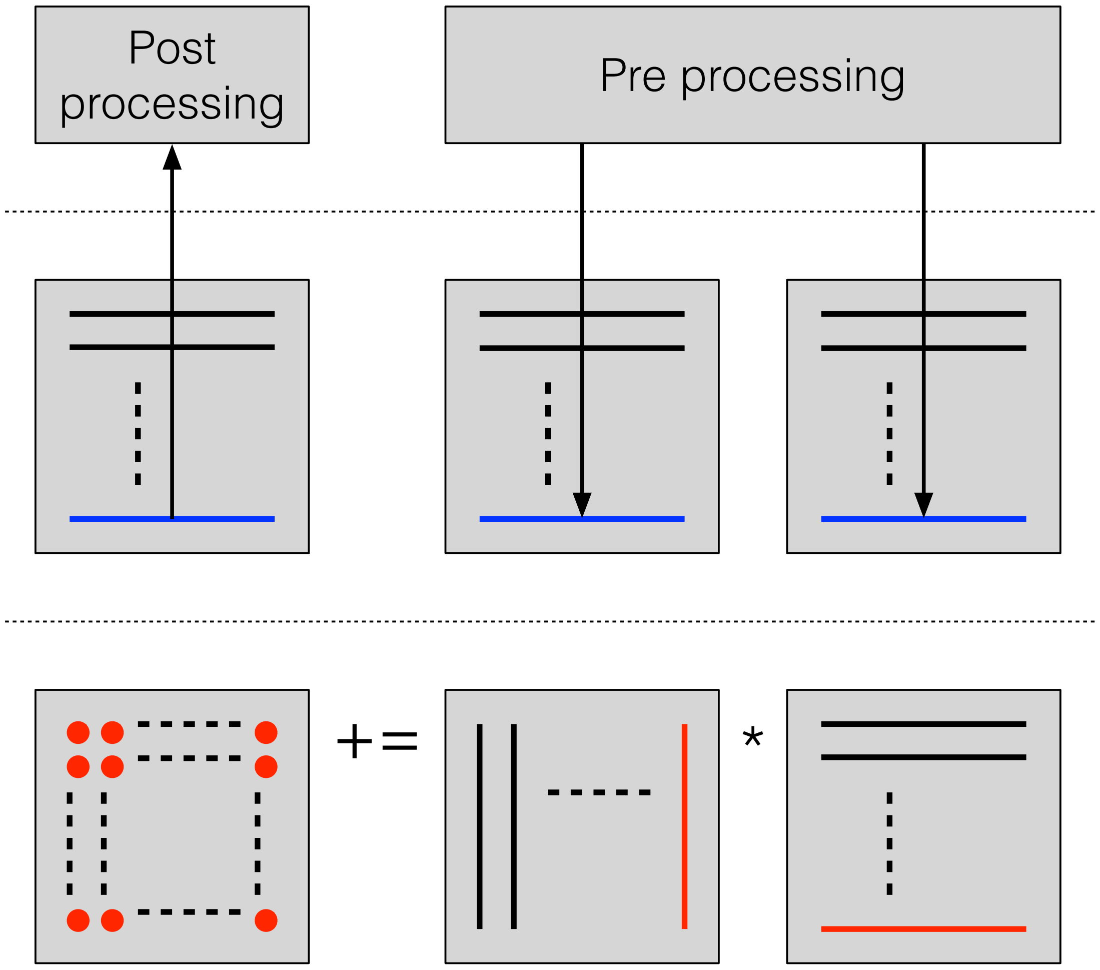


Outer Product Based Matrix Multiplication

- Mathematically it's 3 loops
  - A is needed in transpose order
    - Transpose = bad for typical memory accesses
    - So handle this via store ordering of A or back
  - B is in linear order
- Per cycle transfer data
  - Read A(:, k) and B_{back}(k, :), write C_{back}(m, :)
- Per cycle compute all partial outputs C(:, :)

```
C = C0                     // e.g., bias matrix
For k = 0 to K - 1         // k = K - 1
  For m = 0 to M - 1
    For n = 0 to N - 1
      C(m, n) += A(m, k) B(k, n)
    End
  End
End
```

Parallel



Winograd Style Convolution Algorithm

- Related side note

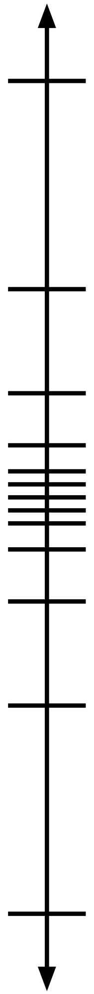
- Similar to fast algorithms for multiplying complex numbers, FFTs, matrix multiplication, ... Winograd figured out a fast algorithm for convolution

- A modified variant has been applied to CNNs, is used in some libraries and is appropriate for some architectures

- Fast algorithms for convolutional neural networks
  - https://arxiv.org/abs/1509.09308

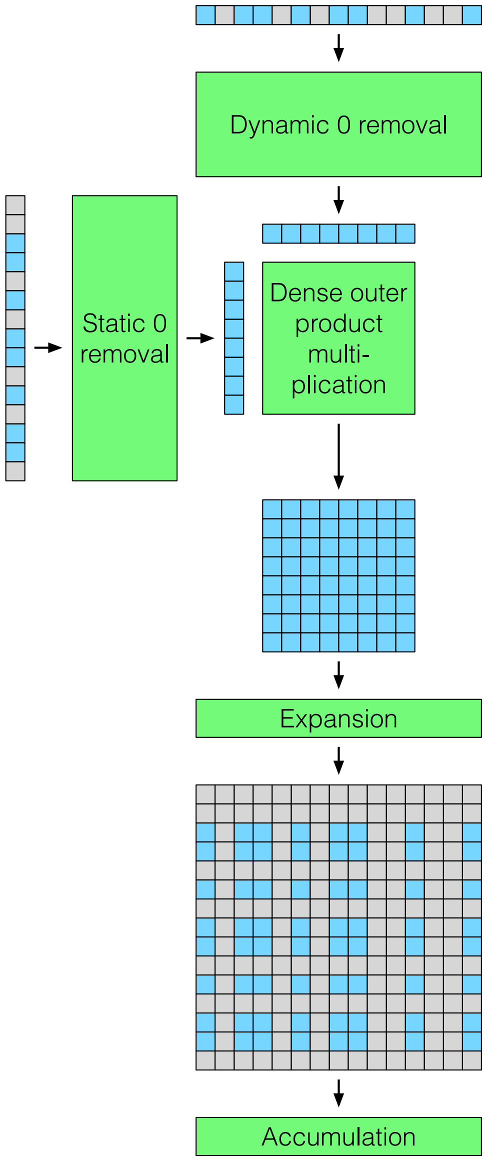
Input Power Of 2 Matrix Multiplication

- Fixed point methods described on earlier slides used quantization schemes with a uniform spacing between levels
- Possibility
  - Non uniform quantization of multiplicative filter coefficients (leave biases as arbitrary 32 bit fixed point values)
  - Choose non uniform levels to be powers of 2, include 0 too
  - Why: multiplication of the filter coefficient with the feature map becomes a simple shift and add
  - Much less complexity than normal multiplication
- A challenge of this is the distance between the larger values
  - Definitely requires some additional re training
- ShiftCNN: generalized low-precision architecture for inference of convolutional neural networks
  - https://arxiv.org/abs/1706.02393



Sparse Matrix Multiplication

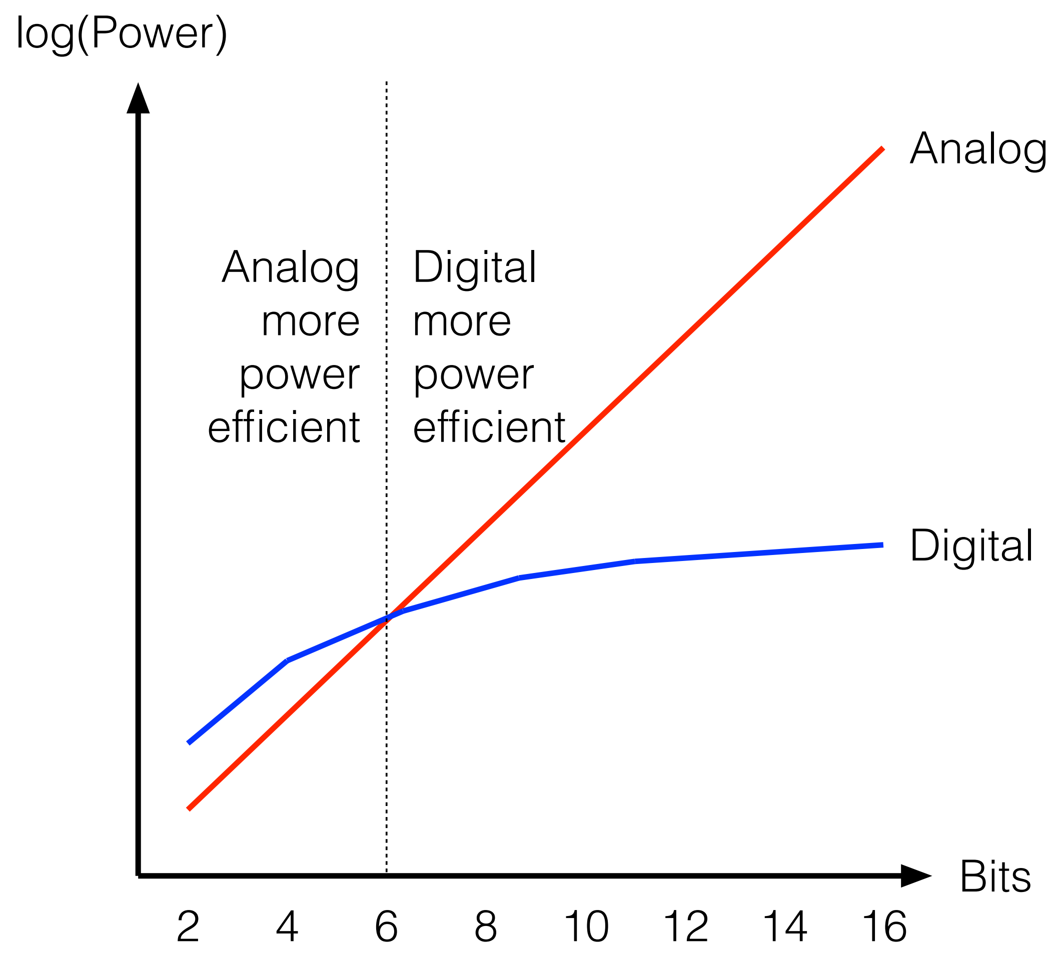
- Matrix multiplication methods on previous slides were designed for dense matrix multiplication
- However
  - Around 50% of feature map values are 0 (dynamic)
  - It's also possible to force sparsity in filter coefficients (static)
    - Possibly with some implications to accuracy
    - Not fully clear if it's better than having a smaller dense set of filter coefficients
    - But regardless, it's a thing people can do
- It's possible to take advantage of this to improve matrix multiplication
  - Similar to Sparse BLAS to BLAS, the higher the level of sparsity the higher the potential advantage
  - Traditionally in high performance compute things are dense or very sparse (e.g., 1/100); in xNNs sparsity can be between these points which leads to different methods for index tracking
- Can implement via static / dynamic compression around an outer product based dense matrix multiplication primitive



Analog Matrix Multiplication

The key is using either technology to build a matrix multiplier with appropriate data transformation; an Eric Vittoz style thought experiment implies that if power efficiency is the top priority, ~ 1, 2 and 4 bit precision ops go in analog and ~ 8, 16 and 32 bit ops go in digital

- Digital scaling
  - Addition, comparisons, memory and data movement are linear in the number of bits
  - Multiplication is square in the number of bits
  - So digital scales somewhere in between linear and quadratic in the number of bits
- Analog scaling
  - For architectures where bits are in amplitude
  - Adding an extra bit at the same slicer separation requires doubling the power rail range
  - Doubling the power rail range leads to ~ 4x the power
  - Maybe for architecture with 2 levels that increase frequency the answer is more linear (should verify)
  - But frequency increase hits exponential wall of power and eventually need to go back to adding more levels



Bias Addition

- Data type
  - Same data type as filter coefficients for floating point
  - ~ 4x bits as filter coefficients for fixed point

- Operation
  - Bias matrix has rank 1 outer product structure
  - Can take advantage of this for all multiplication methods for loading the matrix with a vector and replication
  - Can also implement as part of matrix multiplication via the affine to linear conversion via matrix augmentation

Post Processing

- The most convenient candidates for post processing are memory localized and don't have significant memory dependencies across tiles

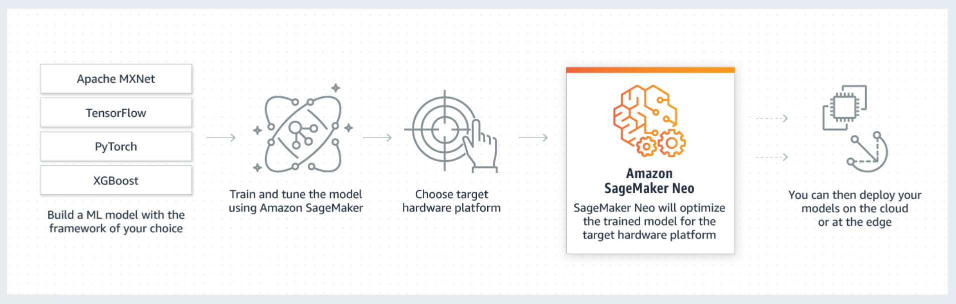
- Elementwise nonlinearities
  - ReLU (fully localized) is very cheap / simple; definite include option for this
  - Others are likely included depending on the connection to / performance of the small general processor

- Range tracking for fixed point
  - The compute scale is either determined statically or dynamically
  - To simplify dynamically setting the compute scale, the max / min range of the accumulators can be tracked

Backup – Example Software

Amazon SageMaker ML Workflow

- Build
  - Collect and prepare training data using SageMaker Ground Truth
    - https://aws.amazon.com/sagemaker/groundtruth/
  - Create ML models
    - Purchase ML models in Amazon Marketplace
    - Develop ML in MXNet, TensorFlow, PyTorch or XGBoost using SageMaker Notebooks (running on Amazon Compute instances)



- Train
  - Train models SageMaker Training
  - Optimize and compile for a target platform using SageMaker Neo compiler and runtime
    - https://aws.amazon.com/blogs/aws/amazon-sagemaker-neo-train-your-machine-learning-models-once-run-them-anywhere/
    - https://aws.amazon.com/sagemaker/neo/
    - https://github.com/neo-ai/neo-ai-dlr
    - https://neo-ai-dlr.readthedocs.io/en/latest/

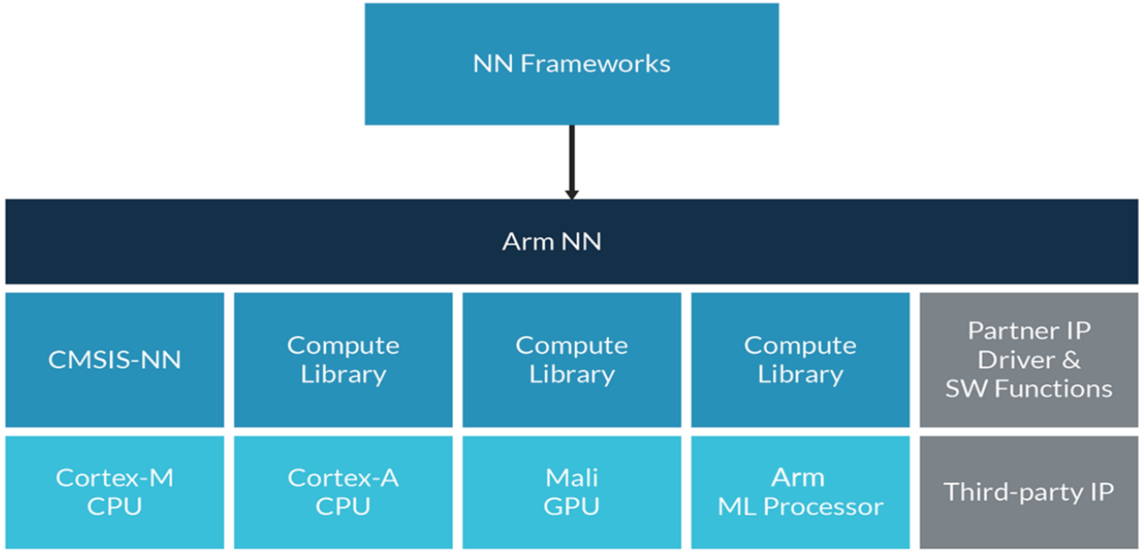
- Deploy
  - In the cloud with SageMaker Hosting or EC2 instances
  - On the the edge on Amazon IoT Greengrass devices
    - https://aws.amazon.com/greengrass/ml/

- Links
  - https://aws.amazon.com/sagemaker/
  - https://docs.aws.amazon.com/sagemaker/index.html
  - https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf

Figure from https://aws.amazon.com/sagemaker/neo/

ARM NN Graph Optimizer And Runtime

- ARM NN
  - Open source Linux software and tools to bridge between xNN frameworks and ARM cores
  - Uses the compute library to target A cores, GPUs and the ML processor
  - Does not currently provide support for M cores (uses CMSIS-NN instead)
  - Translates xNN frameworks to an internal representation and distributes to present cores (effectively a graph optimizer and runtime)
  - Open source and donated to the Linaro Machine Intelligence Initiative
- Supported xNN frameworks
  - TensorFlow, TensorFlow Lite, PyTorch, ONNX, MXNet, Caffe, Caffe2, Android NN API
  - https://developer.arm.com/solutions/machine-learning-on-arm/developer-material/how-to-guides/configuring-the-arm-nn-sdk-build-environment-for-tensorflow
  - https://developer.arm.com/solutions/machine-learning-on-arm/developer-material/how-to-guides/configuring-the-arm-nn-sdk-build-environment-for-tensorflow-lite
  - https://developer.arm.com/solutions/machine-learning-on-arm/developer-material/how-to-guides/configuring-the-arm-nn-sdk-build-environment-for-onnx
  - https://developer.arm.com/solutions/machine-learning-on-arm/developer-material/how-to-guides/configuring-the-arm-nn-sdk-build-environment-for-caffe



- Links
  - https://mlplatform.org
  - https://github.com/ARM-software/armnn
  - https://developer.arm.com/ip-products/processors/machine-learning/arm-nn
  - https://developer.arm.com/ip-products/processors/machine-learning/compute-library

Figure from https://mlplatform.org

ARM M Core Library

- ARM CMSIS-NN
  - Neural network software library for ARM M cores
  - 8 and 16 bit integer implementations
- Functions
  - Neural network convolution
  - Neural network activation
  - Fully-connected layer
  - Neural network pooling
  - Softmax
  - Neural network support
- Links
  - https://arm-software.github.io/CMSIS_5/NN/html/index.html
  - https://arxiv.org/abs/1801.06601

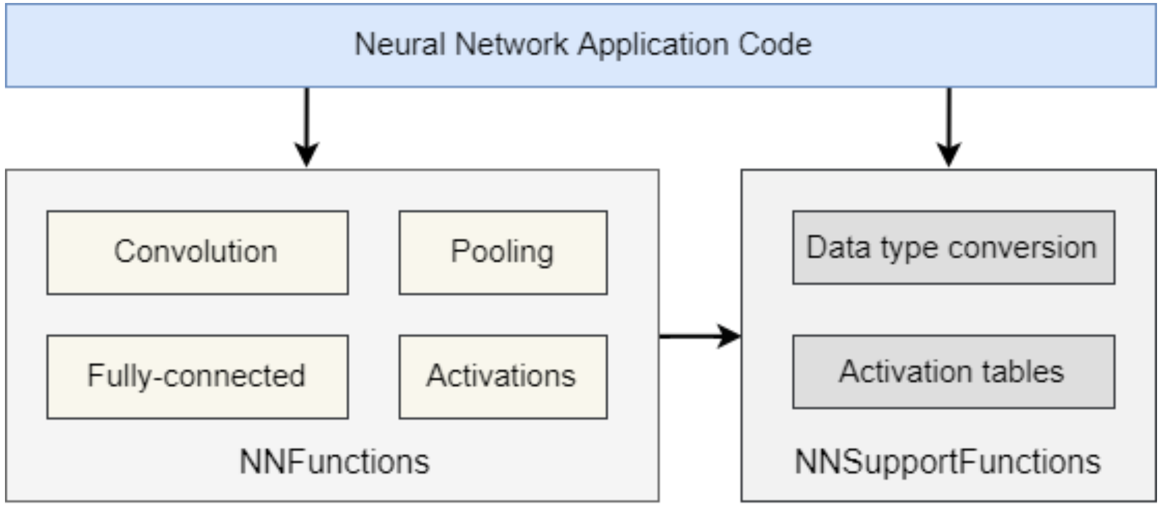


Figure from https://arm-software.github.io/CMSIS_5/NN/html/index.html

ARM A Core, Mali GPU And ML Proc Library

- Compute library
  - Basic arithmetic, mathematical, and binary operator functions
  - Color manipulation
  - Convolution filters
  - Canny Edge, Harris corners, optical flow
  - Pyramids
  - HOG
  - SVM
  - H/SGEMM
  - Convolutional Neural Networks building blocks
- Adding a new backend
  - https://github.com/ARM-software/armnn/tree/branches/armnn_19_02/src/backends

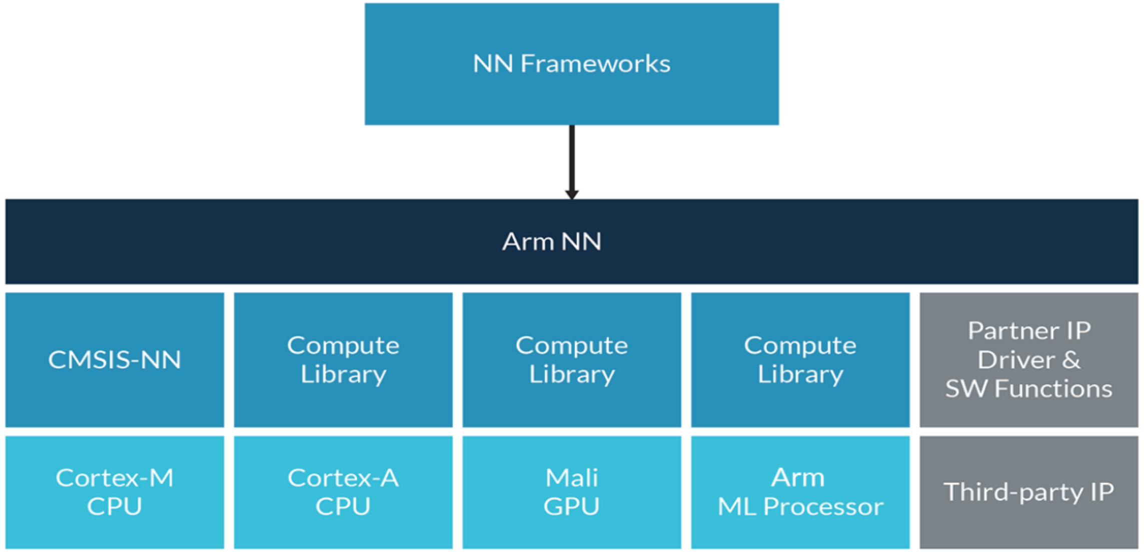
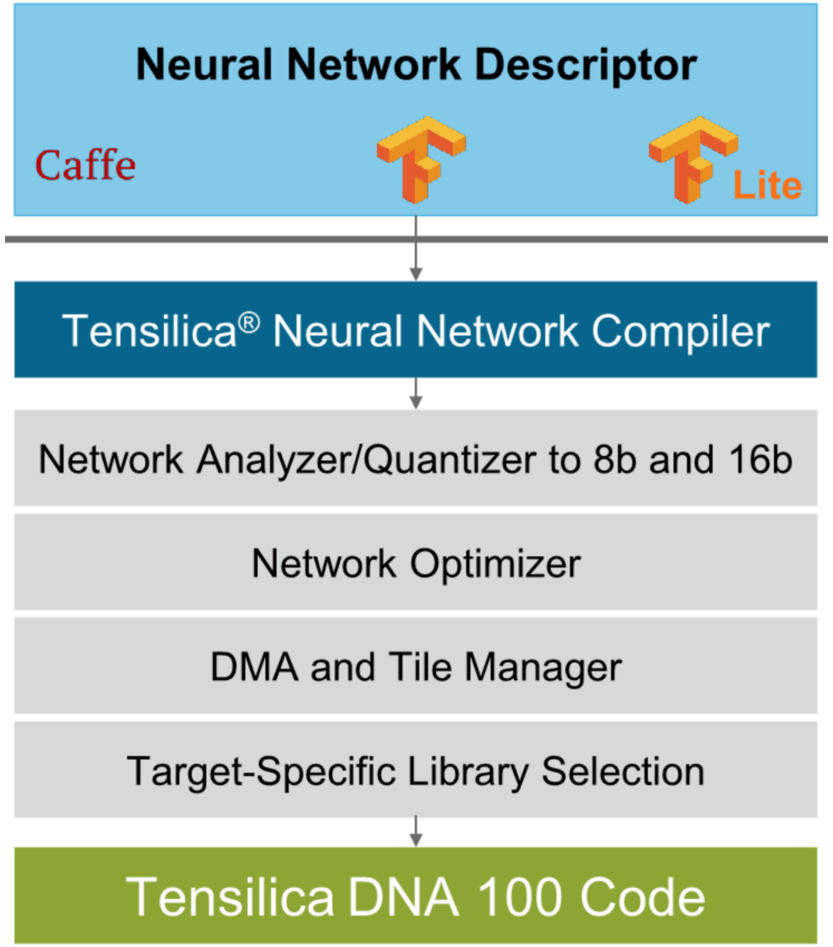


Figure from https://mlplatform.org

Cadence Tensilica DNA 100



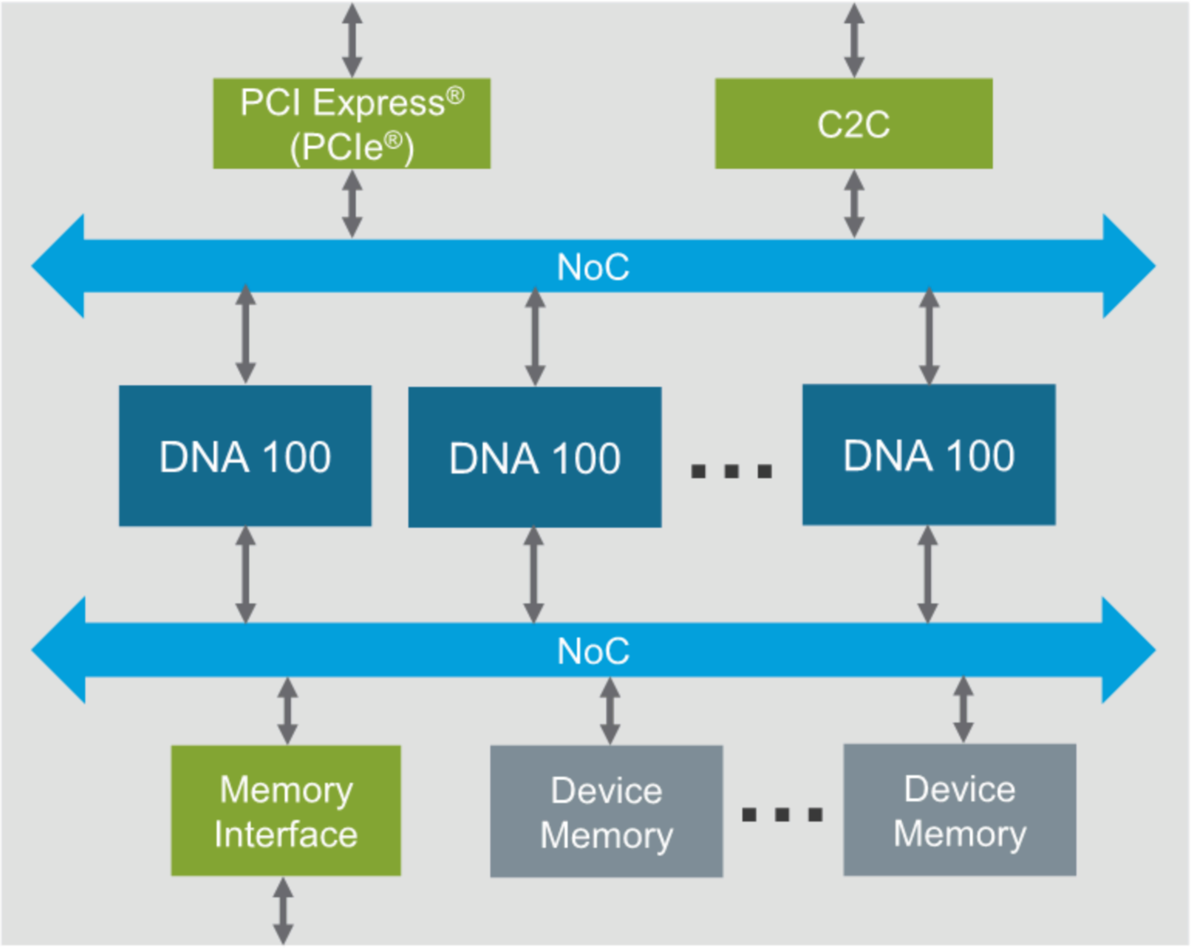


Figure from https://www.anandtech.com/show/13377/cadence-announces-tensilica-dna-100-a-bigger-nn-ip

CEVA CDNN And NeuPro

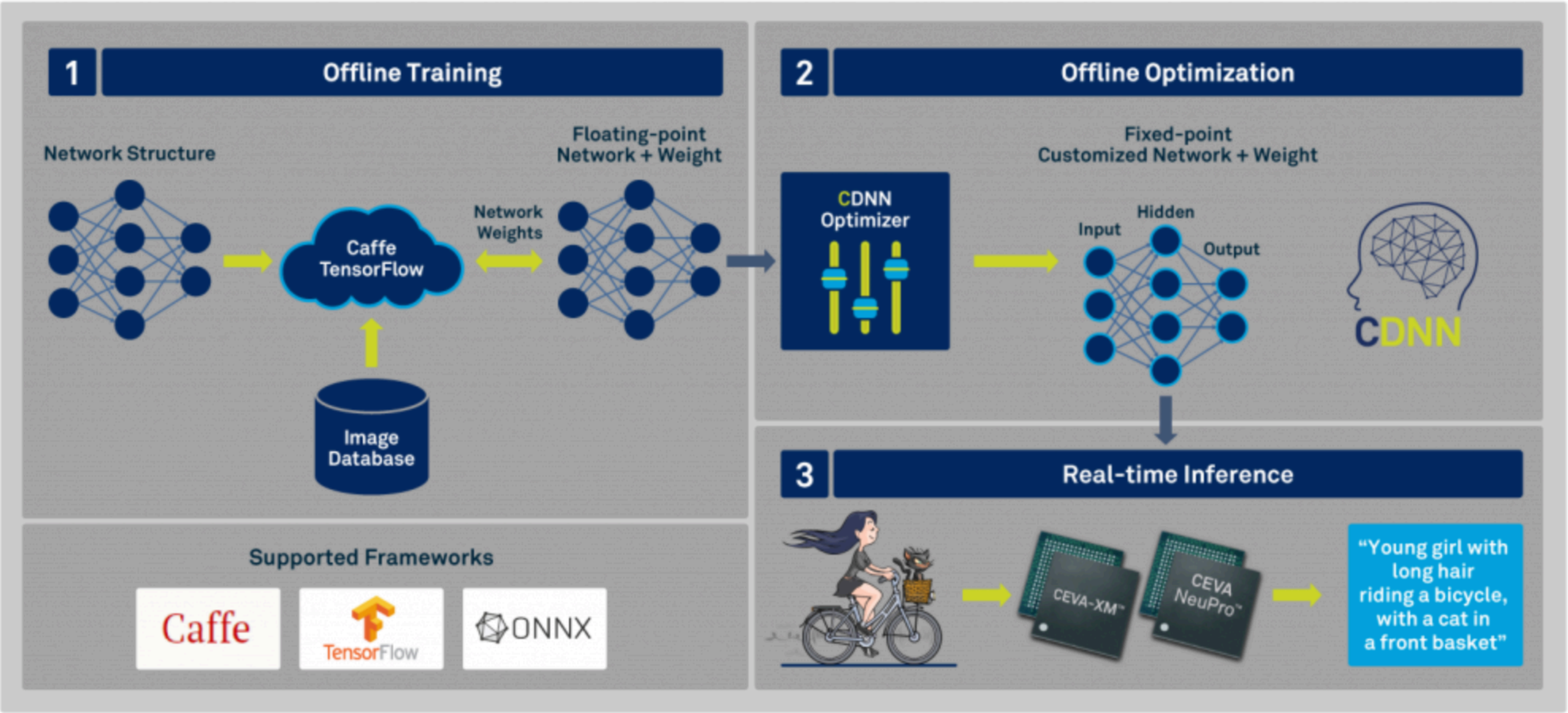


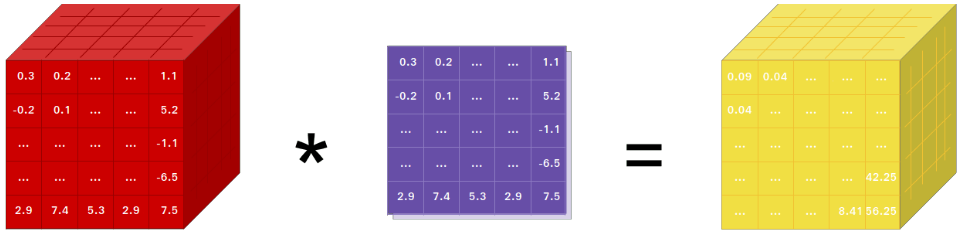
Figure from https://www.ceva-dsp.com/product/ceva-deep-neural-network-cdnn/

Facebook PyTorch

- PyTorch is Facebook's open source library for creating and training machine learning models (https://pytorch.org)

- Key features
  - Optimized tensor library for deep learning using CPUs and GPUs
  - Tape based autograd system
  - Front end support for eager and graph mode

- Components
  - Torch: NumPy like tensor library with GPU acceleration
  - Torch.nn: neural network library
  - Torch.hub: pre trained model repository
  - Torch.distributions: probability library
  - Torch.optim: optimizer library
  - Torch.autograd: tape based auto grad library
  - Torch.jit: compiler to create models from PyTorch code
  - Torch.utils: data loaders and other common utilities
  - Torch.Tensor: tensor type definition
  - Torch.distributed: distributed communication
  - Torch.multiprocessing: Python multiprocessing
  - Torch.onnx: ONNX exporter



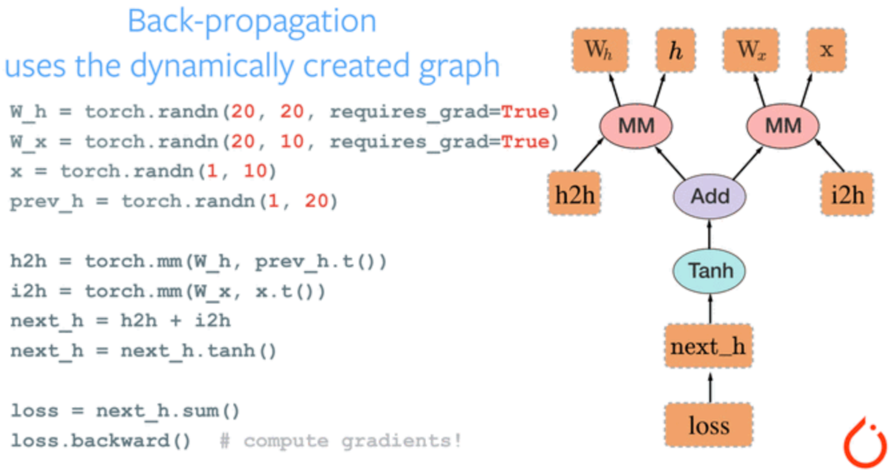
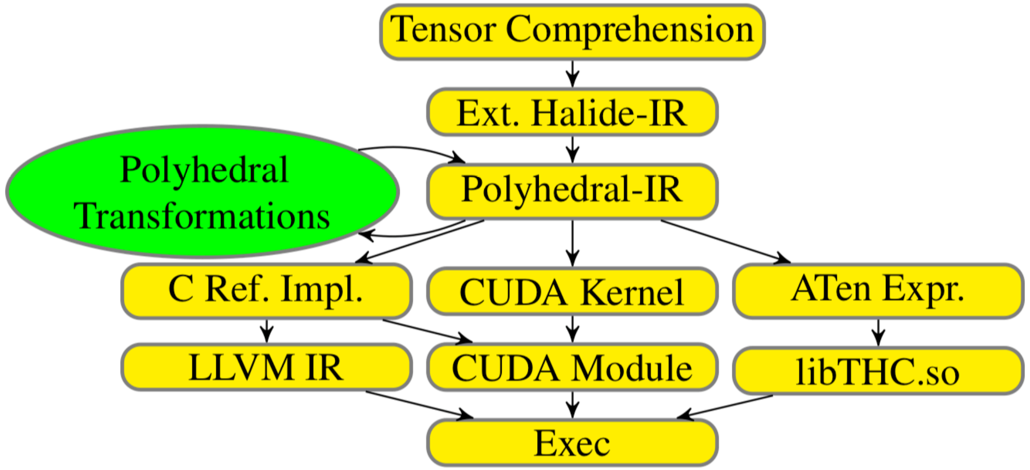


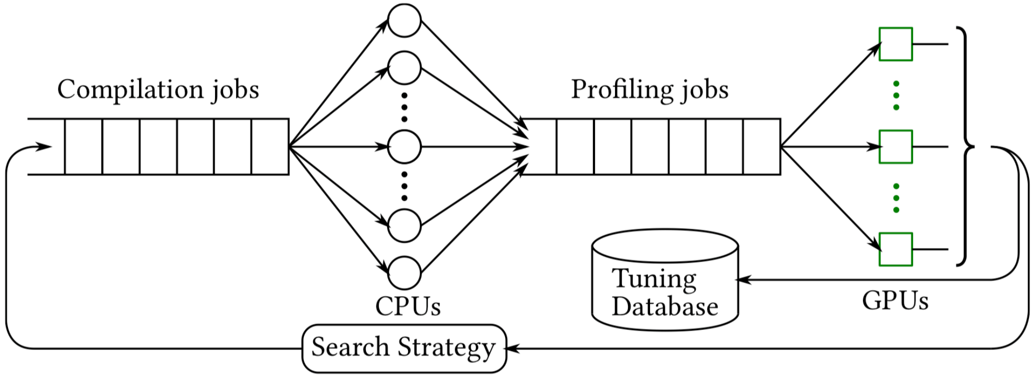
Figure from https://github.com/pytorch/pytorch

Facebook PyTorch Custom Layer Optimizer

- Tensor Comprehensions
  - A notation for concisely writing layers
  - Integrates with other frameworks
  - 1 function == 1 kernel
  - No allocation of memory
  - Focuses on the loop structure implied by tensors



- Links
  - Tensor comprehensions
    - https://github.com/facebookresearch/TensorComprehensions
  - Tensor comprehensions documentation
    - https://facebookresearch.github.io/TensorComprehensions/index.html
  - Tensor comprehensions: framework-agnostic high-performance machine learning abstractions
    - https://arxiv.org/abs/1802.04730



Figures from https://arxiv.org/abs/1802.04730

Facebook PyTorch Model Format

- Open neural network exchange format (ONNX)
    - An open source format for xNN models that provides a common IR that's runtime agnostic (so it needs a runtime)
    - Allows models to be trained in 1 framework and transferred to another for inference
    - Includes deep learning (standard) and classical machine learning (-ml) variants
    - See https://onnx.ai and https://github.com/onnx

- Includes
    - Definitions of an extensible computational graph model
        - https://github.com/onnx/onnx/blob/master/onnx/onnx.proto3
        - https://github.com/onnx/onnx/blob/master/onnx/onnx-ml.proto3
    - Definitions of standard data types
    - Definitions of built in operators
        - https://github.com/onnx/onnx/blob/master/docs/Operators.md
        - https://github.com/onnx/onnx/blob/master/docs/Operators-ml.md

- Tools
    - Frameworks: PyTorch, Caffe2, Cognitive Toolkit, MXNet, Chainer, PaddlePaddle, Matlab, SAS, Neural Network Libraries
    - Converters: TensorFlow, Keras, CoreML, sciit-learn, XGBoost, LIBSVM, ncnn
    - Visualizers: Netron, VisualDL
    - Compilers: Intel AI, Skymizer, TVM
    - Runtimes: Nvidia, Qualcomm, BitMain, Tencent, Vespa, Windows, Synopsys, **ONNX Runtime**, Ceva, MACE, Habana

Facebook PyTorch Runtime (From MSFT)

- ONNX Runtime
  - An inference engine for ONNX models created by Microsoft
  - Currently supports CUDA, TensorRT, MLAS, MKL-DNN, MKL-ML and nGraph execution providers (execution provider == custom accelerator / runtime abstraction)
  - https://github.com/microsoft/onnxruntime
  - https://github.com/microsoft/onnxruntime/blob/master/docs/HighLevelDesign.md
- Flow
  - Convert ONNX model to in memory graph representation
  - Perform execution provider independent optimizations
  - Partition the graph into sub graphs based on the available execution providers
  - Assign sub graphs to execution providers
- Extensibility
  - Adding custom operators / kernels
  - Adding execution providers
  - Adding graph transformations

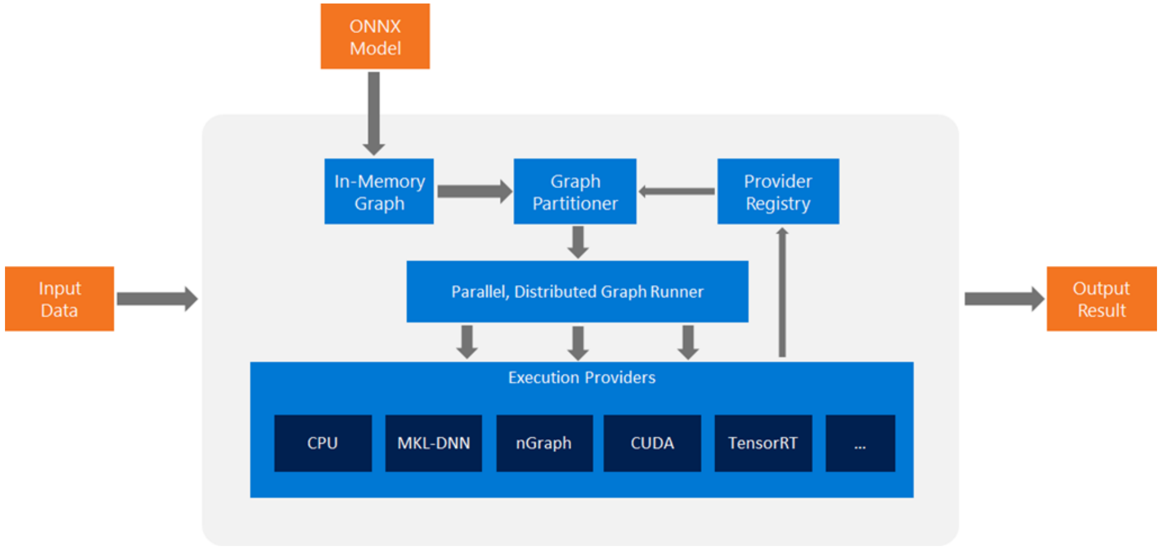


Figure from https://github.com/microsoft/onnxruntime/blob/master/docs/HighLevelDesign.md

Facebook PyTorch Compiler And Runtime

- Glow is a machine learning compiler and runtime for hardware accelerators
  - Not all input operators need to be supported on all hardware back ends
- High level graph IR
  - Strongly types node based graph representation
  - Domain specific target independent optimizations
- Low level graph IR
  - Instruction based address only intermediate representation allows copy elimination, static memory allocation and instruction scheduling
- Machine code
  - Hardware specific code generation

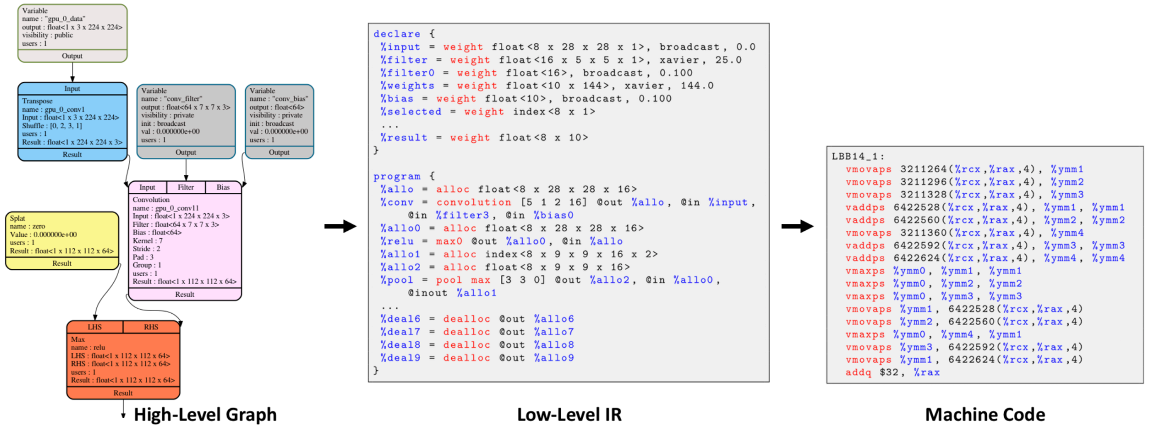


Figure from https://github.com/pytorch/glow

Facebook PyTorch Compiler And Runtime

- Compiler flow (from https://arxiv.org/abs/1805.00907)
  1. The graph is either loaded via the graph loader (from ONNX or Caffe2 format) or constructed via the C++ interface
  2. The graph is differentiated if needed
  3. The graph is optimized
  4. Linear algebra node lowering takes place
  5. Additional rounds of optimizations occur, both target independent and target specific
  6. The graph is scheduled into a linear sequence of nodes that minimizes memory usage
  7. IRGen converts the low level graph into instructions
  8. Low level IR optimizations are performed
  9. Backend specific optimizations and code generation are performed

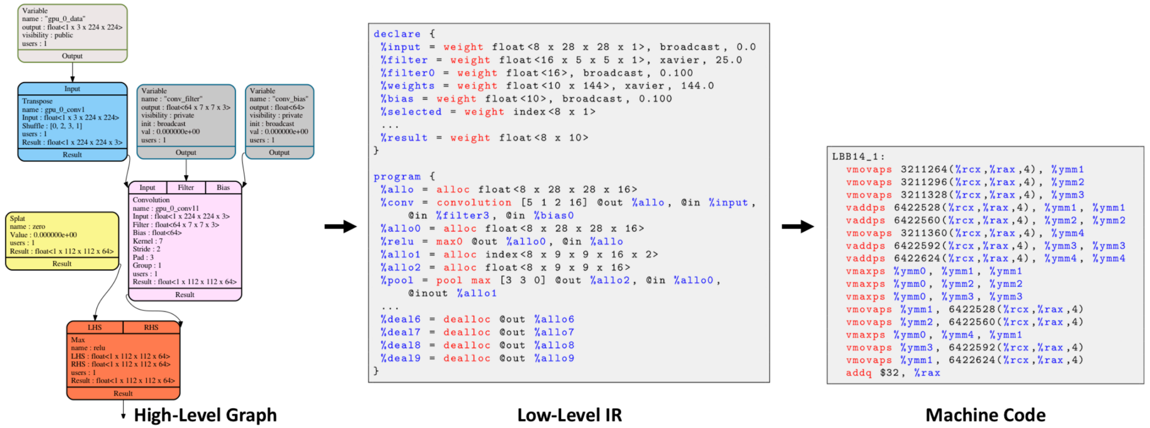


Figure from https://github.com/pytorch/glow

Facebook PyTorch Compiler And Runtime

- Runtime flow for adding a network (from https://arxiv.org/abs/1805.00907)
  1. The Partitioner splits the network into one or more sub networks
  2. The Provisioner compiles each sub network and assigns them to one or more devices
  3. One or more DeviceManagers load the sub networks and their weights onto its associated device

- Runtime flow for handling an inference request (from https://arxiv.org/abs/1805.00907)
  1. The HostManager creates a new execution graph with intermediate storage
  2. The Executor kicks off the first sub network execution
  3. The DeviceManager loads inputs onto the card and begins execution; when done it reads outputs and signals completion
  4. The Executor triggers any sub networks with satisfied dependencies
  5. When complete the HostManager returns outputs

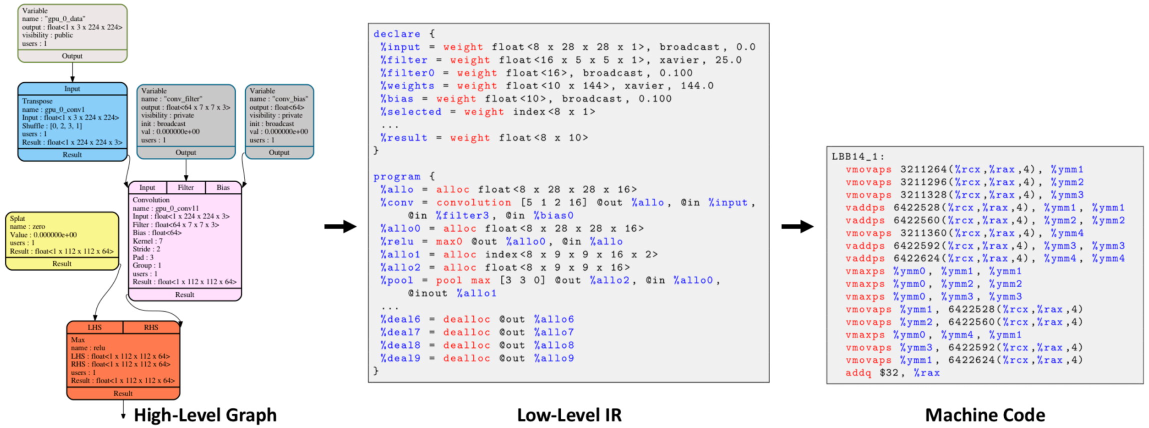
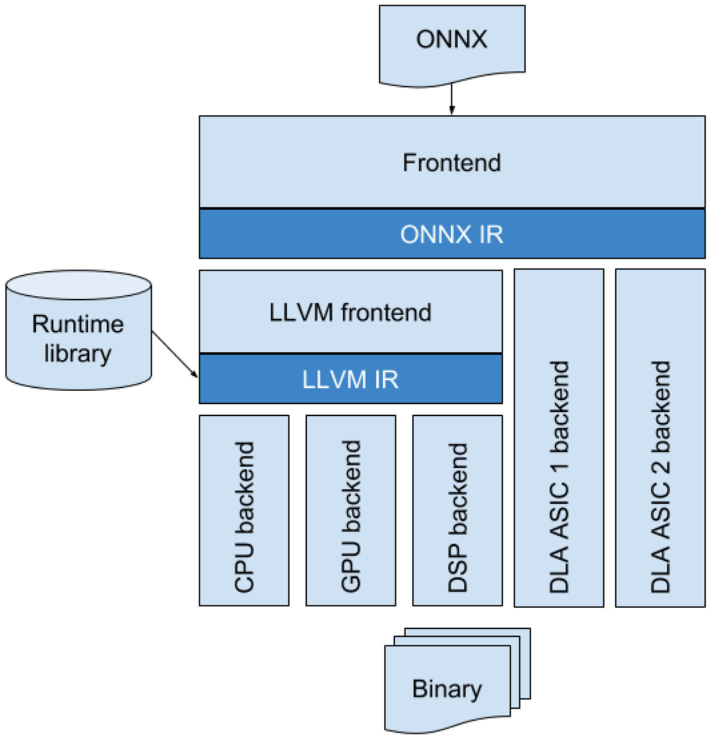
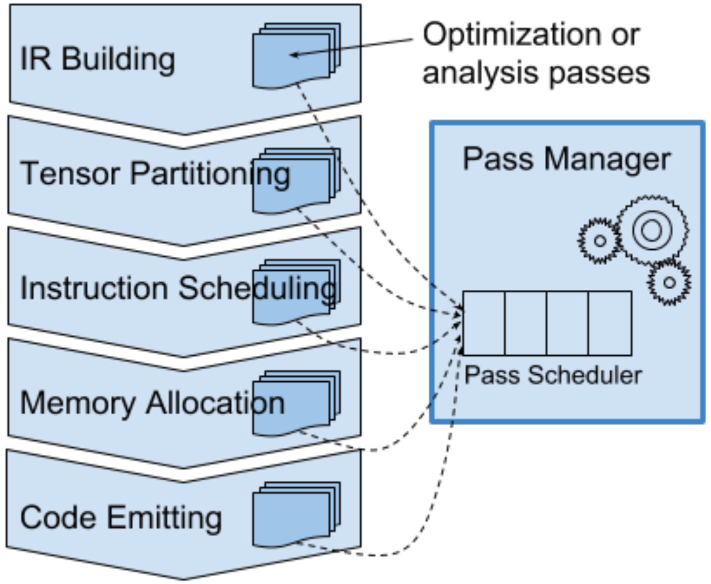


Figure from https://github.com/pytorch/glow

Facebook PyTorch Compiler And Runtime

- From ONNC (not FB)
- ONNX API and IR with LLVM and generic interfaces
  - Allows targeting of CPUs, DSPs and GPUs via LLVM code
  - Allows targeting of DSAs via generic interface
- For more info
  - https://onnc.ai
  - https://github.com/ONNC/onnc
  - https://www.youtube.com/watch?time_continue=1&v=-FuKZFfWIXo





Google TensorFlow

- TensorFlow is Google's open source library for creating and training machine learning models
  - 2.x: https://www.tensorflow.org

- Versions
  - 1.x: default is declarative execution
    - Specify graph, execute graph
  - 2.x: default is eager execution
    - Imperative environment, operations execute immediately
    - The standard method for using the Keras API in 2.x is ~ declarative like
    - And the @tf.function decorator can be used to create declarative code
    - Declarative code (specify graph, execute graph) is good for performance

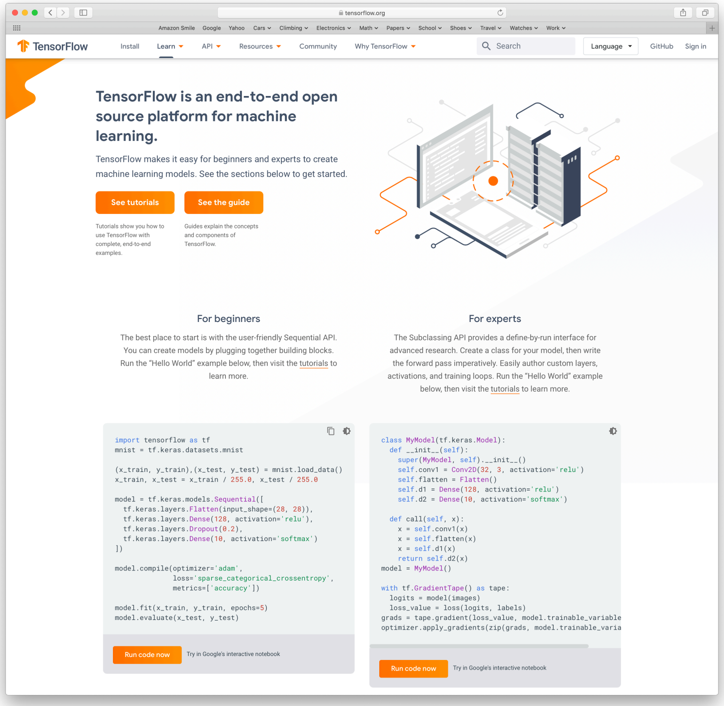


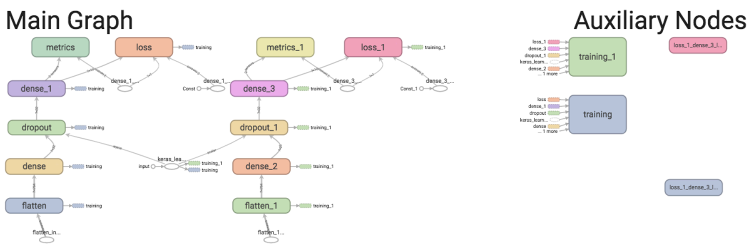
Figure from https://www.tensorflow.org/overview

Google TensorFlow

Examples below indicate the tf.data or high level Keras API

- High level graph specification
  - Data (tf.data)
  - Network (tf.keras.Model)
  - Loss (tf.keras.Model.compile)
  - Gradient back prop (tf.keras.Model.compile)
  - Weight update (tf.keras.Model.compile)

- High level graph execution
  - Training (tf.keras.Model.fit)
  - Evaluation (tf.keras.Model.evaluate)
  - Prediction (tf.keras.Model.predict)



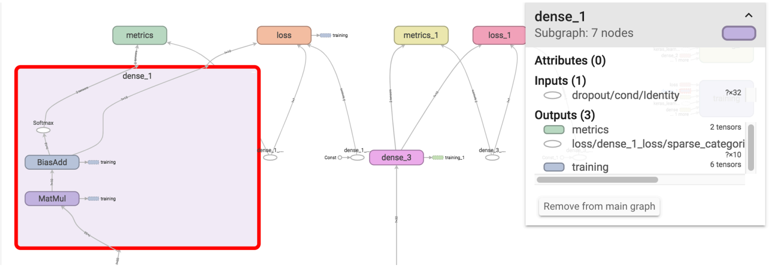
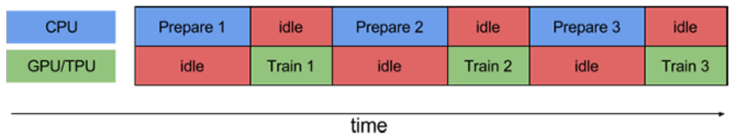


Figure from https://www.tensorflow.org/tensorboard/r2/graphs

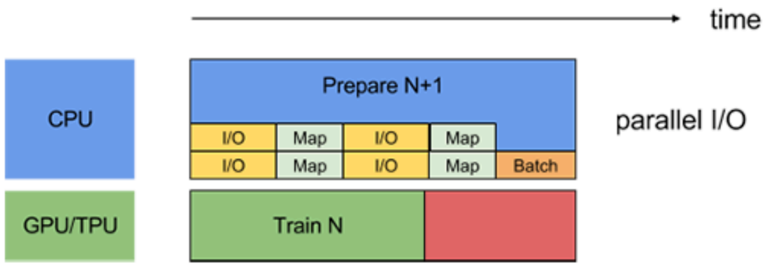
Google TensorFlow Graph Specification

Data

- The tf.data API is used for building data pipelines
  - Inputs and true outputs
  - Training, validation and prediction
  - The data pipeline is part of the graph



- The tf.data.Dataset abstraction is used to represent a sequence of elements where each element consists of 1 or more tensors
  - Basic strategy is to create a datasets from data in memory or files
  - Then transform the dataset via map, shuffle, batch and repeat operations
  - To improve performance consider pre fetching to overlap data I/O for the next batch on the CPU with processing the current batch on the GPU, parallel data mapping and parallel data I/O



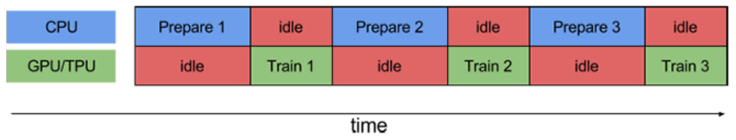
- References
  - https://www.tensorflow.org/guide/data
  - https://www.tensorflow.org/guide/data_performance

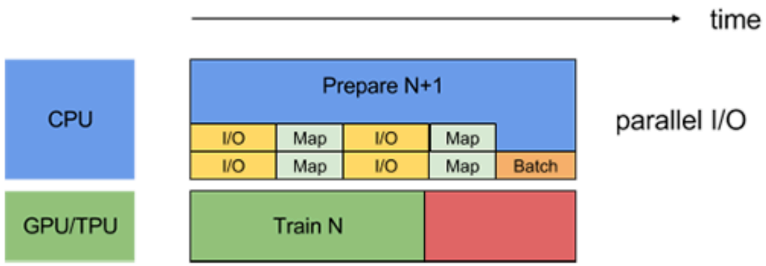
Figures from https://www.tensorflow.org/beta/guide/data_performance

Google TensorFlow Graph Specification

Data

- For examples of specific types of data
  - Load CSV with tf.data
    - https://www.tensorflow.org/tutorials/load_data/csv
  - Load NumPy Data with tf.data
    - https://www.tensorflow.org/tutorials/load_data/numpy
  - Load images with tf.data
    - https://www.tensorflow.org/tutorials/load_data/images
  - Load text with tf.data
    - https://www.tensorflow.org/tutorials/load_data/text
  - Using TFRecords and tf.Example
    - https://www.tensorflow.org/tutorials/load_data/tf_records
  - Unicode strings
    - https://www.tensorflow.org/tutorials/text/unicode
  - TF.Text
    - https://www.tensorflow.org/tutorials/tensorflow_text/intro





Google TensorFlow Graph Specification

Data

- The Keras API simplifies downloading and loading data from public research datasets into NumPy arrays

```
# import
from keras.datasets import mnist

# download and load MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- References
  - https://keras.io/datasets/

Google TensorFlow Graph Specification

Data

- TensorFlow Datasets API simplifies downloading, loading and creating tf.data.Datasets from public research datasets
- References
  - https://medium.com/tensorflow/introducing-tensorflow-datasets-c7f01f7e19f3
  - https://www.tensorflow.org/datasets
  - https://www.tensorflow.org/datasets/datasets
  - https://www.tensorflow.org/datasets/overview
  - https://www.tensorflow.org/datasets/api_docs/python/tfds

```
# see all registered datasets
tfds.list_builders()

# load a given dataset by name, along with the dataset info
data, info = tfds.load("mnist", with_info=True)
train_data, test_data = data['train'], data['test']
assert isinstance(train_data, tf.data.Dataset)
assert info.features['label'].num_classes == 10
assert info.splits['train'].num_examples == 60000

# you can also access a builder directly
builder = tfds.builder("mnist")
assert builder.info.splits['train'].num_examples == 60000
builder.download_and_prepare()
datasets = builder.as_dataset()

# if you need NumPy arrays
np_datasets = tfds.as_numpy(datasets)
```

Code from https://www.tensorflow.org/datasets/datasets

Google TensorFlow Graph Specification

Network

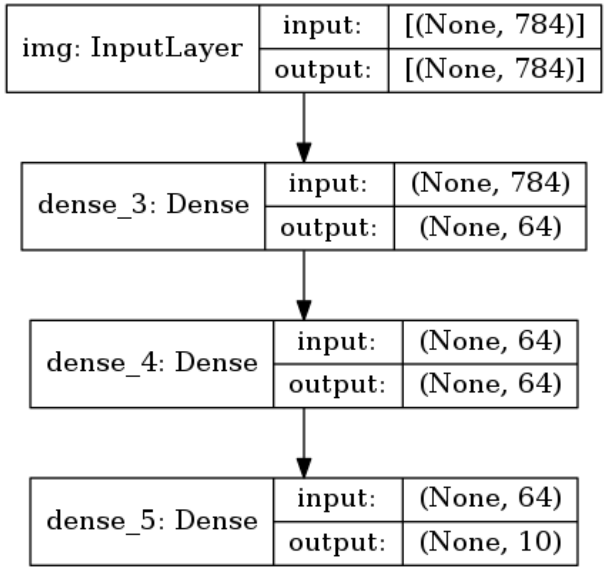
- Networks
  - Are specified as graphs
    - Edges are memory or dependencies
    - Nodes are operators (layers)
  - Map data inputs to outputs
    - Encoder output = features (typically)
    - Encoder + decoder output = predictions (typically)

- High and low level APIs are available for network specification
  - **High level**
    - TensorFlow 2.x has standardized on Keras
    - There's a general trend towards this providing an appropriate level of control and being appropriate for most work
    - https://keras.io
    - https://www.tensorflow.org/guide/keras/overview
  - Low level

Google TensorFlow Graph Specification

Network

- Keras provides multiple API options for specifying network models
  - Overview
    - https://keras.io/models/about-keras-models/
  - Sequential API
    - Ok for getting started with simple examples, but not flexible with respect to topology
    - https://keras.io/getting-started/sequential-model-guide/
    - https://keras.io/models/sequential/
  - **Functional API**
    - A nice level of abstraction for most networks
    - https://keras.io/getting-started/functional-api-guide/
    - https://keras.io/models/model/
    - https://www.tensorflow.org/guide/keras/functional
  - Model sub classing
    - Create layers in __init__ and define the forward pass in call; this allows the forward pass to be run imperatively, but it is preferred to use the functional API when possible
    - https://keras.io/models/about-keras-models/#model-subclassing
    - https://www.tensorflow.org/guide/keras/overview#model_subclassing
    - https://www.tensorflow.org/guide/keras/custom_layers_and_models#building_models



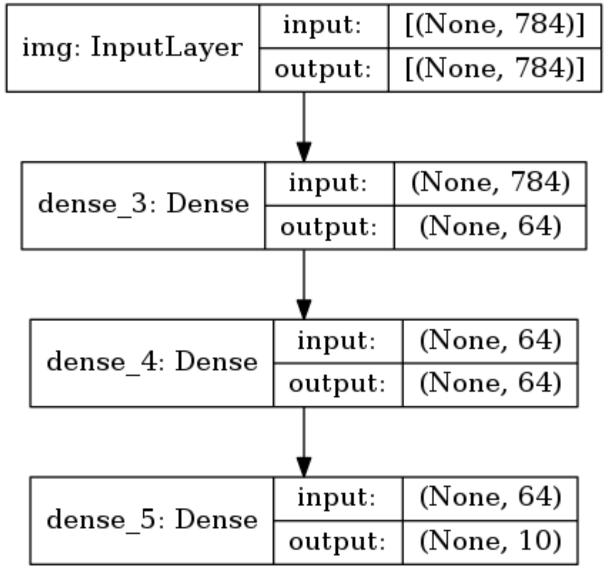
- model.summary() creates a text description of the model including layers, output shapes and numbers of parameters
- keras.utils.plot_model(model, 'file_name.png') creates a figure showing connections
- keras.utils.plot_model(model, 'file_name.png', show_shapes=True) creates a figure showing connections and input / output feature map sizes

Figure from https://www.tensorflow.org/guide/keras/functional

Google TensorFlow Graph Specification

Network

- Things to consider when specifying models
  - Having separate access to the encoder (and possibly multiple points in the encoder) and decoder outputs to simplify transfer learning
  - Enabling arbitrary numbers of repeats for building blocks to simplify the creation of different accuracy vs performance models
- Models are built from layers
  - Built in layers
    - https://keras.io/layers/about-keras-layers/
  - Custom layers
    - https://keras.io/layers/writing-your-own-keras-layers/
    - https://www.tensorflow.org/guide/keras/custom_layers_and_models#the_layer_class
- Operator placement
  - https://www.tensorflow.org/guide/using_gpu
  - Default operator placement uses GPUs when implementations are possible, falls back to CPU when not



- model.summary() creates a text description of the model including layers, output shapes and numbers of parameters
- keras.utils.plot_model(model, 'file_name.png') creates a figure showing connections
- keras.utils.plot_model(model, 'file_name.png', show_shapes=True) creates a figure showing connections and input / output feature map sizes

Figure from https://www.tensorflow.org/guide/keras/functional

Google TensorFlow Graph Specification

Loss, gradient back prop and weight update

- There are different options available for training
  - **Built in training loop**
    - Keras losses https://keras.io/losses/ , weight update https://keras.io/optimizers/ and metrics https://keras.io/metrics/
    - https://www.tensorflow.org/guide/keras/training_and_evaluation#part_i_using_build-in_training_evaluation_loops
    - 1: compile (define loss, implicitly define back prop, specify weight update)
    - 2: fit (train)
    - 3: evaluate and predict
  - Custom training loop
    - https://www.tensorflow.org/guide/keras/training_and_evaluation#part_ii_writing_your_own_training_evaluation_loops_from_scratch
    - 1: define loss, define weight update, explicitly use gradient tape to record forward pass to implicitly define back prop
    - 2: explicitly retrieve gradients, use gradients with weight update
    - 3: explicitly evaluate and predict
- Why mention the above training items during graph specification?
  - Because the loss, back prop and weight update can be thought of as part of the graph needed for training
  - And when using the high level API they're specified in the compile function

Google TensorFlow Graph Specification

Loss, gradient back prop and weight update

- Loss and optimizer (tf.keras.Model.compile)
    - Adds a loss
    - Backwards path is automatically created from forwards path
    - Adds an optimizer to the model
    - Specifies metrics to track

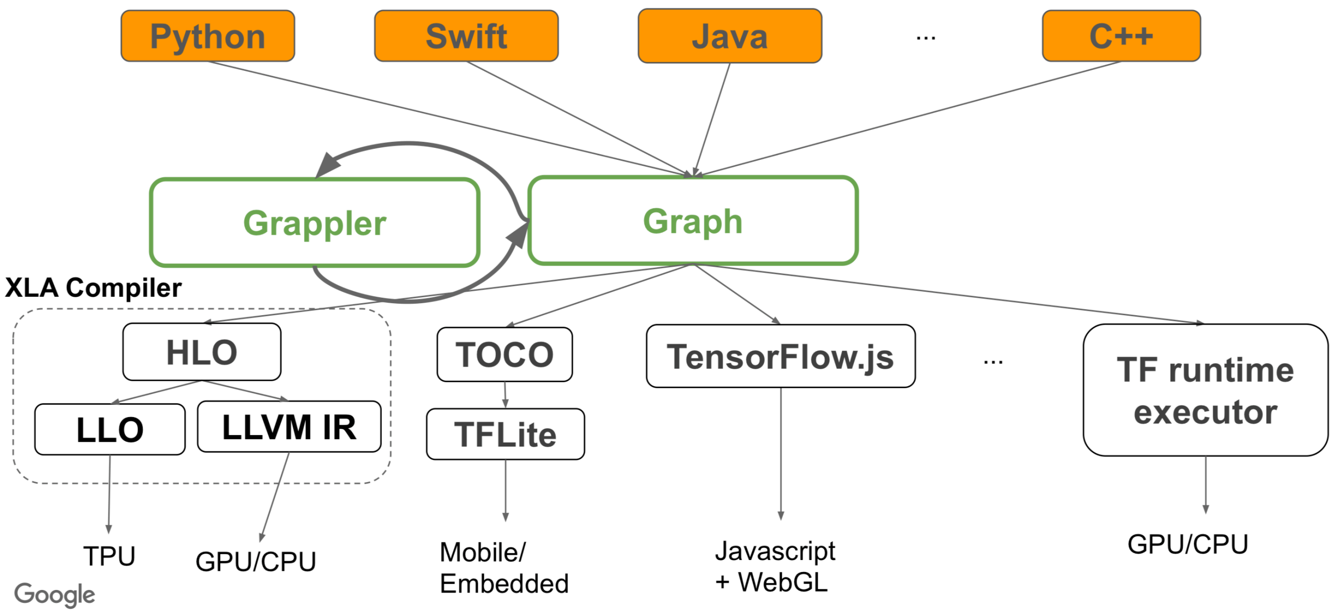
Google TensorFlow Graph Specification

Pre trained models

- TensorFlow Hub provides a library of pre trained graphs / graph fragments referred to as modules that can be used stand alone or as part of other graphs
  - Ex: using a pre trained image to feature encoder used as the encoder for Faster R-CNN
- References
  - https://www.tensorflow.org/hub
  - https://tfhub.dev

Google TensorFlow Graph Optimization

- Grappler is the default high level graph optimizing system in the TensorFlow runtime
  - Re writes TensorFlow high level graphs to improve out of the box TensorFlow performance
  - Includes plug in infrastructure to register custom optimizers and high level graph re writers
- TensorFlow graph optimizations
  - http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf



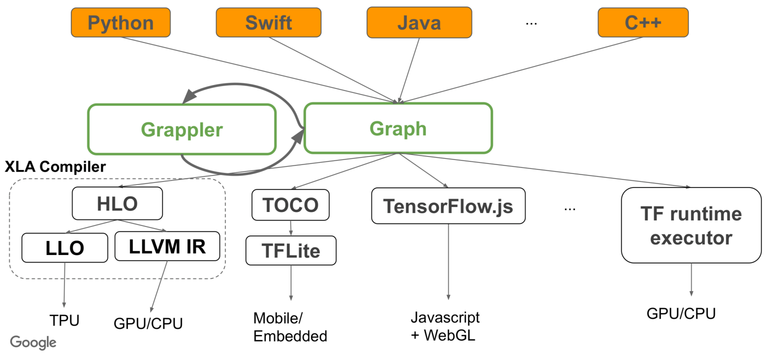
- You don't explicitly call Grappler for graph optimization
- It's called for you automatically during the compile process
- But it's worthwhile to know that it's there, the functionality that it provides and how it can be extended / modified

Figure from http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Google TensorFlow Graph Optimization

- Optimizer loop (from below reference)

```
i= 0
while i < num iterations (default=2):
   Pruning()          # remove nodes not in fanin of outputs, unused functions
   Function()         # function specialization and inlining, symbolic gradient inlining
   DebugStripper()    # remove assert, print, check_numerics
   ConstFold()        # constant folding and materialization
   Shape()            # symbolic shape arithmetic
   Remapper()         # op fusion
   Arithmetic()       # node deduping (CSE) and arithmetic simplification
   if i==0: Layout()  # layout optimization for GPU
   if i==0: Memory()  # swap out / swap in, recompute, split large nodes
   Loop()             # loop invariant node motion, stack push and dead node elimination
   Dependency()       # prune / optimize control edges, noop / identity node pruning
   Custom()           # run registered custom optimizers (e.g. TensorRT)
   i += 1
```

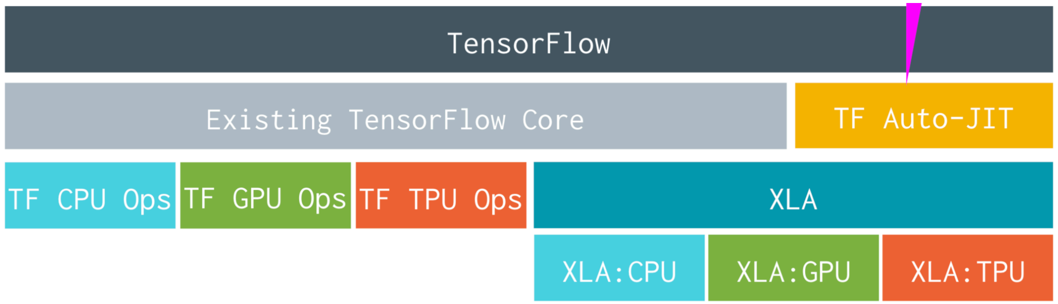


- TensorFlow graph optimizations
  - http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Figure from http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Google Sub Graph Optimization

- XLA is a domain specific compiler for TensorFlow computation analysis, optimization and code generation
  - Example use: fuse together a number of primitive operations (a sub graph) into a single larger operation (node) optimized for a specific target and generate associated code



- Overview
  - TensorFlow w/XLA: TensorFlow, compiled!
  - https://autodiff-workshop.github.io/slides/JeffDean.pdf

Figure from https://autodiff-workshop.github.io/slides/JeffDean.pdf

Google Sub Graph Optimization

- Improve execution speed
  - Compile sub graphs to eliminate the overhead of the TensorFlow runtime
  - Fuse pipelined operations to reduce memory overhead
  - When an individual op is large all is well (assuming that there's not a large on device memory for linking big ops) but when individual ops are small this helps
  - Specialize to known tensor shapes to allow for more aggressive constant propagation
- Improve memory usage
  - Analyze and schedule memory usage
  - Eliminate intermediate storage buffers
- Reduce reliance on custom ops
  - Fuse low level ops to new high level ops while maintaining performance
- Reduce mobile footprint
  - Remove TensorFlow runtime by ahead of time compiling the sub graph to an object / header file that can then be linked into another program
- Improve portability
  - Make it easy to write a new backend for novel hardware

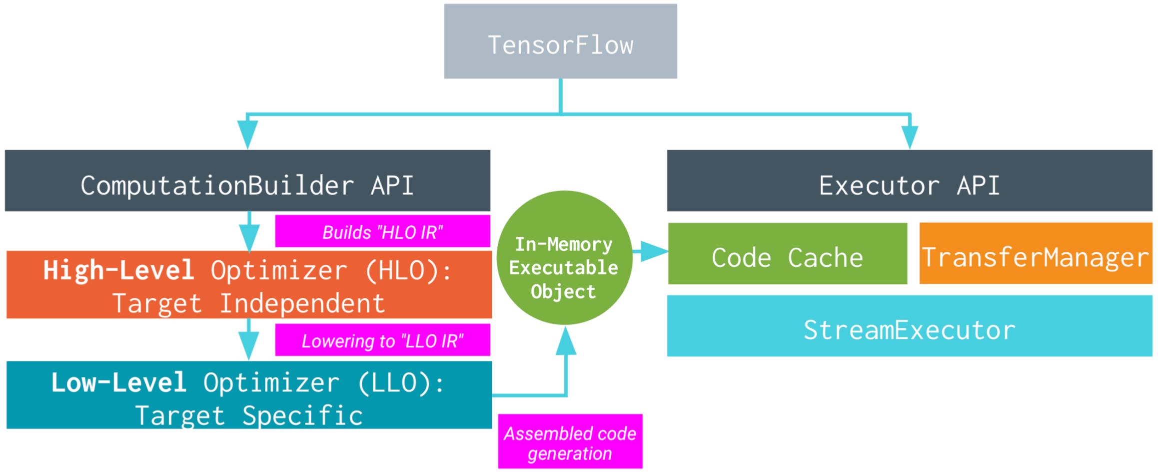


Figure from https://autodiff-workshop.github.io/slides/JeffDean.pdf

Google Sub Graph Optimization

- The input to XLA is high level optimizer (HLO) IR
  - These are graphs with graph operations are defined in operation semantics (https://www.tensorflow.org/xla/operation_semantics)
  - Think of these as all of the supported operations (there are a lot)
  - A while loop is also included
- Target independent front end does analysis and optimization
  - Common sub expression elimination
  - Target independent operation fusion
  - Buffer analysis of allocating runtime memory (though real optimization of this should be hardware specific)
  - The output of this is still HLO IR
- Target dependent back end does analysis and optimization and code generation for CPUs, GPUs and TPUs
  - Hardware target specific optimization
  - The output is LLVM for the CPU and GPU

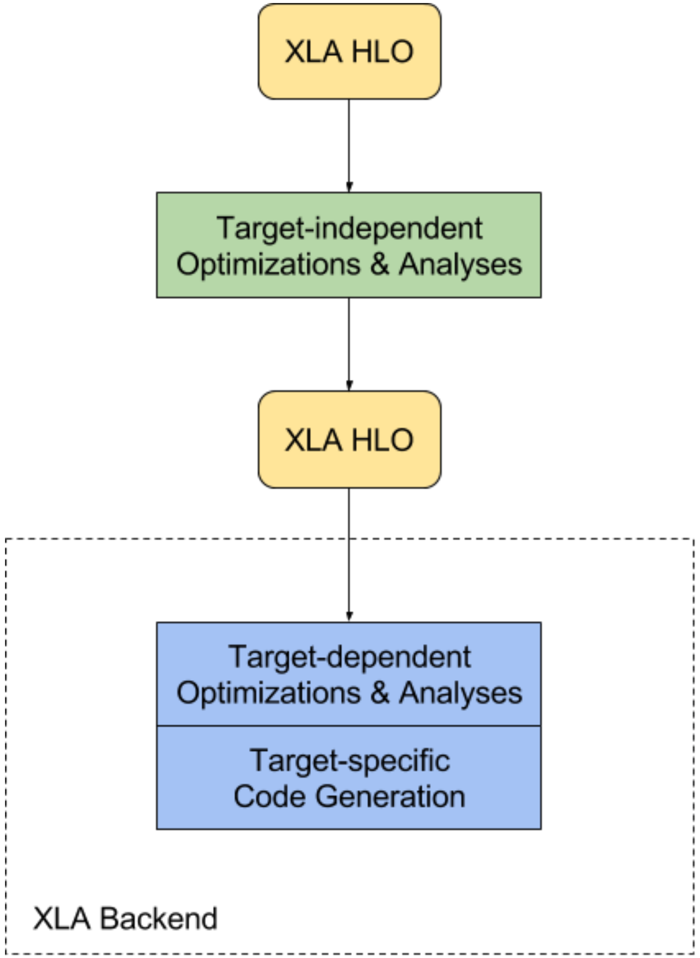


Figure from https://www.tensorflow.org/xla/overview

Google Sub Graph Optimization

- Supported operations and operand broadcasting semantics
  - https://www.tensorflow.org/xla/operation_semantics
  - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/compiler/xla/client/xla_builder.h
  - https://www.tensorflow.org/xla/broadcasting
- Operand shape and memory layout specification
  - Shapes
    - https://www.tensorflow.org/xla/shapes
    - Allows specification of row or col major order (and generalizations)
    - Allows specification of 0 padding dimensions to a larger value (ut not the same as symmetric 0 padding)
  - Tiled layouts
    - https://www.tensorflow.org/xla/tiled_layout
    - Indexing for tiling a larger matrix
    - Can be applied recursively
    - Likely used to tile problems for the TPU

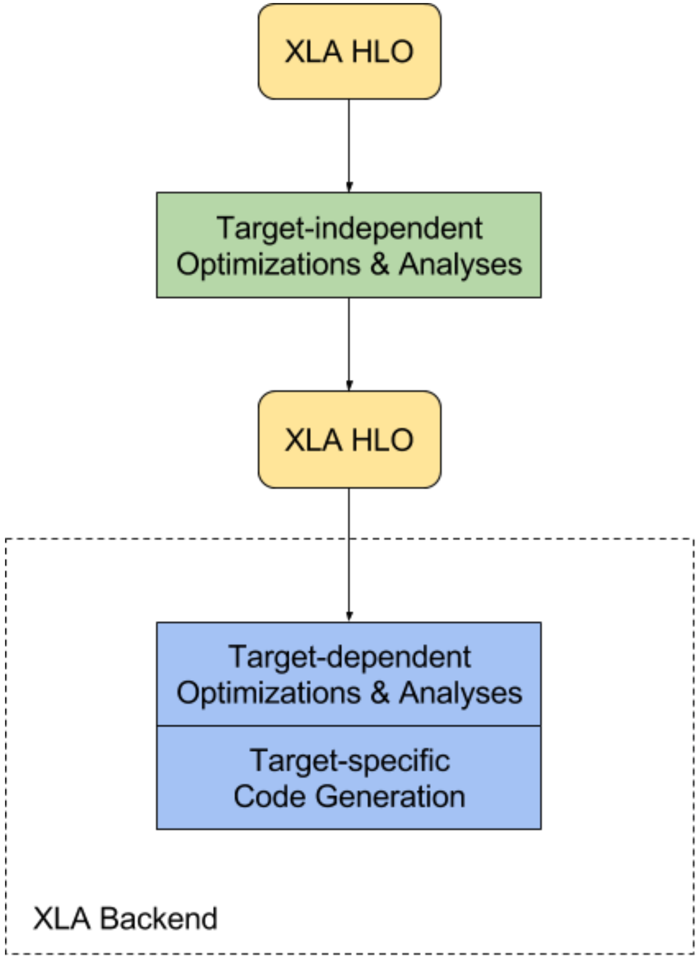


Figure from https://www.tensorflow.org/xla/overview

Google Sub Graph Optimization

- Methods for developing a new backend (to add a new processor)
  - https://www.tensorflow.org/xla/developing_new_backend
  - 3 scenarios
    - 1: CPU architecture with a LLVM backend
    - 2: non CPU architecture with a LLVM backend
    - 3: non CPU architecture without a LLVM backend
  - For scenario 3 need to implement the following functions
    - StreamExecutor to load and launch kernels and invoke pre canned library routines
    - xla::Compiler to encapsulate the compilation of a HLO computation to an xla::Executable
    - xla::Executable to launch a compiled computation to the platform
    - xla::TransferManager to transfer of data between device and host

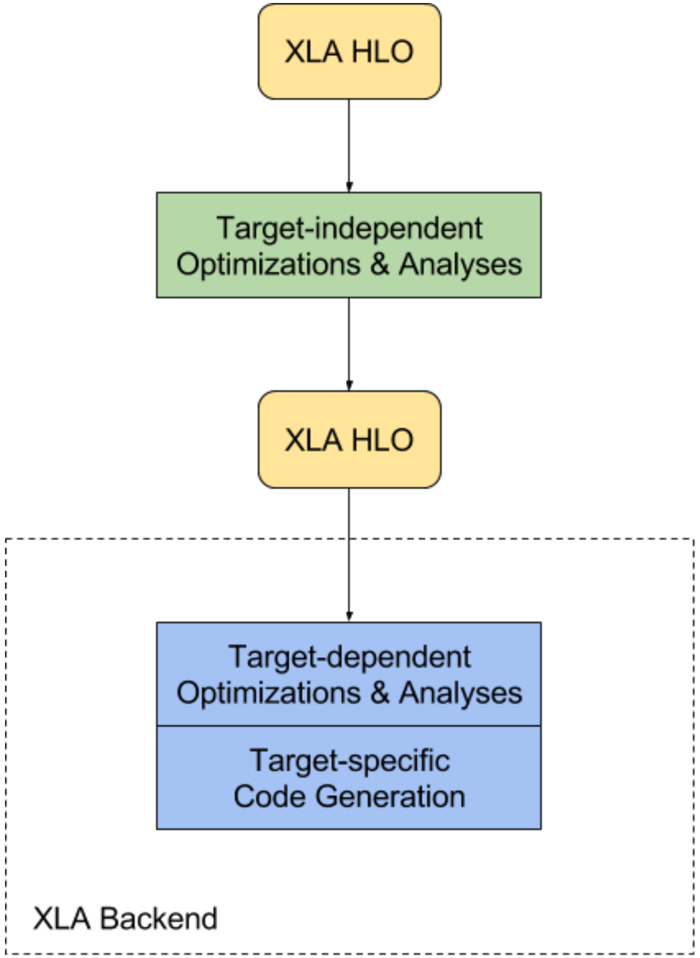


Figure from https://www.tensorflow.org/xla/overview

Google Sub Graph Optimization

- Ahead of time compiler
  - Appropriate for inference
  - tfcompile
    - https://www.tensorflow.org/xla/tfcompile
    - Compiles TensorFlow sub graphs into executable code
    - A sub graph is defined by feeds (input arguments) and fetches (output arguments)
    - Returns a compiled function that implements the sub graph
    - Does not use the TensorFlow runtime
    - Only has dependencies on the kernels in the sub graph
  - Colab example
    - https://www.tensorflow.org/xla/tutorials/xla_compile
    - Shows how to compile a model with specified inputs and output(s)
    - Currently doesn't support with tf.keras.Model.fit or eager mode
  - GitHub example
    - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/compiler/xla/g3doc/tfcompile.md
    - 1: Configure the sub graph to compile
    - 2: Use tf_library build macro to compile the sub graph
    - 3: Write code to invoke the sub graph
    - 4: Create the final binary

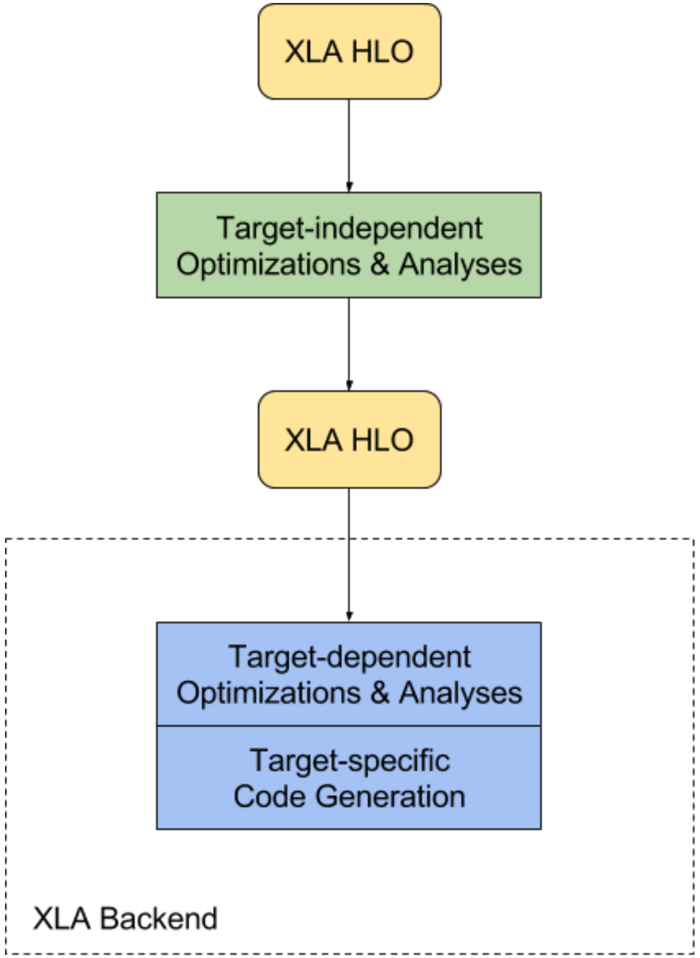


Figure from https://www.tensorflow.org/xla/overview

Google Sub Graph Optimization

- Just in time compiler
  - Appropriate for training
  - compile
    - https://www.tensorflow.org/xla/jit
    - Operations are marked on the graph for XLA
    - These operations are compiled into 1 or more kernels for the device
    - Operator fusion happens for consecutive graph nodes
  - GitHub example
    - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/compiler/xla/g3doc/jit.md

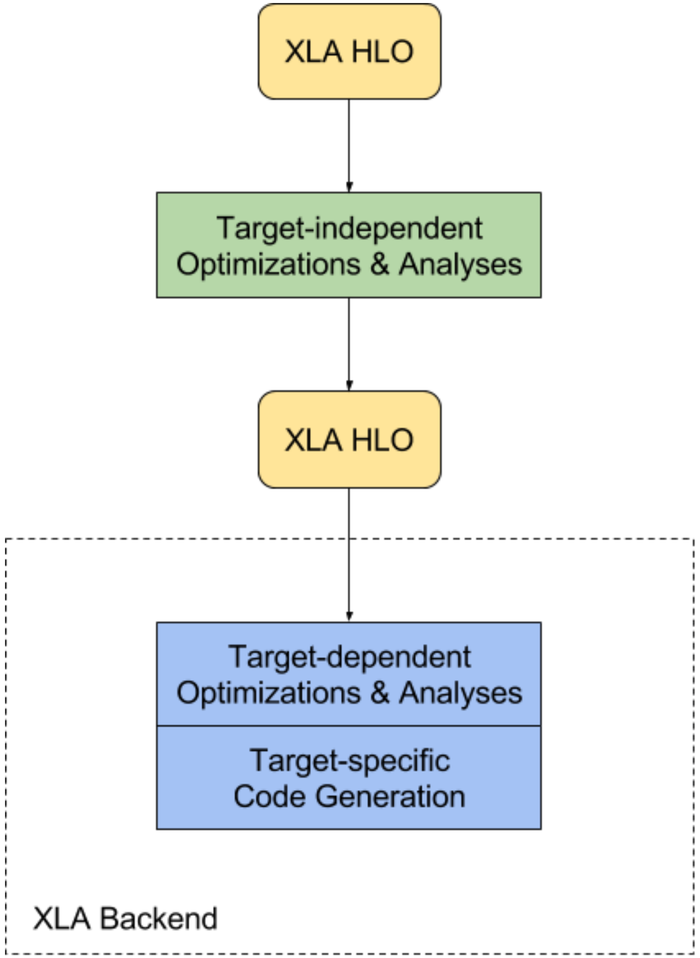


Figure from https://www.tensorflow.org/xla/overview

Google TensorFlow Graph Execution

Training

- Training (tf.keras.Model.fit)
  - https://www.tensorflow.org/guide/keras/training_and_evaluation
  - Applies the model to the training inputs and computes the loss between the true and predicted outputs
  - Uses the optimizer to update the model parameters to minimize the loss
  - Specifies training parameters like epochs, batch size and validation set
  - Tracks loss and metrics

- Distributed training
  - https://www.tensorflow.org/guide/distribute_strategy
  - https://www.tensorflow.org/tutorials/distribute/keras
  - https://www.tensorflow.org/tutorials/distribute/training_loops
  - https://www.tensorflow.org/tutorials/distribute/multi_worker_with_keras
  - MirroredStrategy: synchronous distributed training on multiple GPUs on one machine with each variable mirrored across all GPUs
  - CentralStorageStrategy: synchronous distributed training on multiple GPUs on one machine with variables on the CPU
  - MultiWorkerMirroredStrategy: MirroredStrategy with multiple machines
  - TPUStrategy: MirroredStrategy for TPUs
  - ParameterServerStrategy: some machines are parameter servers and some machines are workers

Google TensorFlow Graph Execution

Training utilities

- Saving and restoring
    - https://www.tensorflow.org/guide/keras/saving_and_serializing
    - Strategy for saving and restoring models created using the serial and functional APIs
        - https://www.tensorflow.org/guide/keras/saving_and_serializing#part_i_saving_sequential_models_or_functional_models
    - Strategy for saving and restoring models created via model sub classing
        - https://www.tensorflow.org/guide/keras/saving_and_serializing#saving_subclassed_models

- Options
    - Whole model saving
        - Architecture, weights, training configuration and optimizer state
        - Enables restarting training (useful, things will crash)
        - Can export to Keras or native TensorFlow formats
    - Architecture only saving
    - Weights only saving

Google TensorFlow Graph Execution

Training utilities

- Callbacks
  - Callbacks are functions applied during specific points in training that can be applied to fit, evaluate and predict
  - Built in callbacks are provided and custom callbacks can be defined by extending the base class keras.callbacks.Callback
  - https://keras.io/callbacks/
  - https://www.tensorflow.org/guide/keras/custom_callback
  - https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/callbacks

- Built in callbacks
  - History: records events into a History object
  - RemoteMonitor: stream events to a server
  - BaseLogger: accumulates epoch averages of metrics
  - ProgbarLogger: prints metrics to stdout
  - CSVLogger: streams epoch results to a csv file
  - TensorBoard: writes a log for TensorBoard
  - ModelCheckpoint: save the model after every epoch
  - LearningRateScheduler: learning rate scheduler
  - ReduceLROnPlateau: reduce learning rate when a metric has stopped improving
  - TerminateOnNaN: terminates training when a NaN loss is encountered
  - EarlyStopping: stop training when a monitored quantity has stopped improving
  - LambdaCallback: creates simple, custom callbacks on-the-fly

- In TensorFlow 2.x TensorBoard now works in Colab notebooks
  - https://www.tensorflow.org/tensorboard/r2/get_started

Google TensorFlow Graph Execution

Evaluation and prediction

- Evaluate (tf.keras.Model.evaluate)
  - https://www.tensorflow.org/guide/keras/training_and_evaluation
  - Applies the model to the validation inputs and computes the loss between the true and predicted outputs
  - Tracks loss and metrics (so it doesn't do gradient back prop and weight update)

- Predict (tf.keras.Model.predict)
  - https://www.tensorflow.org/guide/keras/training_and_evaluation
  - Generate predicted outputs from test inputs (so it doesn't do loss eval, gradient back prop and weight update)

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- Distributed master
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

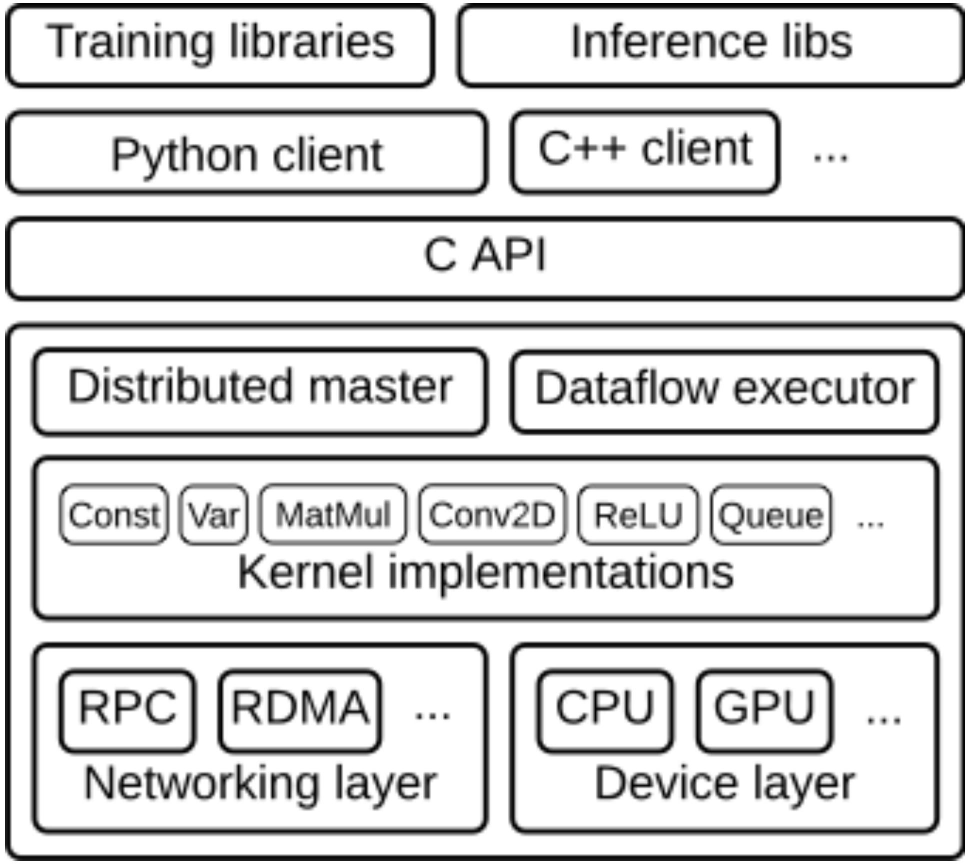


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- Distributed master
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

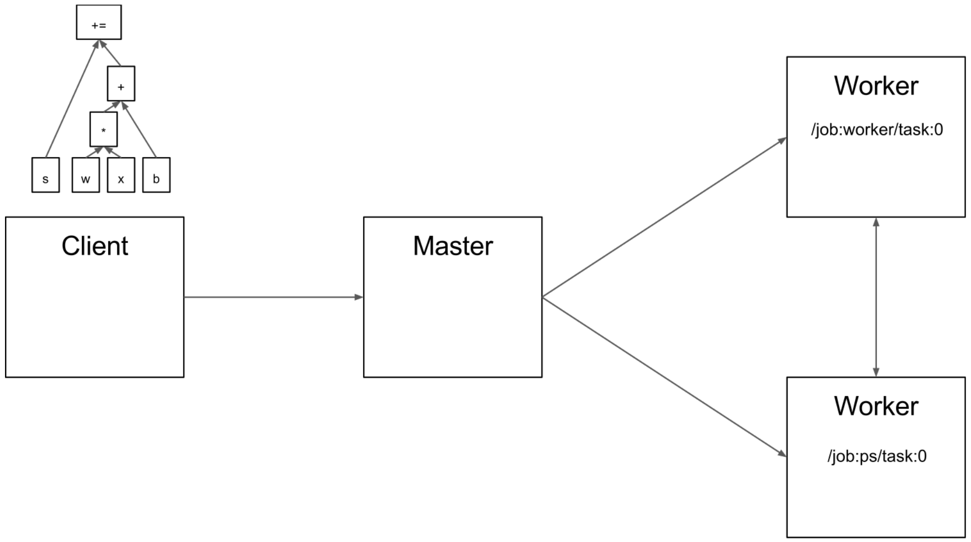


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- **Distributed master**
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

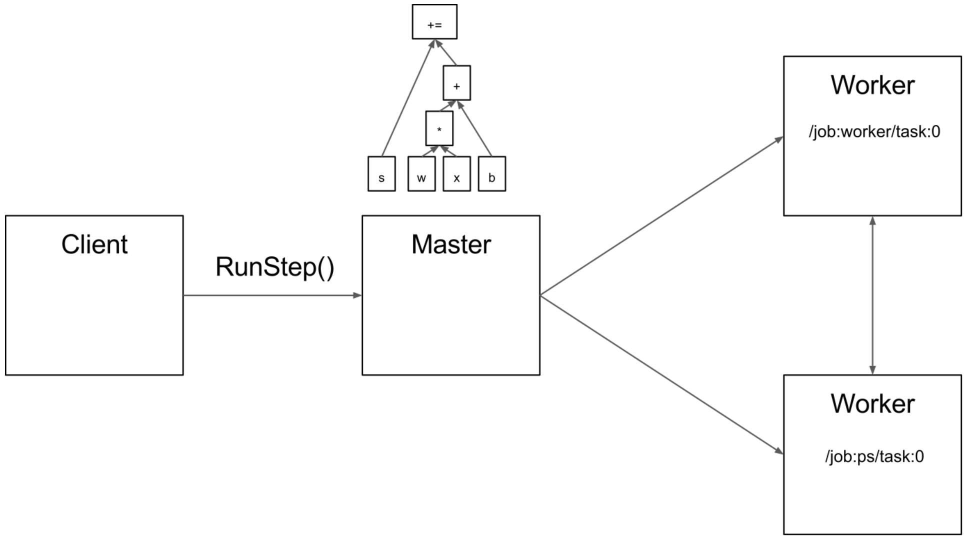


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- **Distributed master**
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

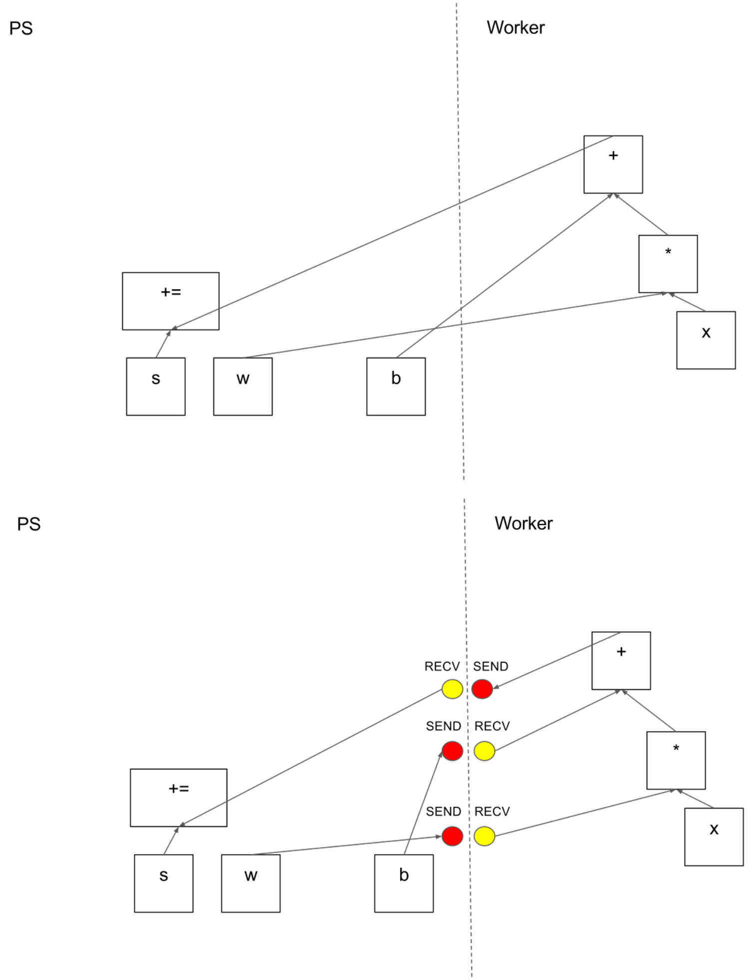


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- **Distributed master**
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

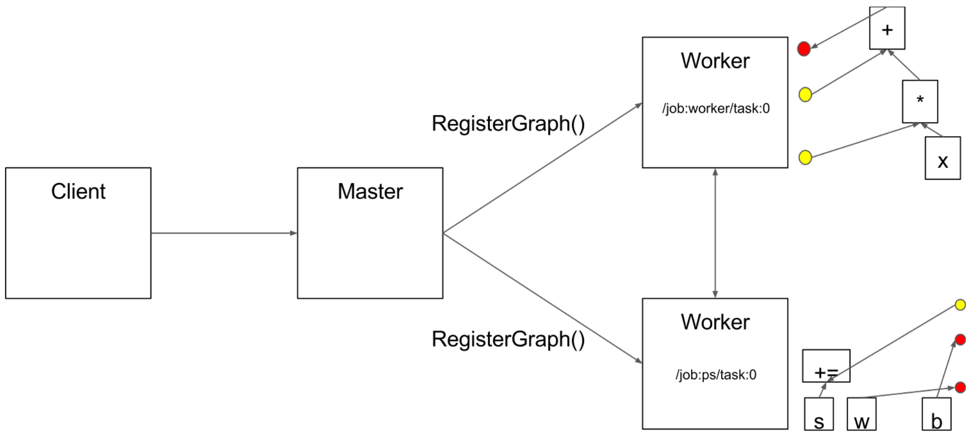


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Graph Execution

Runtime (large, training optimized)

- Calls to fit, evaluate and predict execute graphs
  - https://www.tensorflow.org/guide/extend/architecture
  - http://public.kevinrobinsonblog.com/docs/A%20tour%20through%20the%20TensorFlow%20codebase%20-%20v4.pdf
- Client
  - Defines the computation as a dataflow graph
  - Initiates graph execution using a session (or Keras API equivalent)
- Distributed master
  - Prunes a specific subgraph from the graph based on the arguments to Session.run()
  - Partitions the subgraph into multiple pieces that run in different processes and devices
  - Distributes the graph pieces to worker services
  - Initiates graph piece execution by worker services
- Worker services (1 for each task)
  - Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, ...)
  - Send and receive operation results to and from other worker services
- Kernel Implementations
  - Perform the computation for individual graph operations

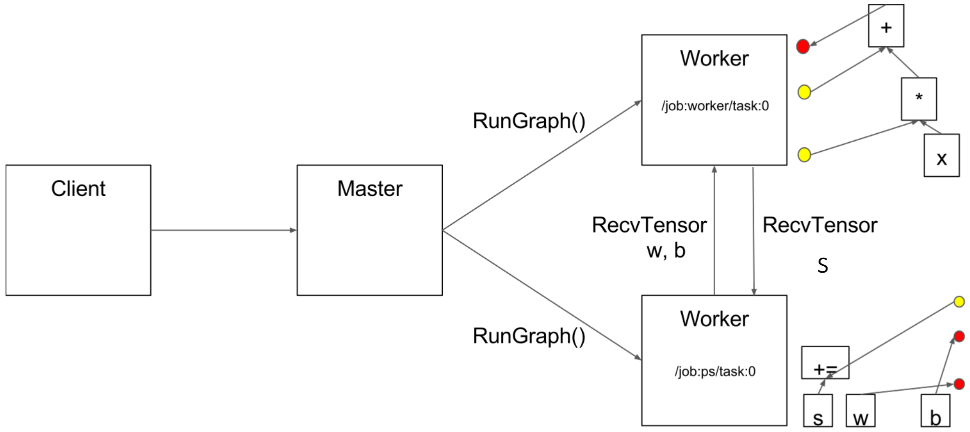
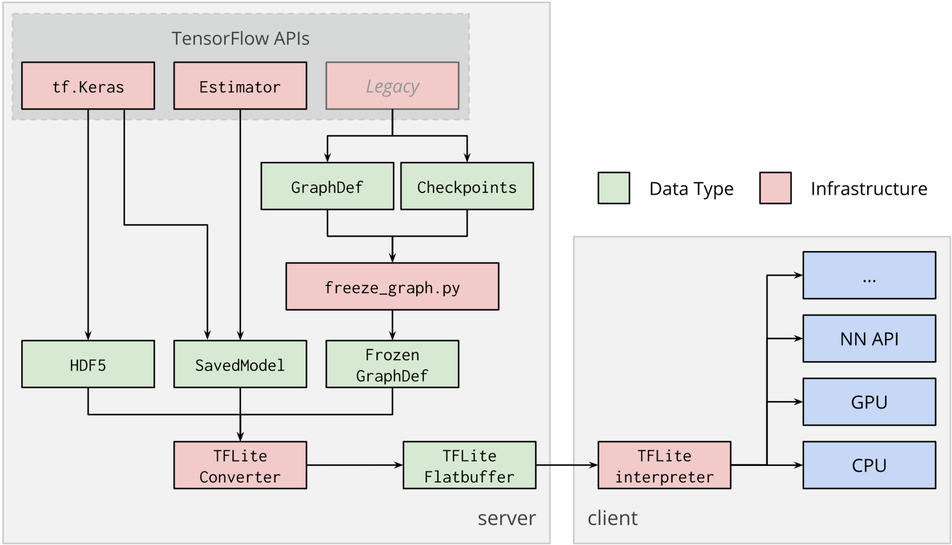


Figure from https://www.tensorflow.org/guide/extend/architecture

Google TensorFlow Lite Graph Execution

Runtime (small, inference optimized)

- TensorFlow Lite is Google's open source framework for using (inference) machine learning models on embedded devices
  - Takes around 200 kB of memory
  - https://www.tensorflow.org/lite
  - https://www.tensorflow.org/lite/guide
  - https://www.tensorflow.org/lite/guide/roadmap



TensorFlow 1.x → TensorFlow Lite

- Flow
  - 1. Create and train a TensorFlow model using supported operators
  - 2. Use the TensorFlow Lite Converter to convert the TensorFlow model to a TensorFlow Lite model
    - https://www.tensorflow.org/lite/convert/index
  - 3. Run the TensorFlow Lite model using the TensorFlow Lite Interpreter
    - https://www.tensorflow.org/lite/guide/inference

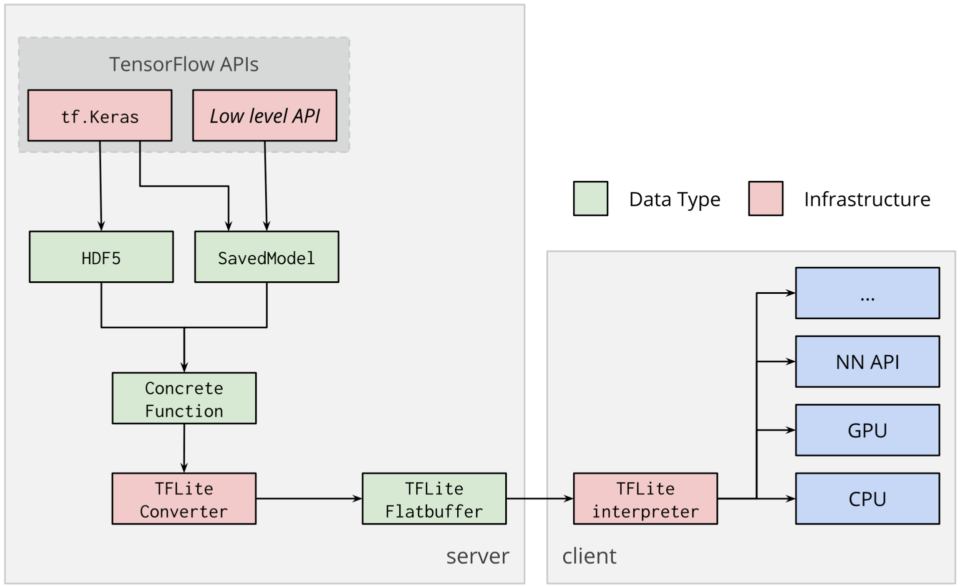
Figure from https://www.tensorflow.org/lite/convert

Google TensorFlow Lite Graph Execution

Runtime (small, inference optimized)

- TensorFlow Lite is Google's open source framework for using (inference) machine learning models on embedded devices
  - Takes around 200 kB of memory
  - https://www.tensorflow.org/lite
  - https://www.tensorflow.org/lite/guide
  - https://www.tensorflow.org/lite/guide/roadmap

- Flow
  - 1. Create and train a TensorFlow model using supported operators
  - 2. Use the TensorFlow Lite Converter to convert the TensorFlow model to a TensorFlow Lite model
    - https://www.tensorflow.org/lite/convert/index
  - 3. Run the TensorFlow Lite model using the TensorFlow Lite Interpreter
    - https://www.tensorflow.org/lite/guide/inference



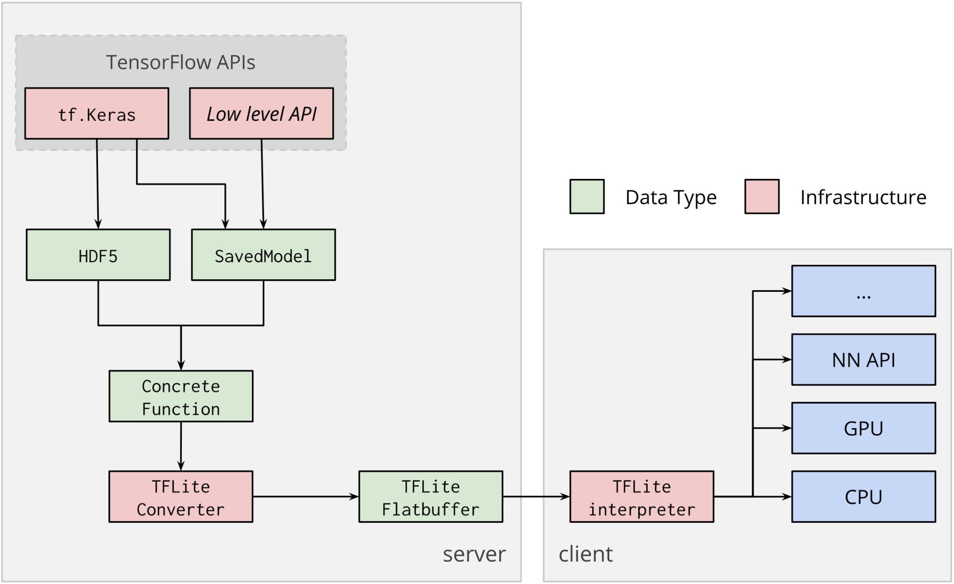
TensorFlow 2.x → TensorFlow Lite

Figure from https://www.tensorflow.org/lite/r2/convert

Google TensorFlow Lite Graph Execution

Runtime (small, inference optimized)

- Standard operators
    - Select TensorFlow ops supported out of the box
    - https://www.tensorflow.org/lite/guide/ops_compatibility
- Custom operators
    - Can be added
    - https://www.tensorflow.org/lite/guide/ops_custom
    - Require 4 functions with a C++ interface
        - Init() is called once at the beginning for every node on the graph and used to initialize variables
        - Free() is called at the end and used to free up space
        - Prepare() is called anytime input tensors are resized
        - Eval() is called at inference
    - Use global registration add the custom op to the built in op resolver
    - Example here
        - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/kernels/conv.cc



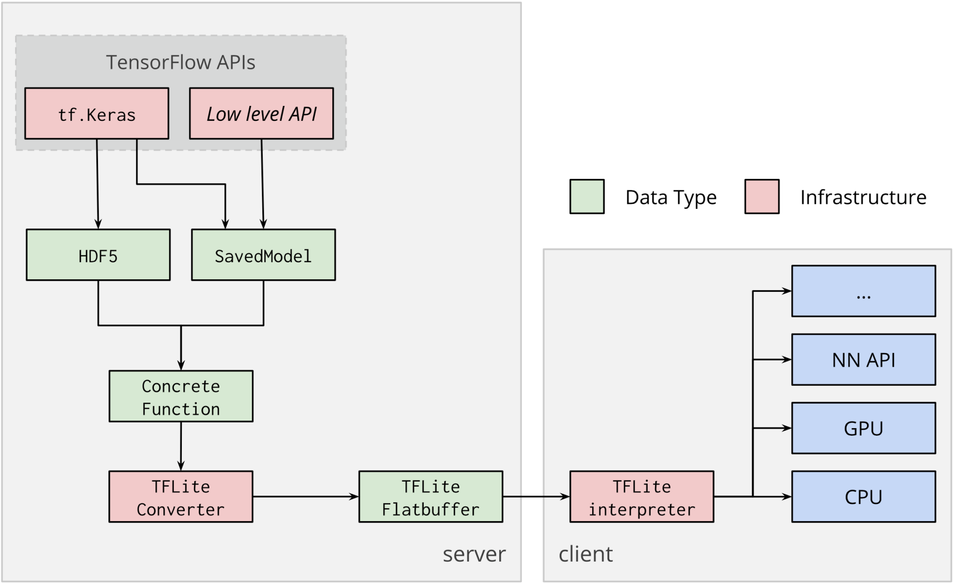
TensorFlow 2.x → TensorFlow Lite

Figure from https://www.tensorflow.org/lite/r2/convert

Google TensorFlow Lite Graph Execution

Runtime (small, inference optimized)

- Model training and conversion
  - Training
    - Quantization aware training / fake quantization (optional)
      - https://github.com/tensorflow/tensorflow/tree/r1.13/tensorflow/contrib/quantize
  - Trained TensorFlow model
    - Format can be SavedModels, Frozen GraphDef, Keras HDF5, tf.Session graph
    - In TensorFlow 2.x the default earer execution requires saving to a graph or creating a concrete function
  - Optimization (optional)
    - Model Optimization Toolkit
      - https://www.tensorflow.org/lite/performance/model_optimization
    - Quantization post training
      - https://medium.com/tensorflow/introducing-the-model-optimization-toolkit-for-tensorflow-254aca1ba0a3
      - https://www.tensorflow.org/lite/performance/post_training_quantization
    - Future support: pruning
      - https://medium.com/tensorflow/tensorflow-model-optimization-toolkit-pruning-api-42cac9157a6a?linkId=67380711
    - Future support: topology modification
  - Visualization
    - Graphviz in 1.x and visualize.py in 2.x
  - Converted TensorFlow Lite model
    - Format is TensorFlow Lite FlatBuffer



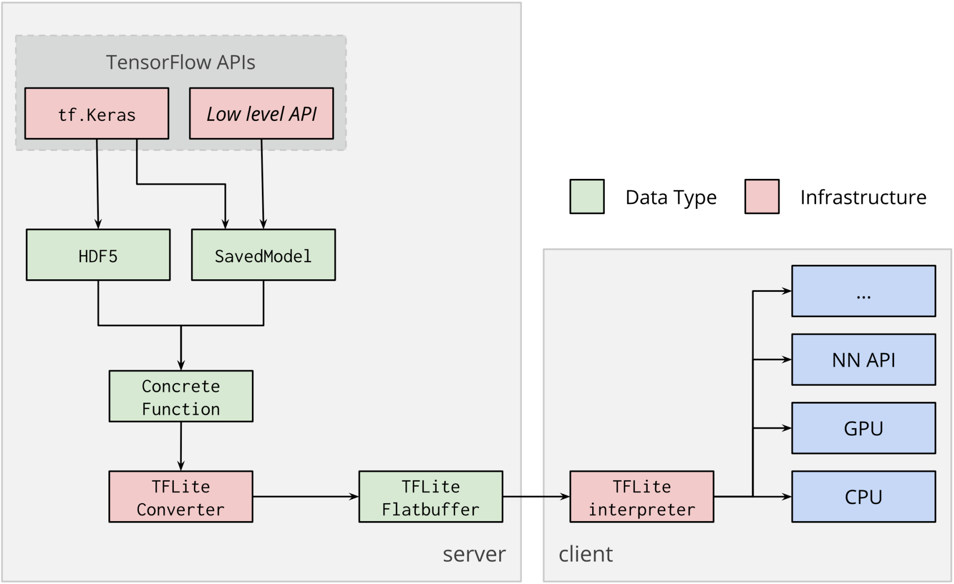
TensorFlow 2.x → TensorFlow Lite

Figure from https://www.tensorflow.org/lite/r2/convert

Google TensorFlow Lite Graph Execution

Runtime (small, inference optimized)

- Inference
  - Load the TensorFlow Lite FlatBuffer format model using the C++ or Python API
  - Build an Interpreter from the FlatBuffer format model
  - Resize inputs if necessary
  - Allocate memory
  - Set inputs
  - Perform inference
  - Read outputs



TensorFlow 2.x → TensorFlow Lite

Figure from https://www.tensorflow.org/lite/r2/convert

Google TensorFlow Lite Acceleration

Acceleration for Android

- Android NDK allows developers to implement parts of their Android apps in C/C++ or other languages
- The Android Neural Network API is a C API that can be used by runtimes such as TensorFlow Lite to offload operations on Android devices
  - Note that it can also be used directly to define models, though this is expected to be less common
- The Android Neural Network Runtime distributes these operations to various processors based on their capabilities with the CPU used as the fallback
- The Android Neural Network HAL allows processor vendors to connect specialized accelerators to the runtime

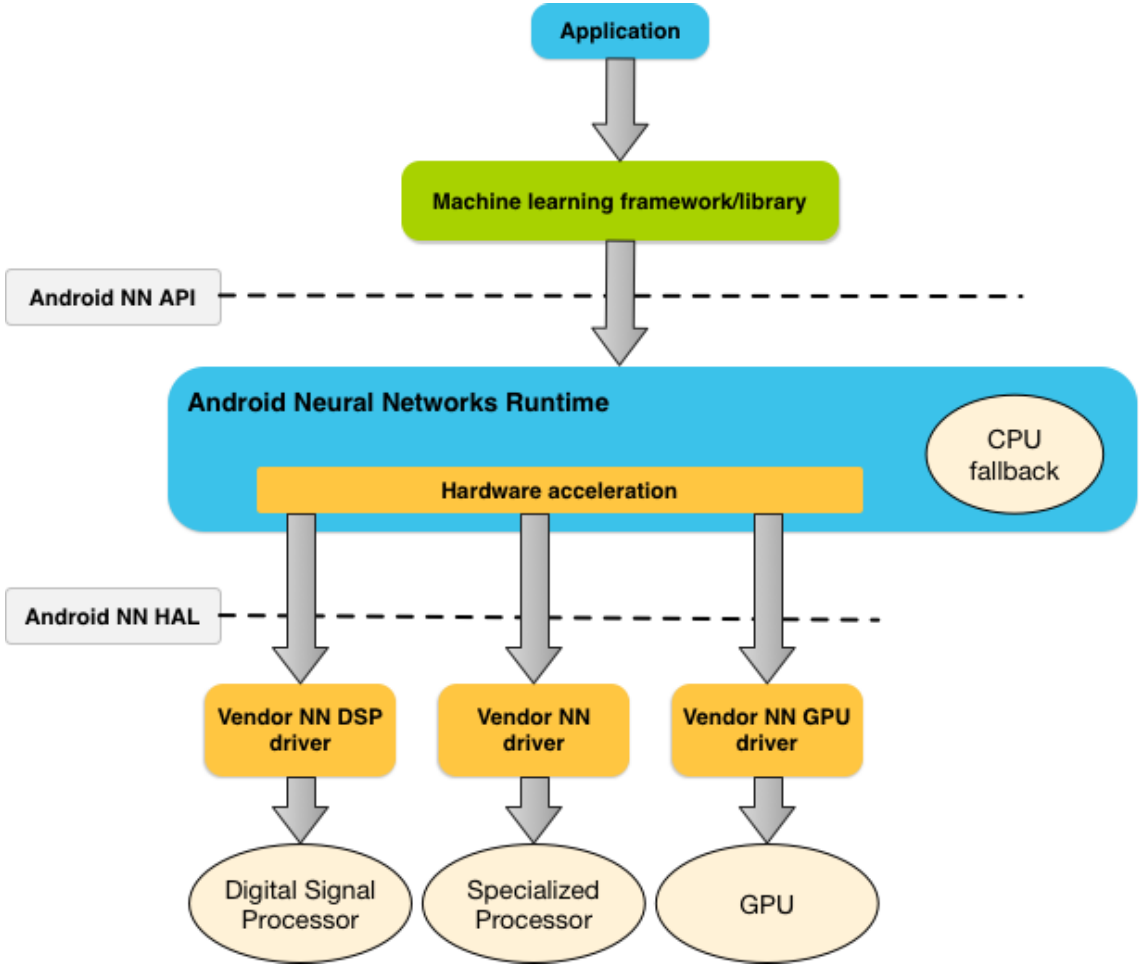


Figure from https://developer.android.com/ndk/guides/neuralnetworks/

Google TensorFlow Lite Acceleration

Acceleration for Android

- Model definition
  - Create a model instance
  - Add operands to the model specifying type, shape and quantization parameters (if applicable); for constants such as weights and biases specify their values at this point in time
  - Add operations to the model specifying type, input operands and output operands
  - Mark input and output operands for the model; these locations will be provided during execution
- Model compilation
  - Create a compilation instance
  - Set optional optimization preferences for power, single result time or throughput time
  - Compile; this determines where operations are run and allows hardware drivers to prepare for execution
- Model execution
  - Create an execution instance
  - Specify input and output operand locations
  - Start execution
  - Wait
  - Repeat for new inputs and outputs

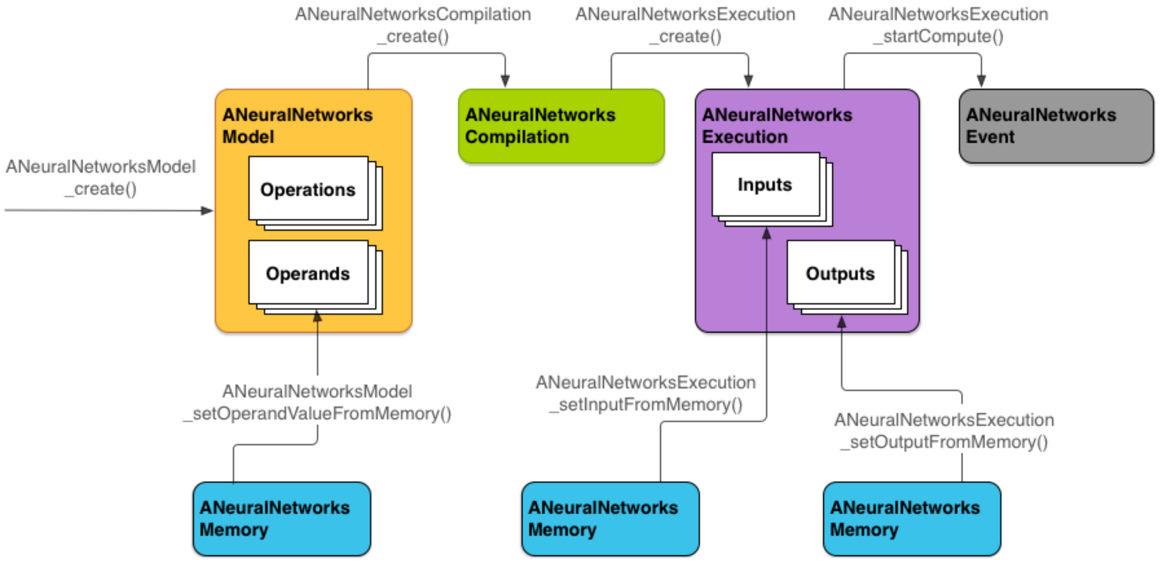


Figure from https://developer.android.com/ndk/guides/neuralnetworks/

Google TensorFlow Lite MCU Graph Execution

Runtime (very small, inference optimized)

- TensorFlow Lite Microcontroller is Google's open source framework for using (inference) machine learning models on very small embedded devices
    - Takes around 20 kB of memory (vs ~ 200 kB for TensorFlow Lite)
    - Does not require a high level OS, C / C++ standard libraries or dynamic memory allocation
    - https://www.tensorflow.org/lite/microcontrollers/overview
    - https://www.tensorflow.org/lite/microcontrollers/get_started
    - https://www.tensorflow.org/lite/microcontrollers/build_convert
    - https://www.tensorflow.org/lite/microcontrollers/library
    - https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/experimental/micro
- Flow
    - 1. Create or obtain a TensorFlow model (https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/experimental/micro/kernels/all_ops_resolver.cc)
    - 2. Convert the model to a TensorFlow Lite FlatBuffer
    - 3. Convert the FlatBuffer to a C byte array
    - 4. Integrate the TensorFlow Lite for Microcontrollers C++ library
    - 5. Deploy to your device

Google Common Graph IR

- MLIR ~ multi level IR
  - Many graph IRs currently exist within the TensorFlow ecosystem
  - Duplication of work and complications
  - Goal of MLIR is to use SSA based design to generalize and improve graphs at all levels
  - Many similarities to LLVM
- MLIR: a new intermediate representation and compiler framework
  - https://medium.com/tensorflow/mlir-a-new-intermediate-representation-and-compiler-framework-beba999ed18d
- Multi-Level intermediate representation overview
  - https://github.com/tensorflow/mlir
- MLIR primer: a compiler infrastructure for the end of Moore's law
  - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/1c082b766d8e14b54e36e37c9fc3ebbe8b4a72dd.pdf
- MLIR tutorial: building a compiler with MLIR
  - https://llvm.org/devmtg/2019-04/slides/Tutorial-AminiVasilacheZinenko-MLIR.pdf
- MLIR: multi-level intermediate representation compiler infrastructure
  - https://llvm.org/devmtg/2019-04/slides/Keynote-ShpeismanLattner-MLIR.pdf

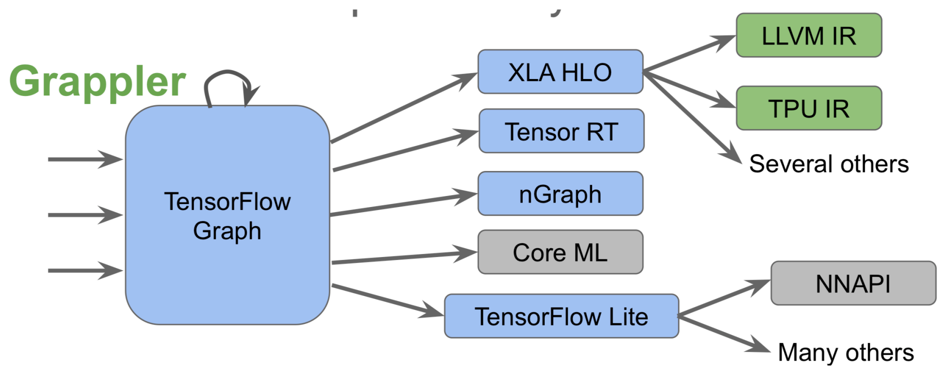


Figure from http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Google Common Graph IR

- MLIR ~ multi level IR
  - Many graph IRs currently exist within the TensorFlow ecosystem
  - Duplication of work and complications
  - Goal of MLIR is to use SSA based design to generalize and improve graphs at all levels
  - Many similarities to LLVM

- MLIR: a new intermediate representation and compiler framework
  - https://medium.com/tensorflow/mlir-a-new-intermediate-representation-and-compiler-framework-beba999ed18d
- Multi-Level intermediate representation overview
  - https://github.com/tensorflow/mlir
- MLIR primer: a compiler infrastructure for the end of Moore's law
  - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/1c082b766d8e14b54e36e37c9fc3ebbe8b4a72dd.pdf
- MLIR tutorial: building a compiler with MLIR
  - https://llvm.org/devmtg/2019-04/slides/Tutorial-AminiVasilacheZinenko-MLIR.pdf
- MLIR: multi-level intermediate representation compiler infrastructure
  - https://llvm.org/devmtg/2019-04/slides/Keynote-ShpeismanLattner-MLIR.pdf

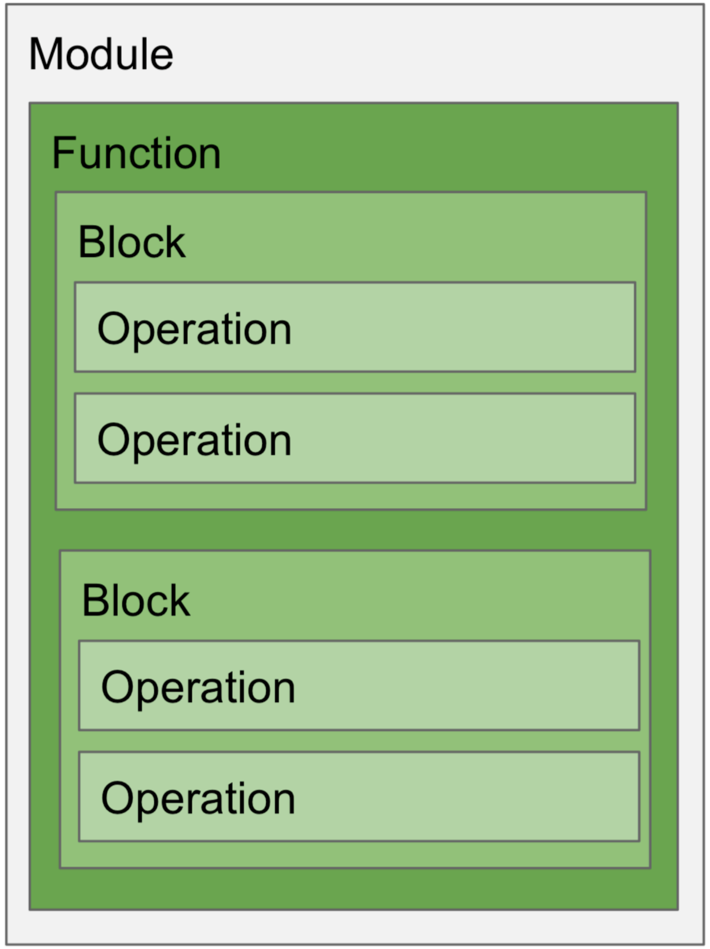


Figure from http://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Google More

- Chris Lattner: compilers, LLVM, Swift, TPU, and ML accelerators | Artificial Intelligence Podcast
  - https://www.youtube.com/watch?v=yCd3CzGSte8
- TF-Replicator: distributed machine learning for researchers
  - https://arxiv.org/abs/1902.00465
- AutoGraph: imperative-style coding with graph-based performance
  - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/14b4035de83550ea372162830e01f47313d6b41e.pdf

Intel nGraph

- A compiler stack designed to enable the connection of multiple front end framework options with multiple back end execution options without requiring a unique back end implementation for each front end framework

- Intel nGraph: an intermediate representation, compiler, and executor for deep learning
  - https://arxiv.org/abs/1801.08058

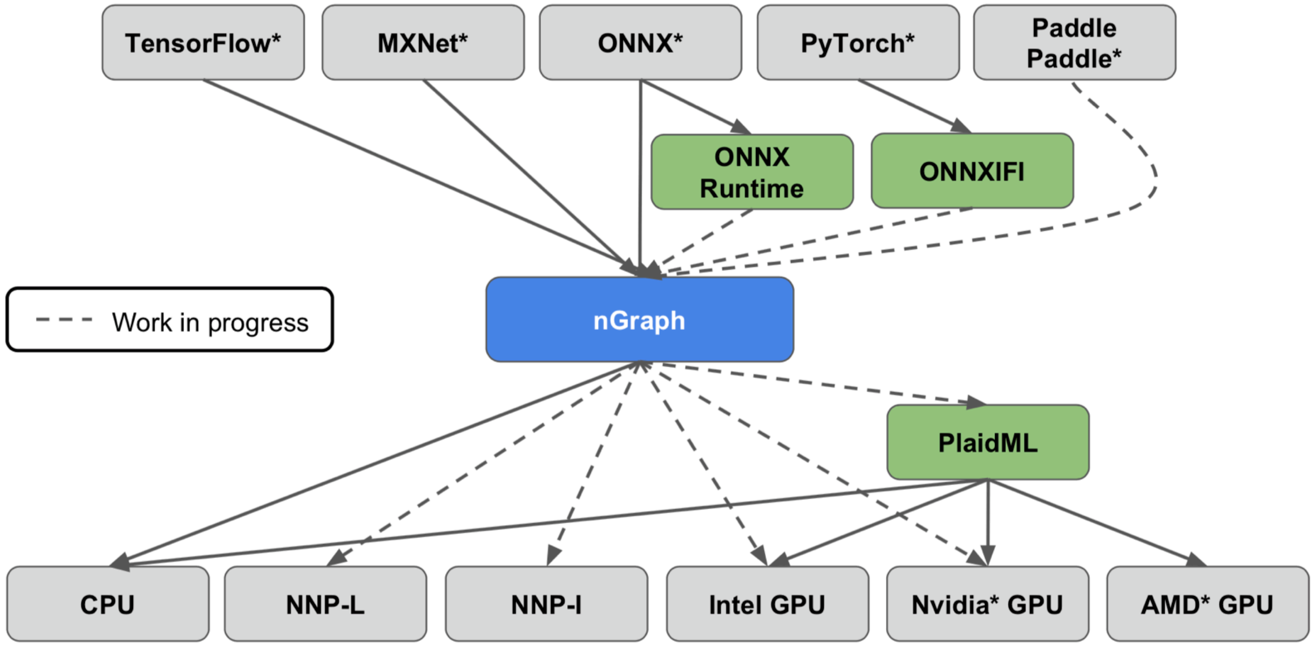


Figure from https://github.com/NervanaSystems/ngraph

Intel nGraph

- A compiler stack designed to enable the connection of multiple front end framework options with multiple back end execution options without requiring a unique back end implementation for each front end framework

- Separate APIs are defined for front end frameworks and back end execution options

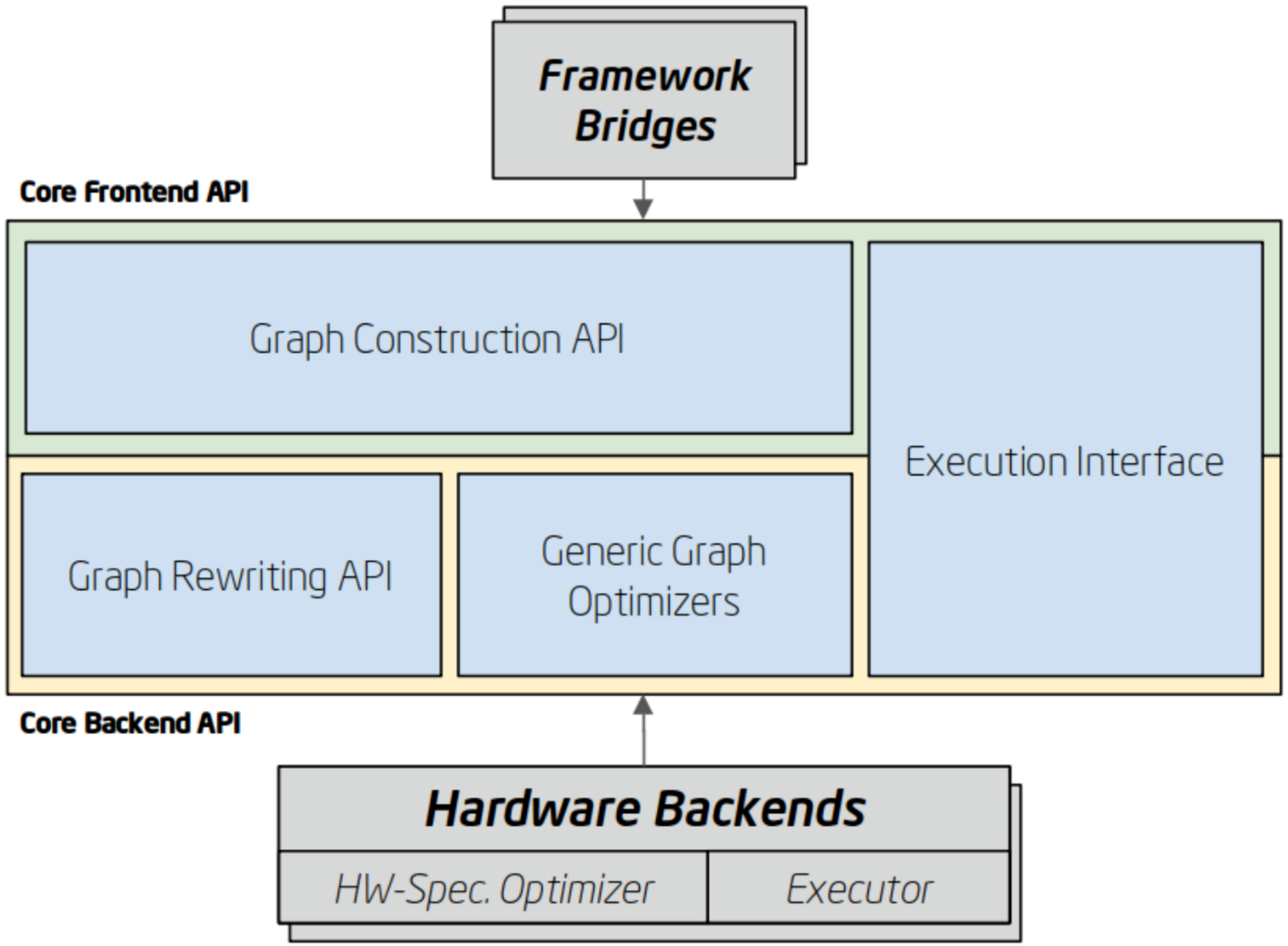


Figure from https://github.com/NervanaSystems/ngraph

Intel nGraph

- A compiler stack designed to enable the connection of multiple front end framework options with multiple back end execution options without requiring a unique back end implementation for each front end framework

- Sub graphs supported by back ends are replaced by single nodes
  - These sub graph can be optimized and implemented in any fashion by the back end compiler
  - They're replaced in the original graph by a single super node
  - Any remaining nodes fall back to the default CPU based implementation provided by the runtime

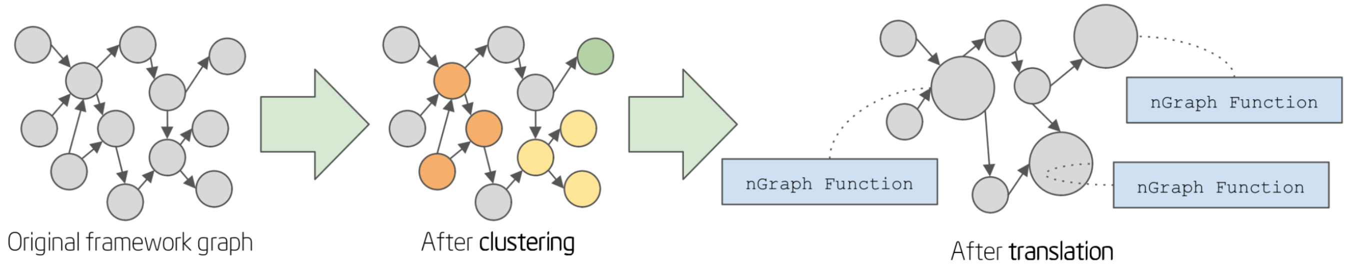


Figure from http://ngraph.nervanasys.com/index.html/core/overview.html

Intel nGraph

- A compiler stack designed to enable the connection of multiple front end framework options with multiple back end execution options without requiring a unique back end implementation for each front end framework

- Sub graphs supported by back ends are replaced by single nodes
  - These sub graph can be optimized and implemented in any fashion by the back end compiler
  - They're replaced in the original graph by a single super node
  - Any remaining nodes fall back to the default CPU based implementation provided by the runtime

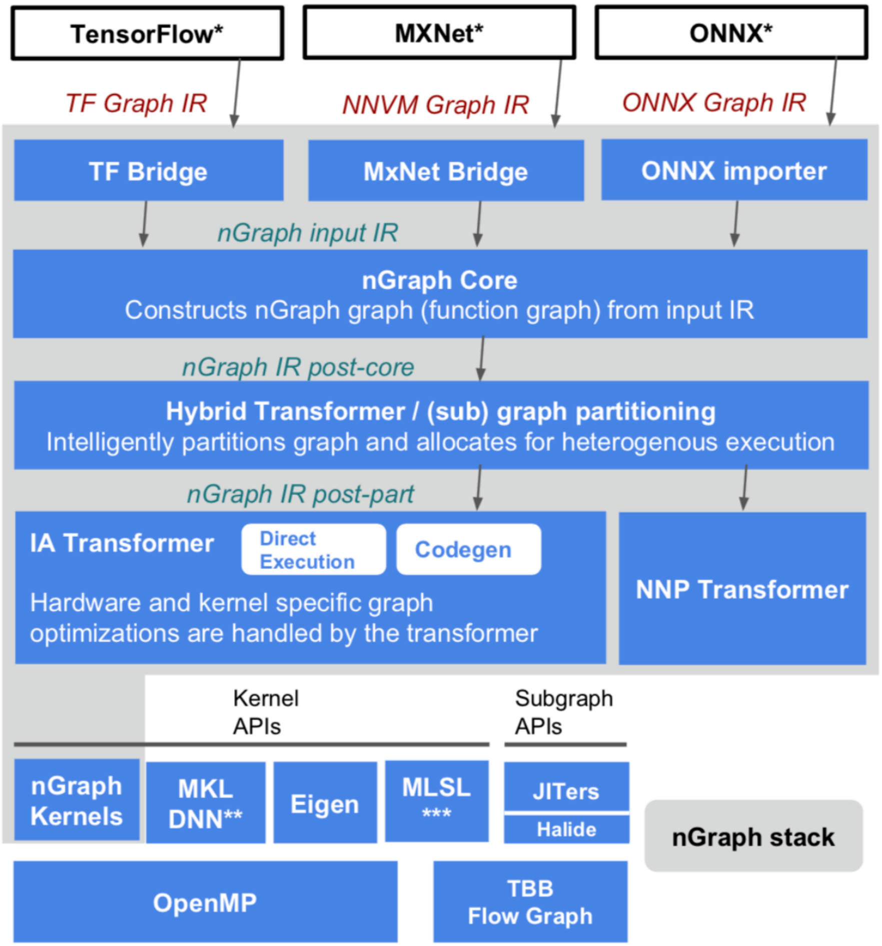


Figure from https://ngraph.nervanasys.com/docs/latest/project/about.html

Nvidia cuDNN

Operator library

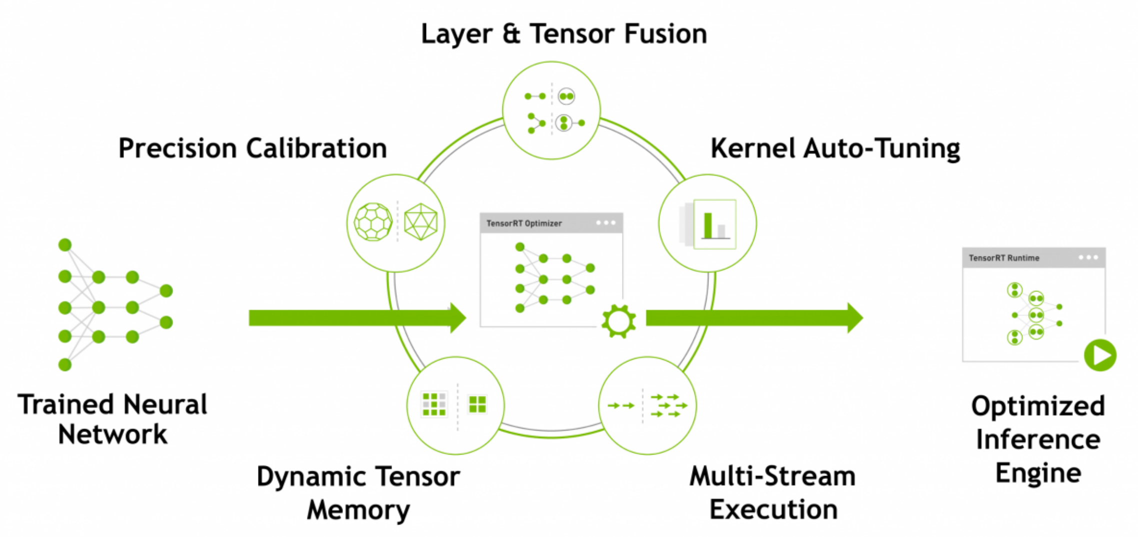
- Widely used by all xNN frameworks for operator acceleration on GPUs
  - cuDNN includes optimized versions of key primitives for xNNs
  - cuBLAS is used for GPU accelerated BLAS
  - NCCL is used for multi GPU communication

- Links
  - https://developer.nvidia.com/cudnn
  - https://docs.nvidia.com/deeplearning/sdk/index.html
  - https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html

Nvidia TensorRT

Inference runtime

- Nvidia TensorRT
  - Nvidia's C++ network optimizer and runtime engine for inference on Nvidia devices
  - Includes TensorFlow, PyTorch, theano, PaddlePaddle, mxnet, ONNX, Caffe and Caffe 2 model parsers
  - Includes C++ and Python APIs for programmatically creating models
  - Targets Nvidia Tesla, Drive, Jetson, NVDLA, ... hardware



- Links
  - https://developer.nvidia.com/tensorrt
  - https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html

Qualcomm Snapdragon NPE SDK

- Neural processing engine SDK
  - Runtime for xNN inference on Qualcomm Snapdragon devices
  - Support for models in Caffe, Caffe 2, ONNX and TensorFlow formats

- Links
  - https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk
  - https://developer.qualcomm.com/docs/snpe/index.html

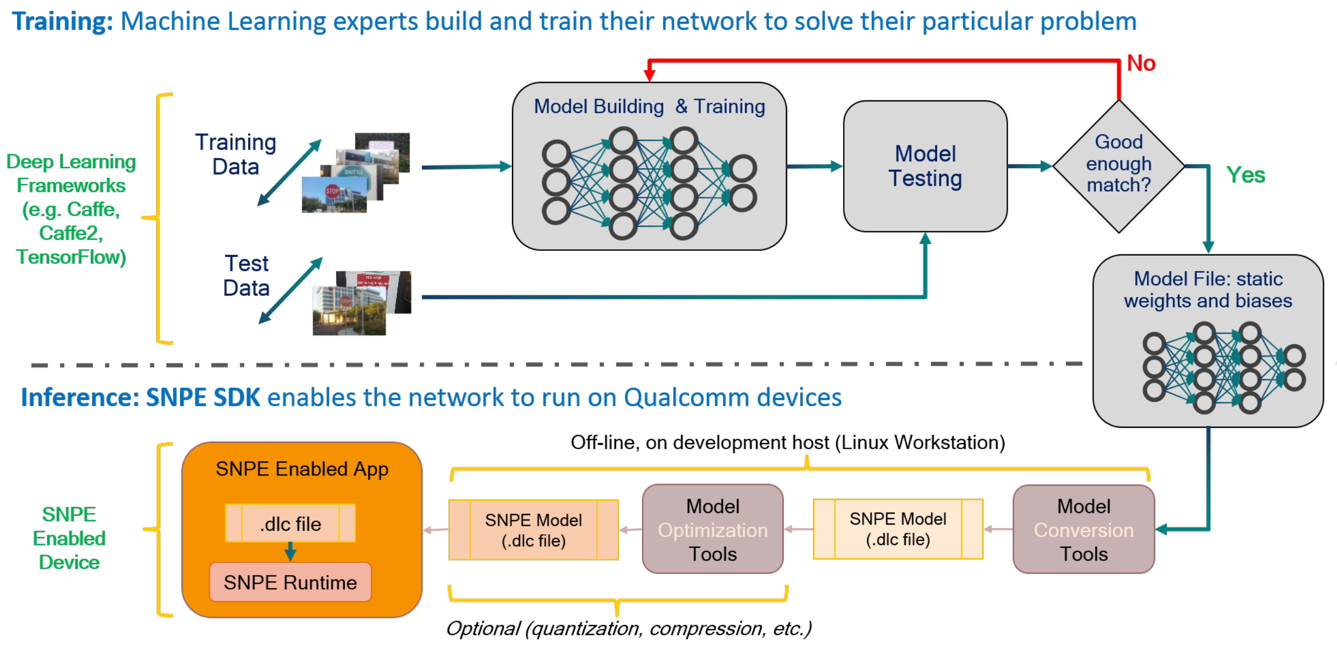


Figure from https://developer.qualcomm.com/docs/snpe/overview.html

UW TVM

- Relay
  - Common high level graph format and opt
    - Pruning, fusion, layout transformation, memory management, ...
  - Registers operators to TVM implementations
  - https://docs.tvm.ai/langref/index.html
- TVM
  - Tensor operator optimization and code gen
    - 1. Tensor expression language to express operators as different program options separating scheduling and hardware intrinsics
    - 2. Automated program optimizer
    - 3. Graph re writer
  - https://tvm.ai and https://arxiv.org/abs/1802.04799
- TOPI: TVM operator inventory
  - Pre made TVM operator recipes
  - https://github.com/dmlc/tvm/tree/master/topi

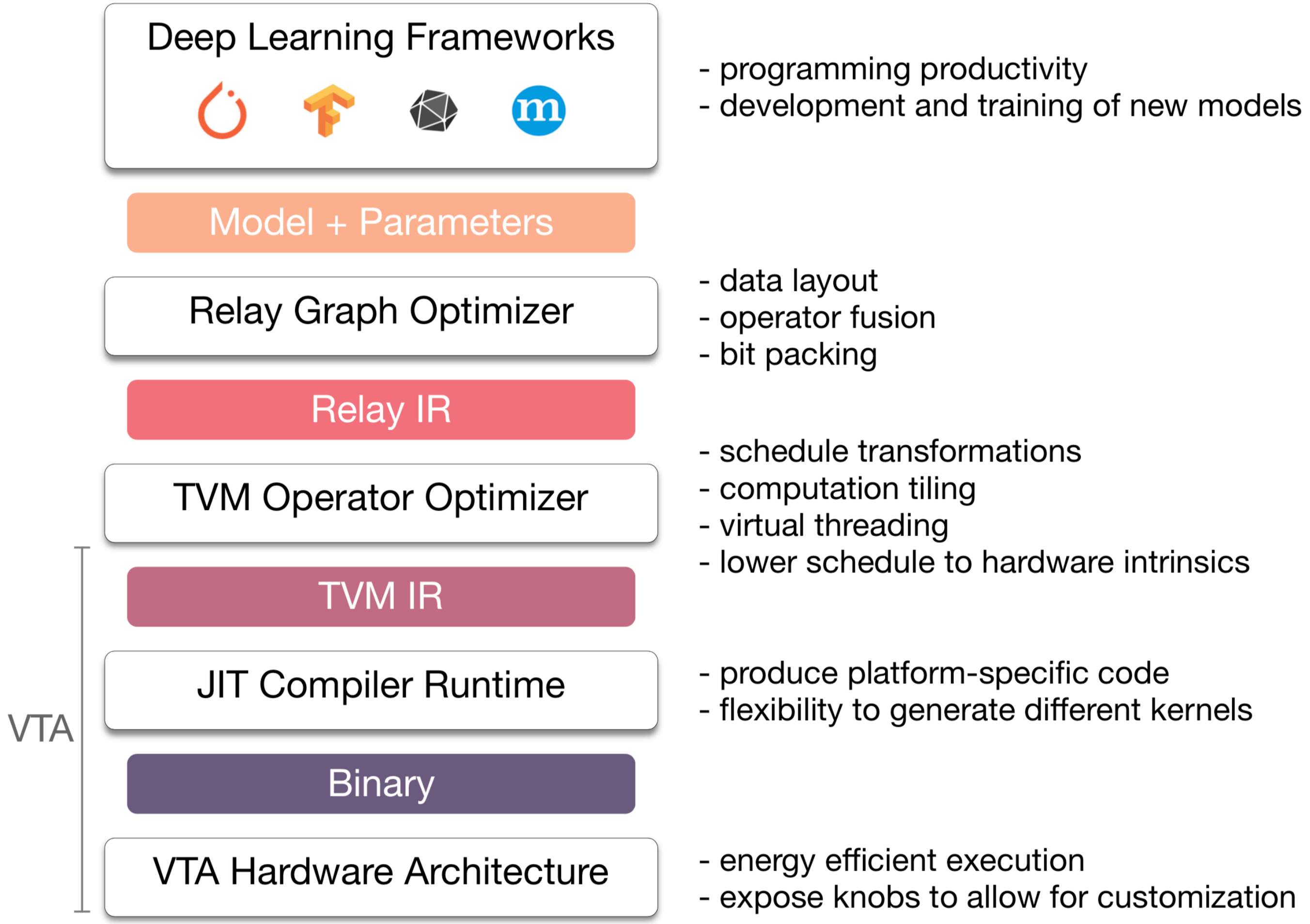


Figure from https://arxiv.org/abs/1807.04188

UW TVM

- Relay
  - Common high level graph format and opt
    - Pruning, fusion, layout transformation, memory management, ...
  - Registers operators to TVM implementations
  - https://docs.tvm.ai/langref/index.html
- TVM
  - Tensor operator optimization and code gen
    - 1. Tensor expression language to express operators as different program options separating scheduling and hardware intrinsics
    - 2. Automated program optimizer
    - 3. Graph re writer
  - https://tvm.ai and https://arxiv.org/abs/1802.04799
- TOPI: TVM operator inventory
  - Pre made TVM operator recipes
  - https://github.com/dmlc/tvm/tree/master/topi

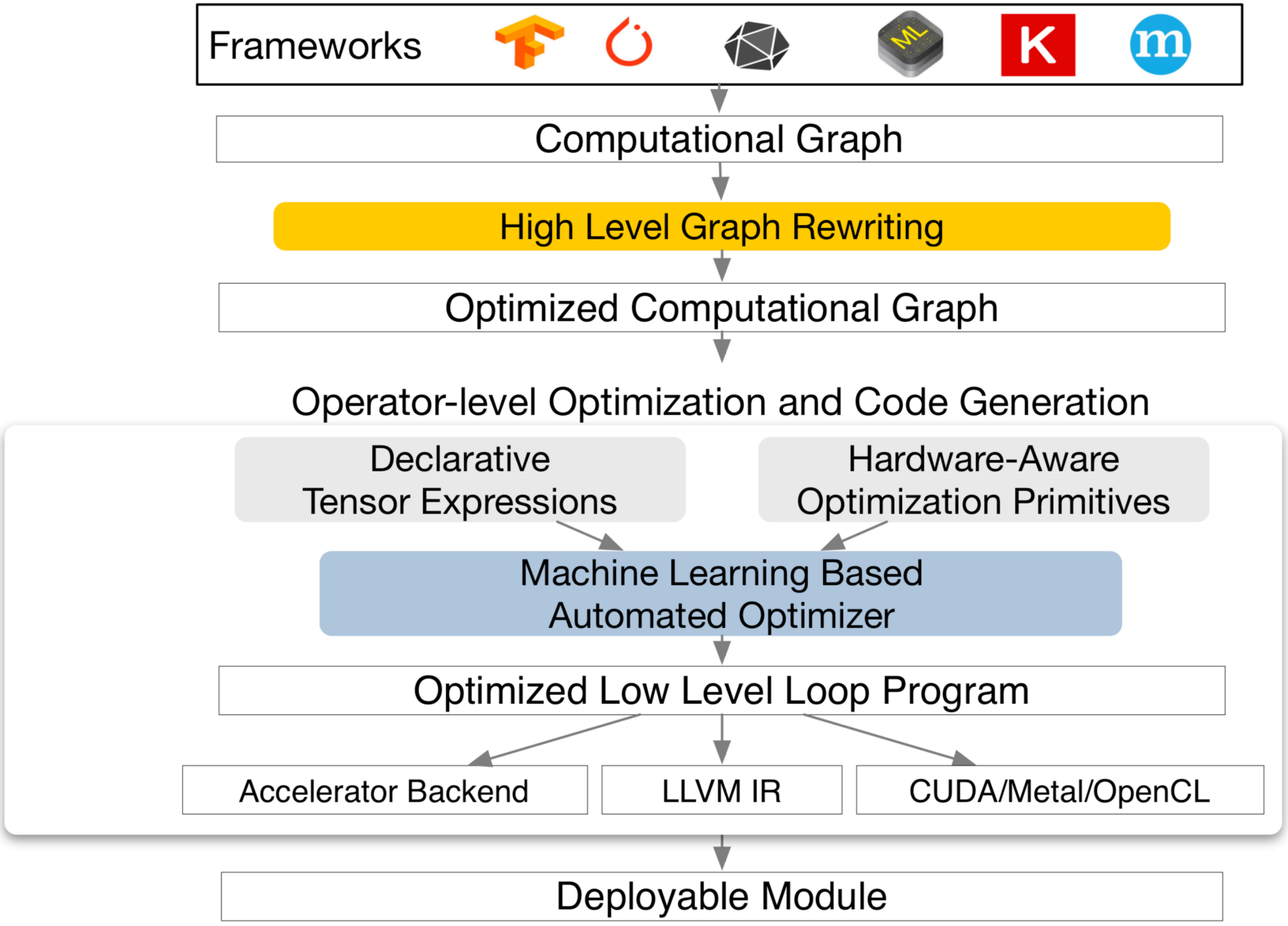


Figure from https://arxiv.org/abs/1802.04799

UW TVM

- Relay
  - Common high level graph format and opt
    - Pruning, fusion, layout transformation, memory management, ...
  - Registers operators to TVM implementations
  - https://docs.tvm.ai/langref/index.html
- TVM
  - Tensor operator optimization and code gen
    - 1. Tensor expression language to express operators as different program options separating scheduling and hardware intrinsics
    - 2. Automated program optimizer
    - 3. Graph re writer
  - https://tvm.ai and https://arxiv.org/abs/1802.04799
- TOPI: TVM operator inventory
  - Pre made TVM operator recipes
  - https://github.com/dmlc/tvm/tree/master/topi

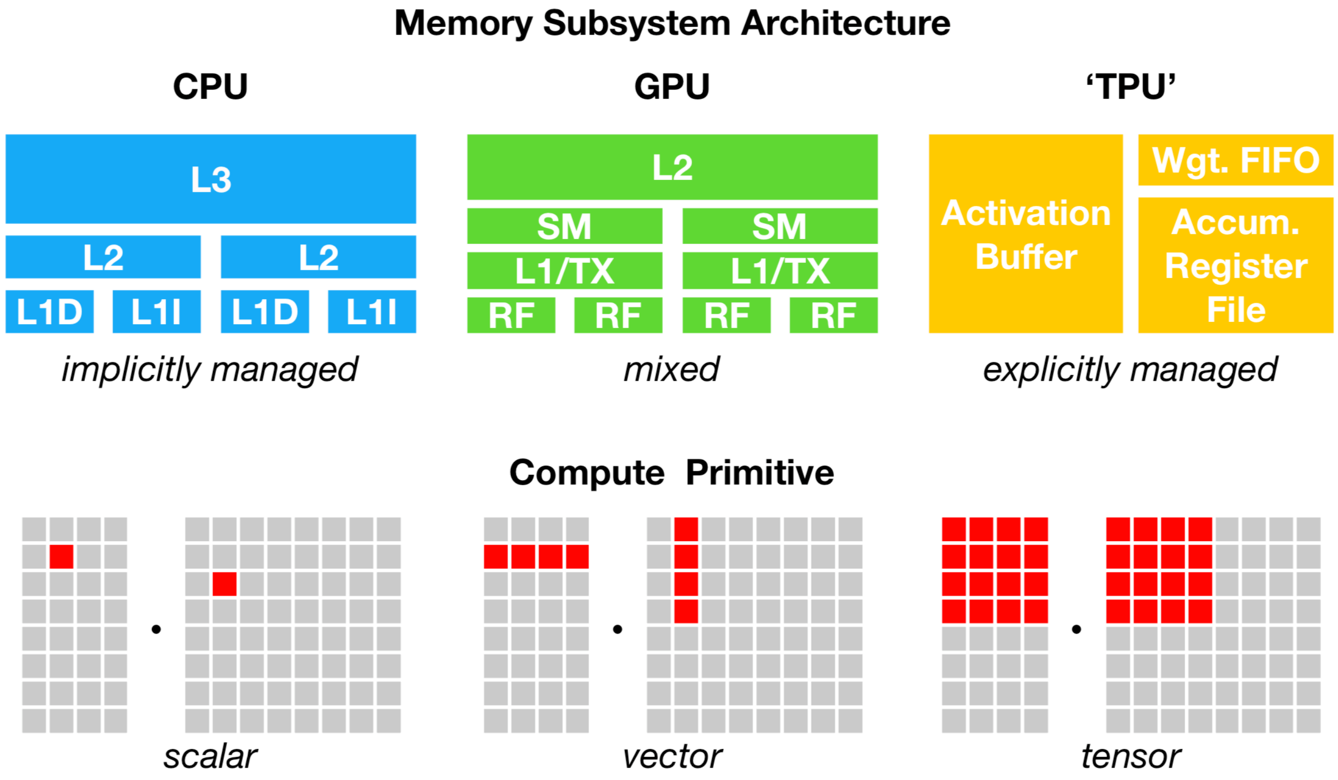


Figure from https://arxiv.org/abs/1802.04799

Xilinx

- Links
  - https://www.xilinx.com/products/design-tools/deephi.html#overview
  - https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

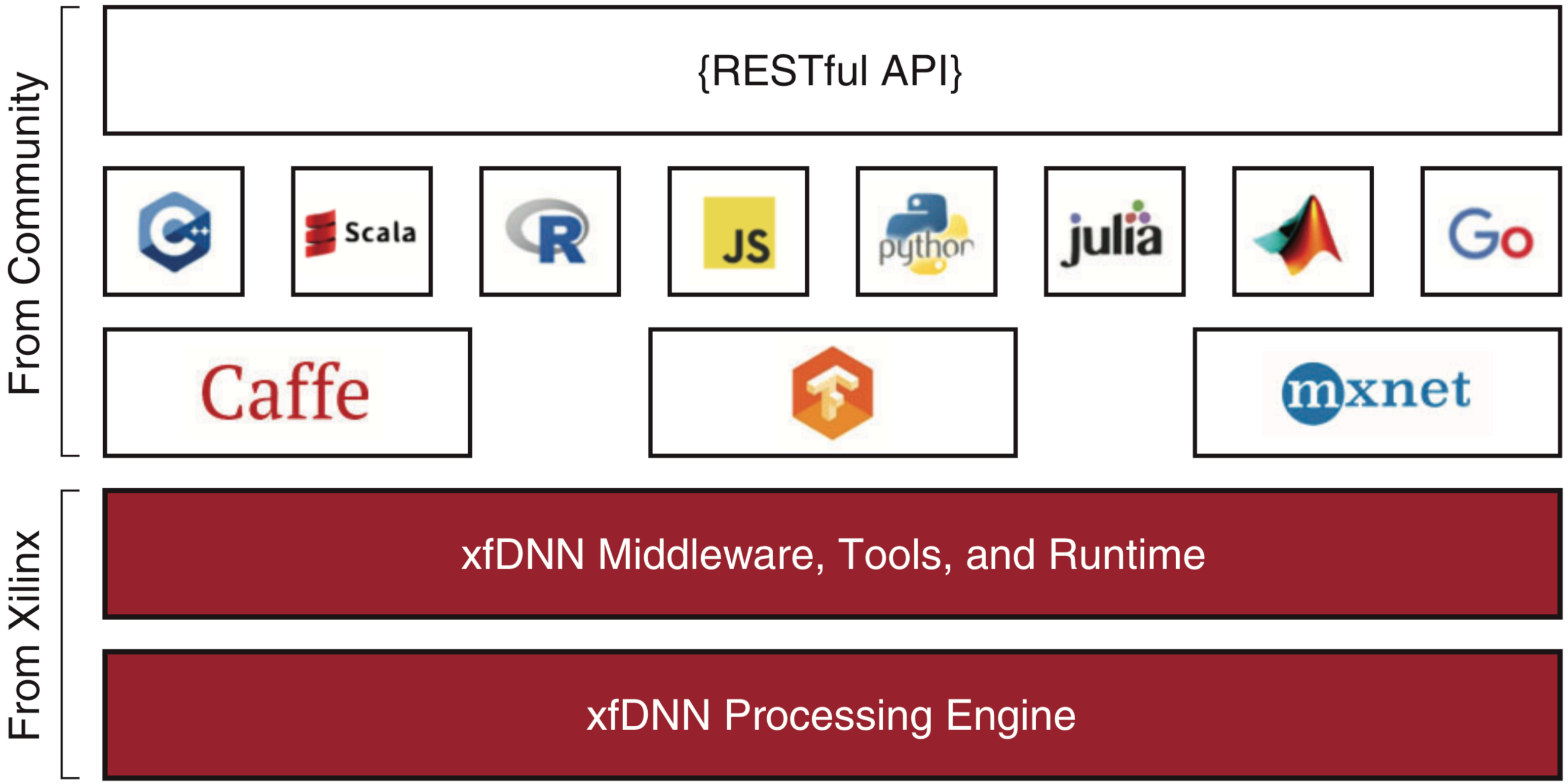
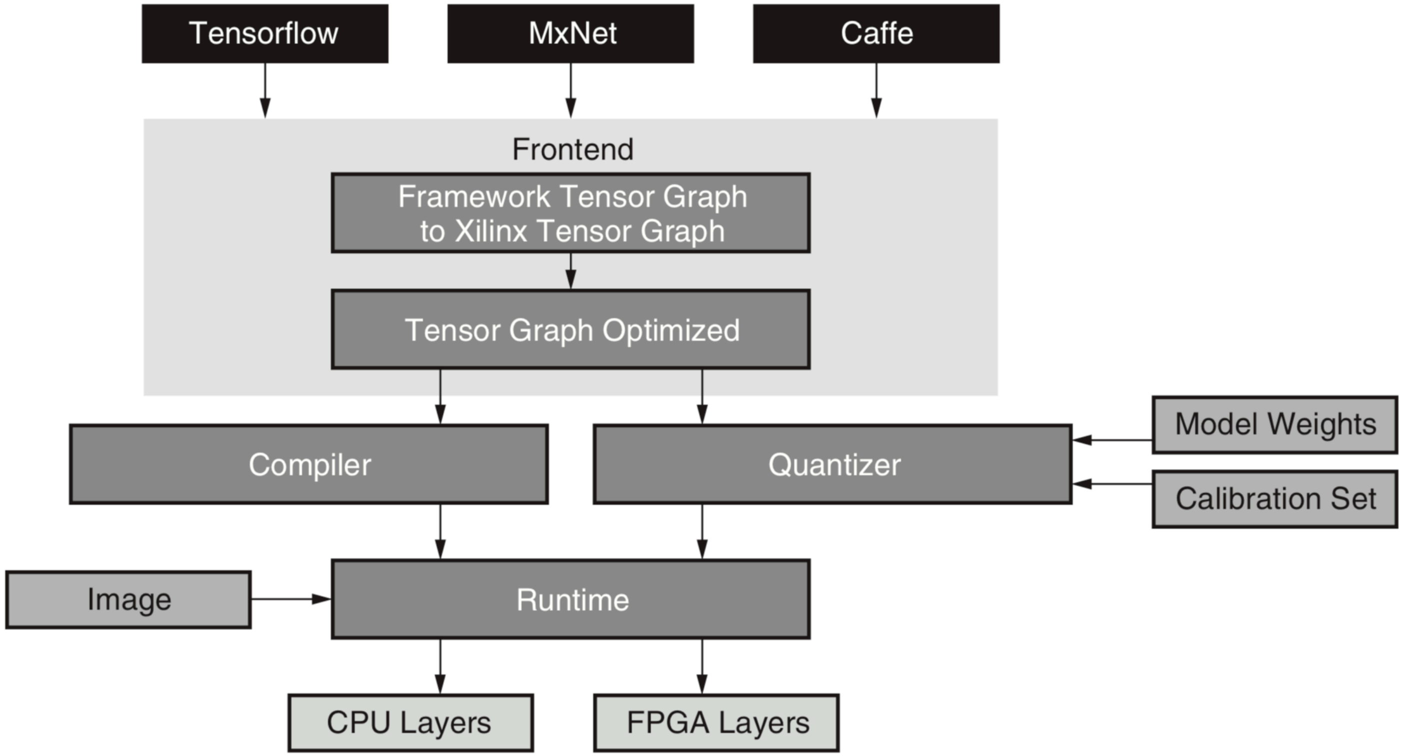


Figure from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Xilinx



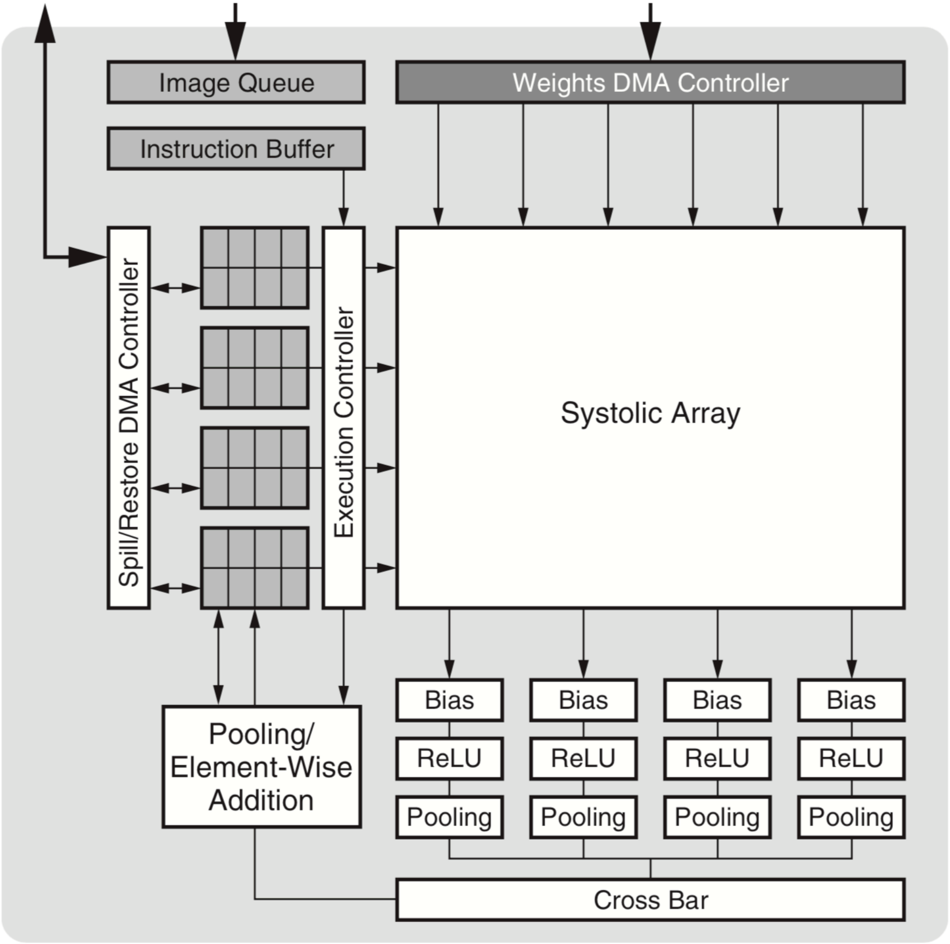


Figure from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Xilinx

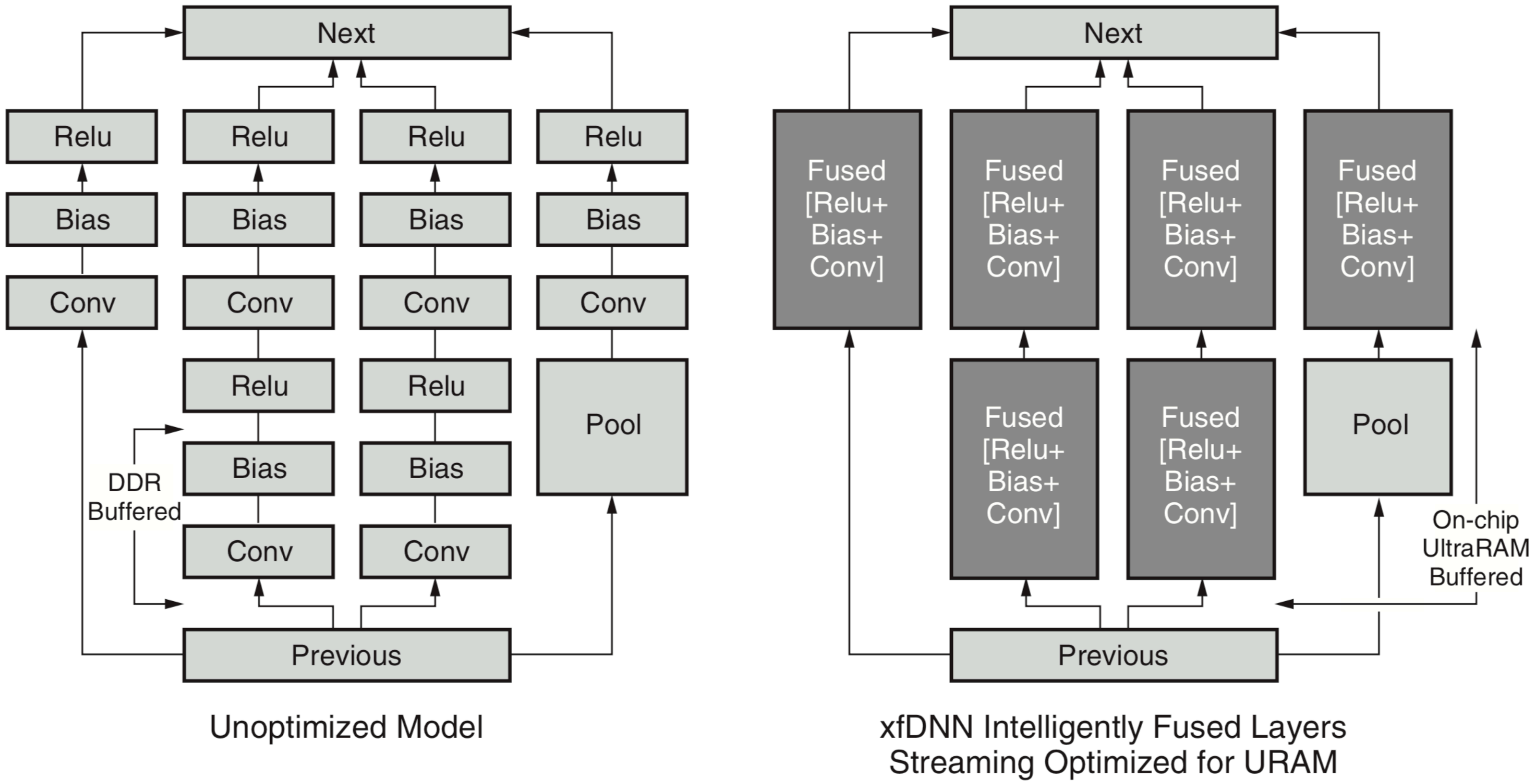
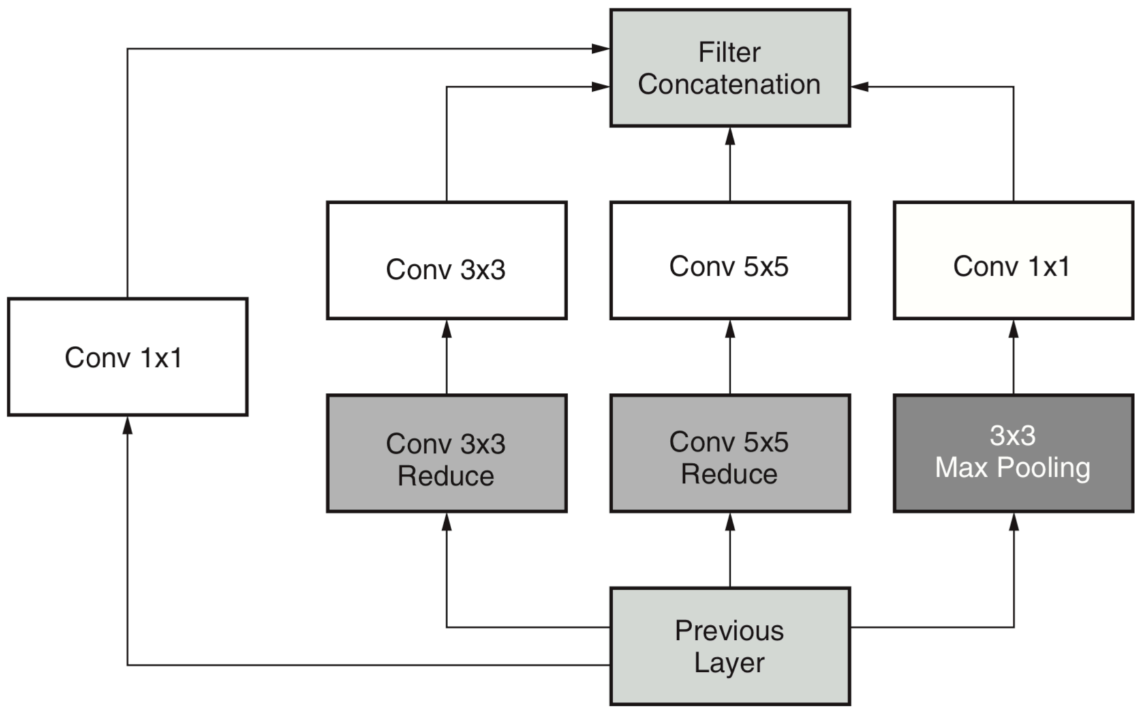
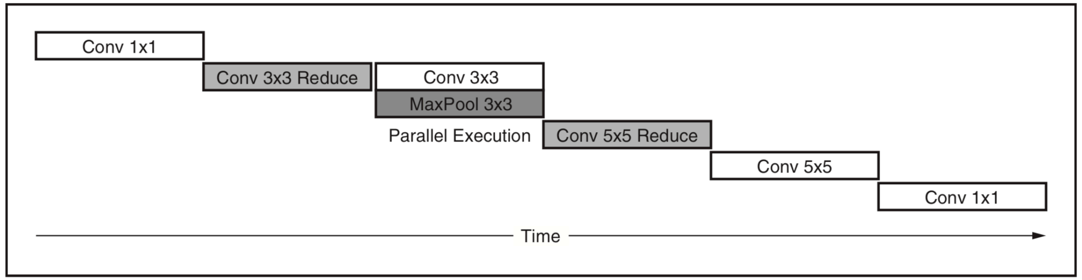


Figure from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Xilinx





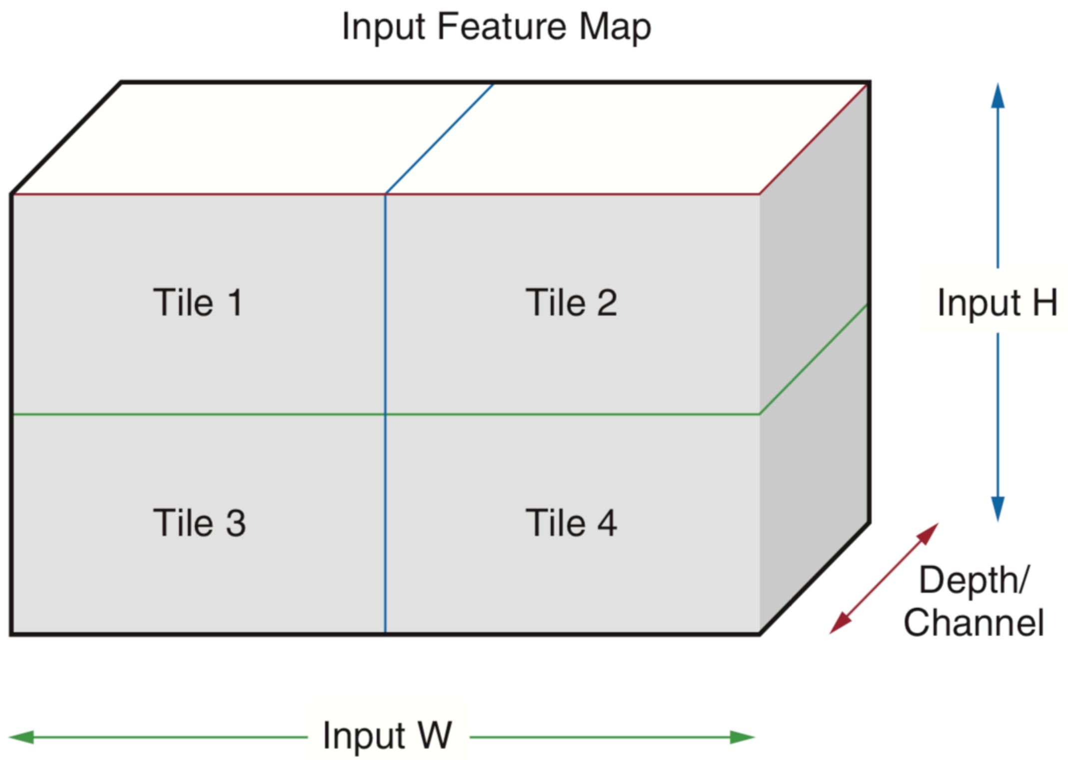


Figure from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Xilinx

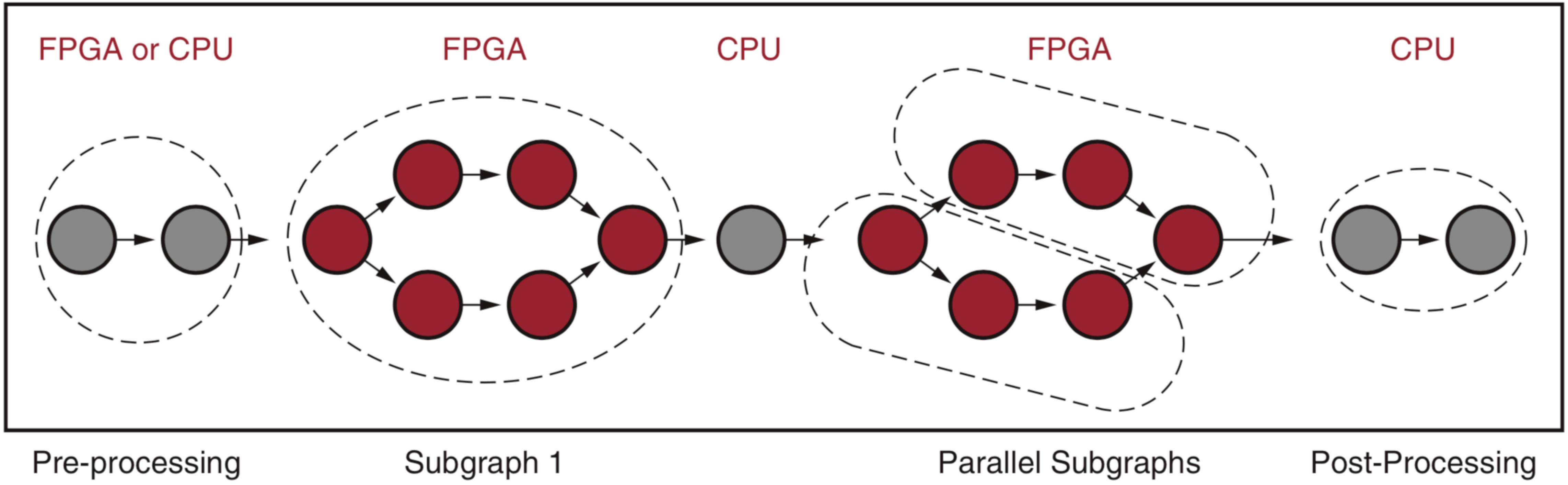


Figure from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf

Xilinx

High level software / hardware view for Versal devices

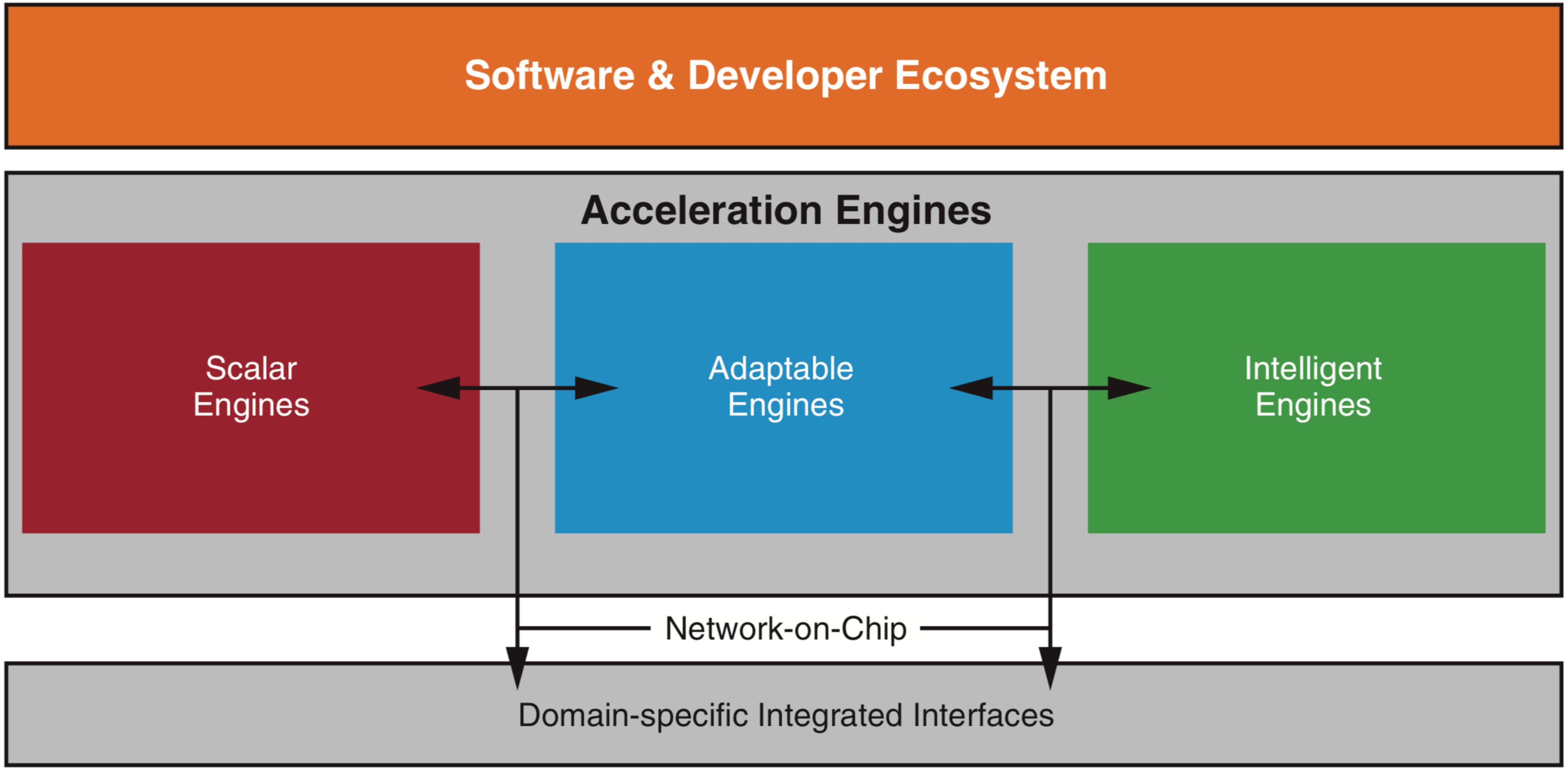


Figure from https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

Xilinx

High level software / hardware view for Versal devices

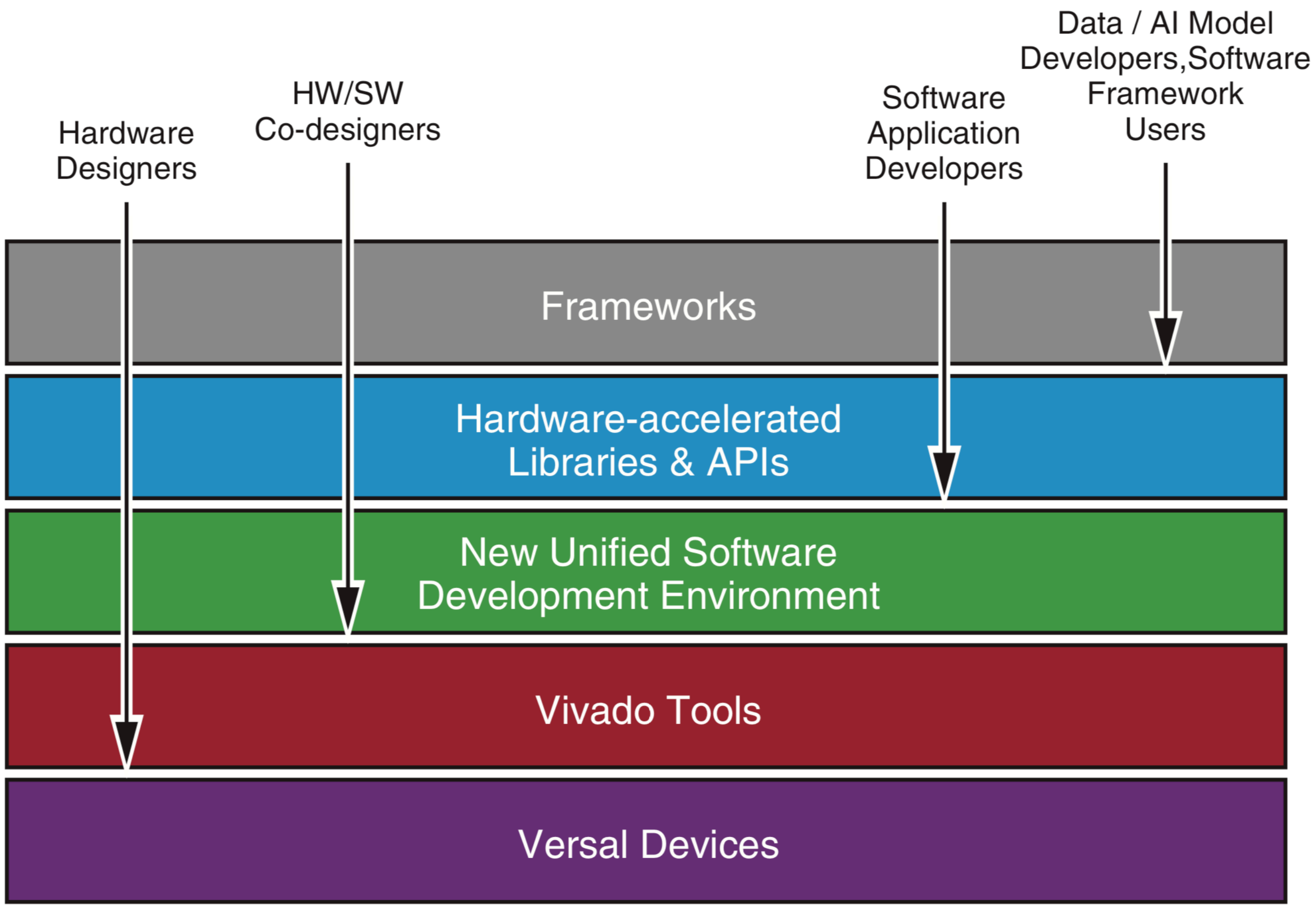


Figure from https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

Backup –
Example Hardware

ARM Reference SoC Diagram

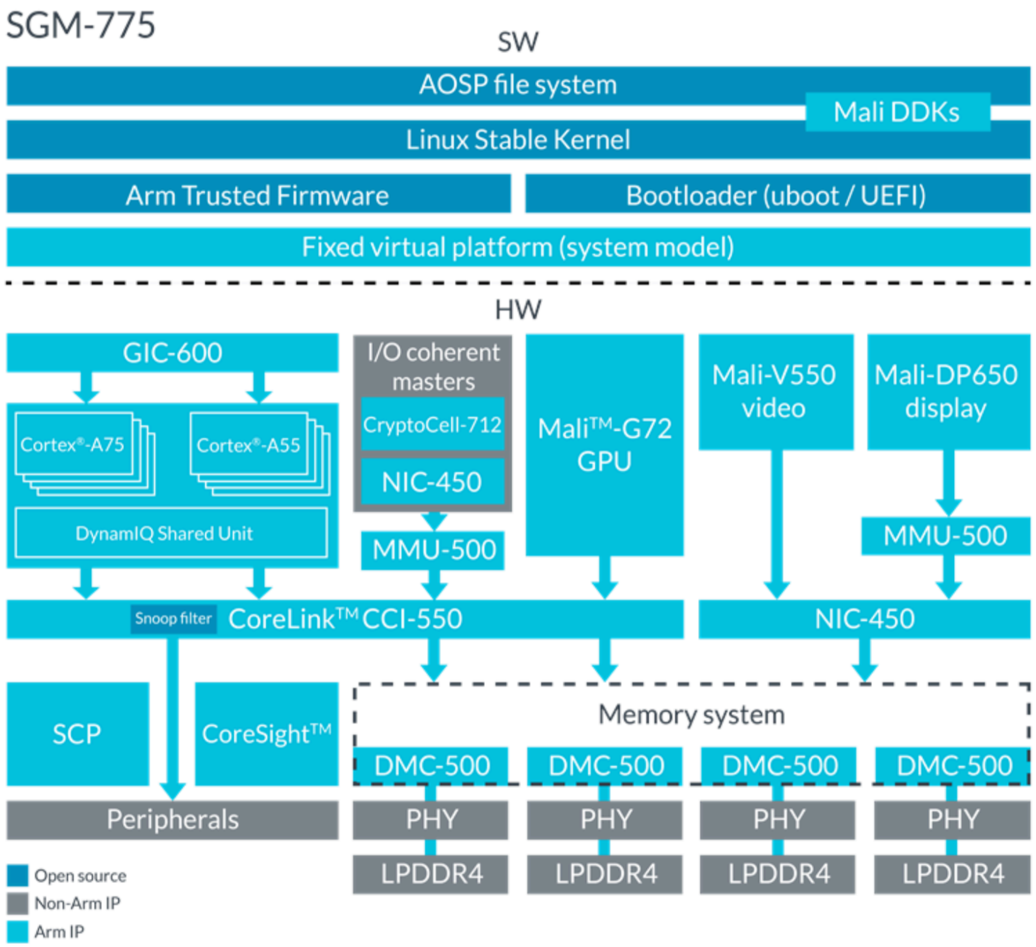


Figure from https://developer.arm.com/products/system-ip

ARM ML Processor

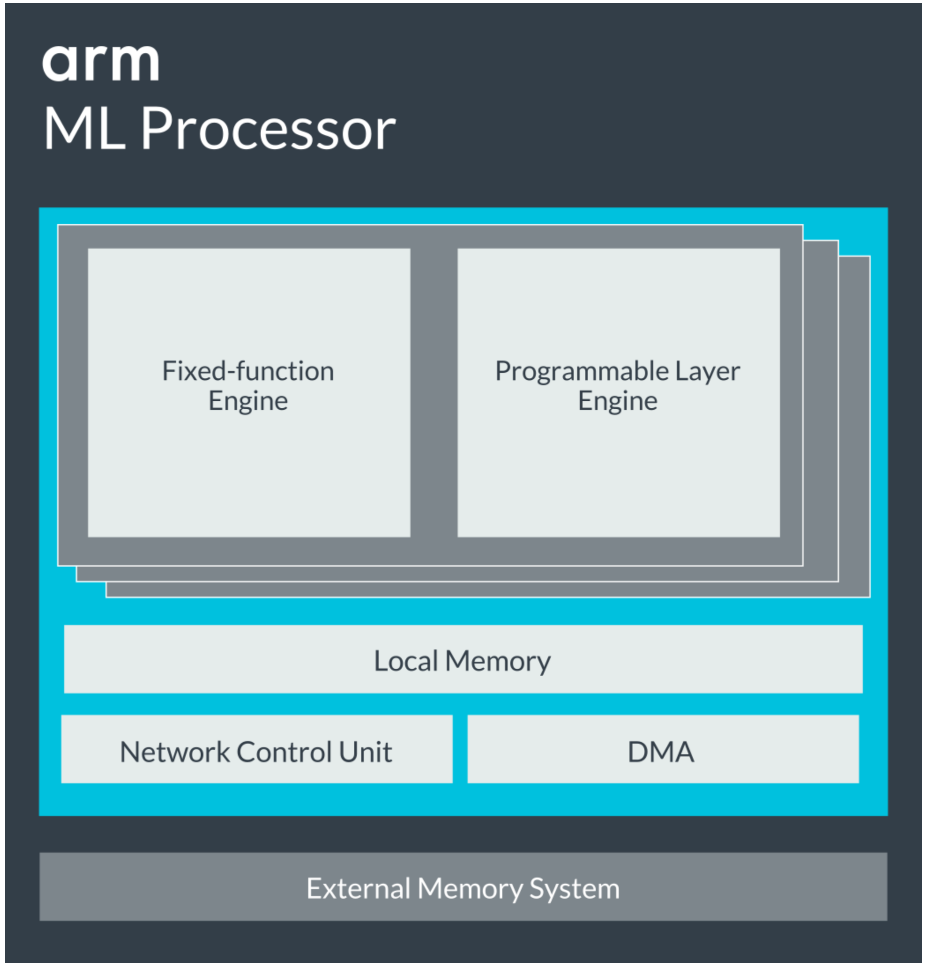
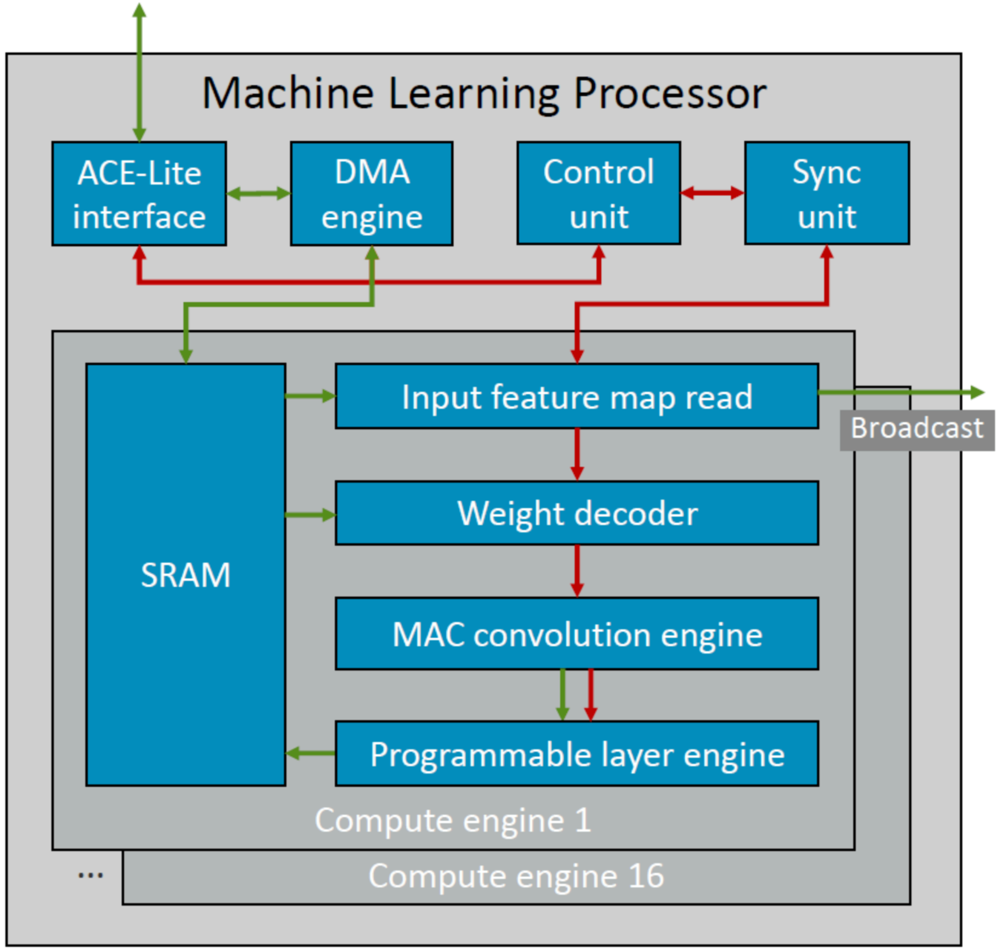


Figure from https://community.arm.com/developer/ip-products/processors/b/ml-ip-blog/posts/arm-ml-processor-exciting-ux-on-edge-devices

ARM ML Processor



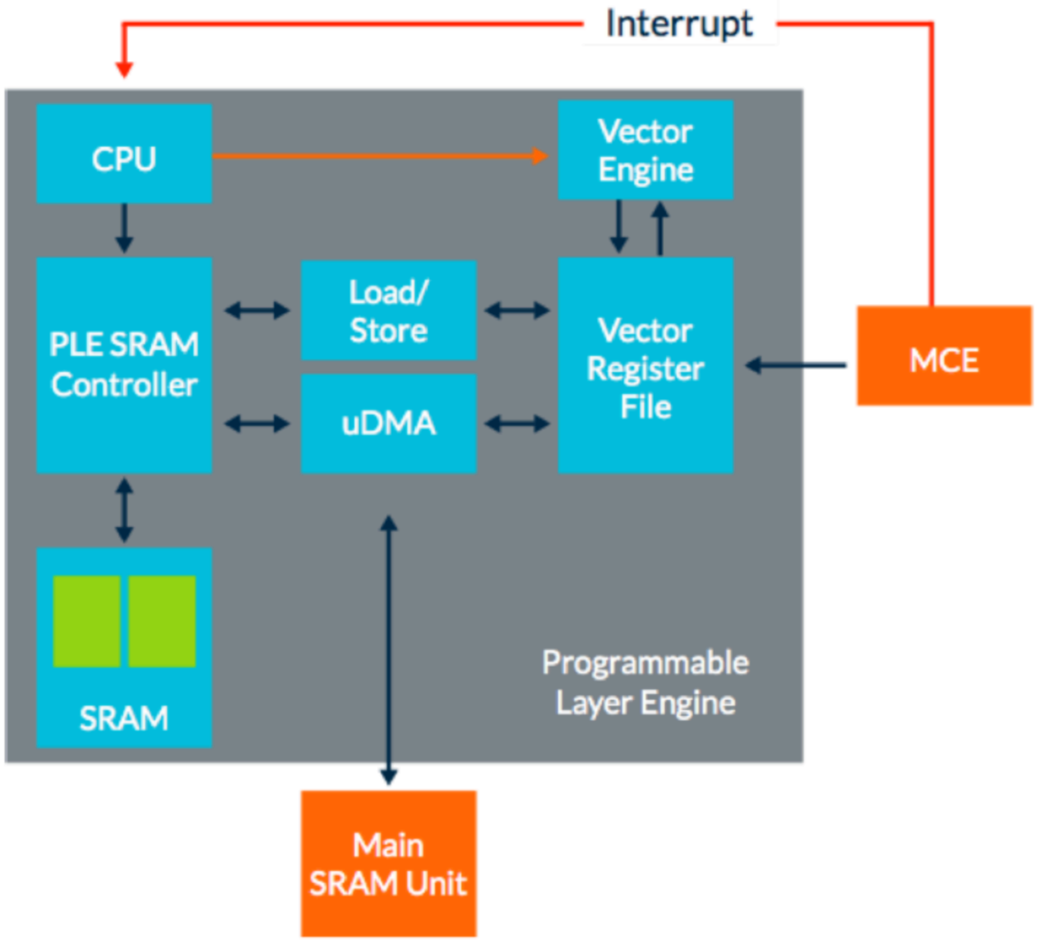


Figure from https://www.anandtech.com/show/12791/arm-details-project-trillium-mlp-architecture

Cadence Tensilica DNA 100

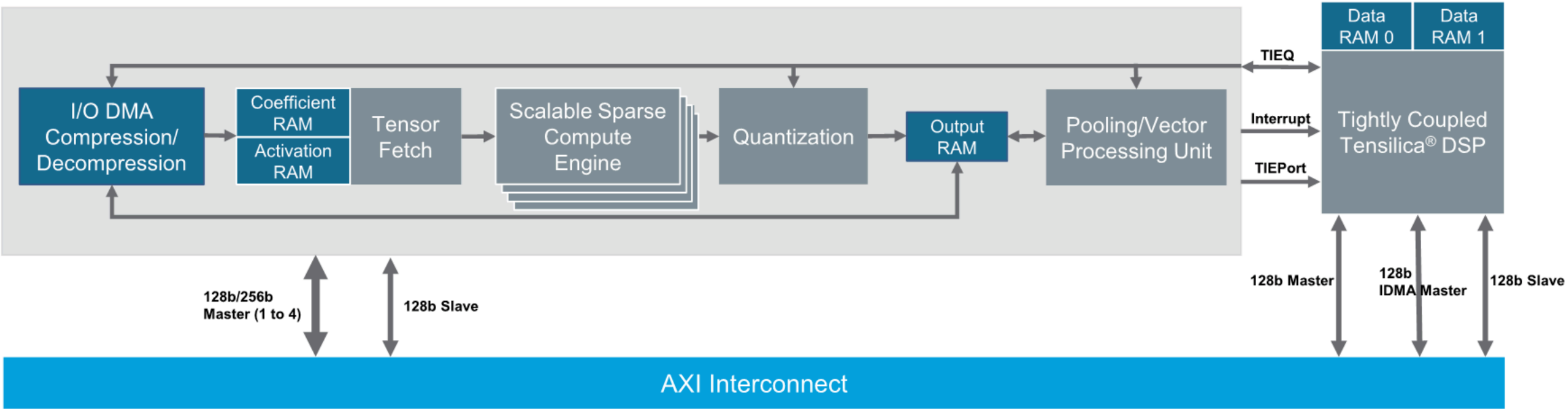


Figure from https://www.anandtech.com/show/13377/cadence-announces-tensilica-dna-100-a-bigger-nn-ip

CEVA CDNN And NeuPro

CEVA-NeuPro HW Block Diagram

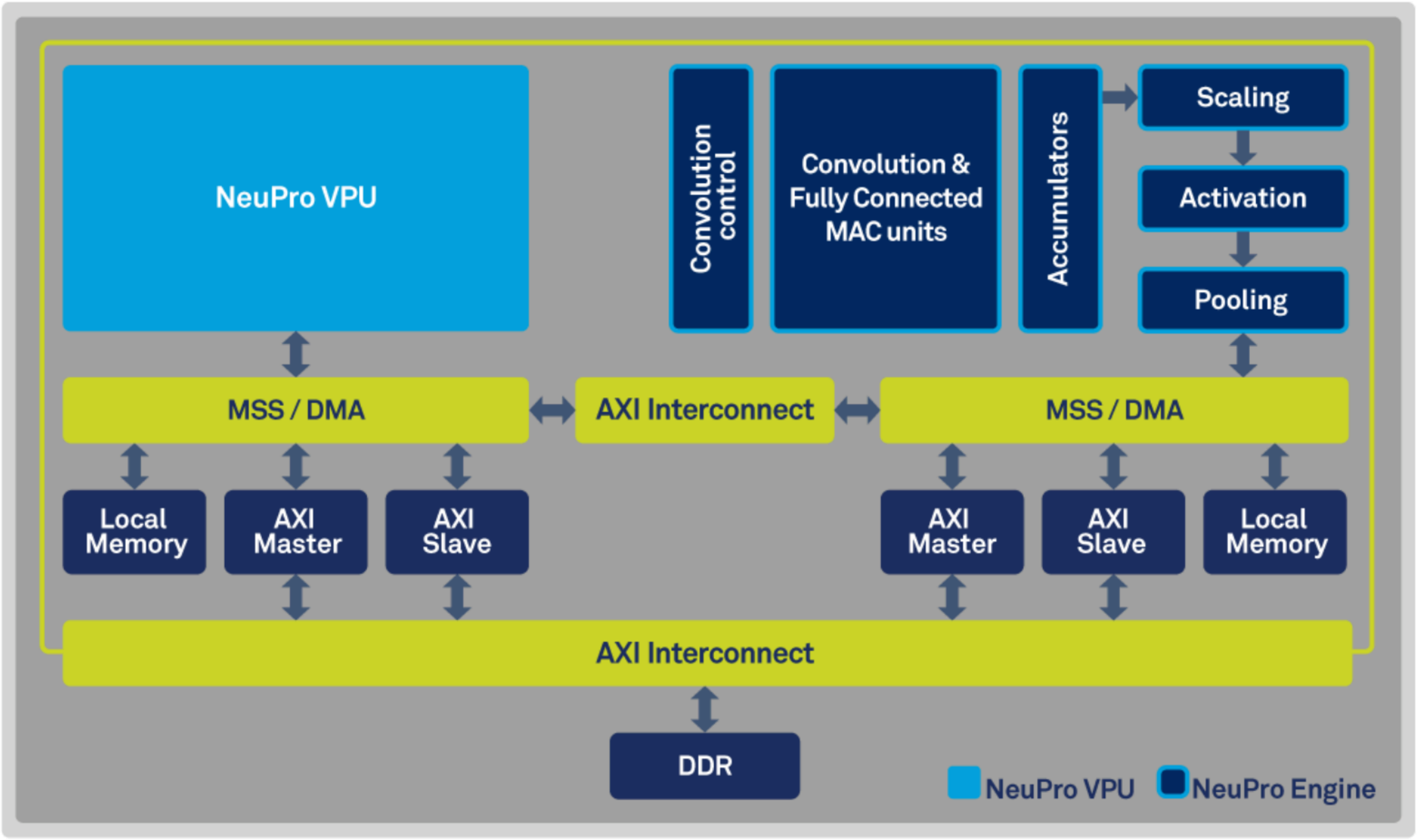


Figure from https://www.ceva-dsp.com/product/ceva-neupro/

Google Coral – Edge TPU

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

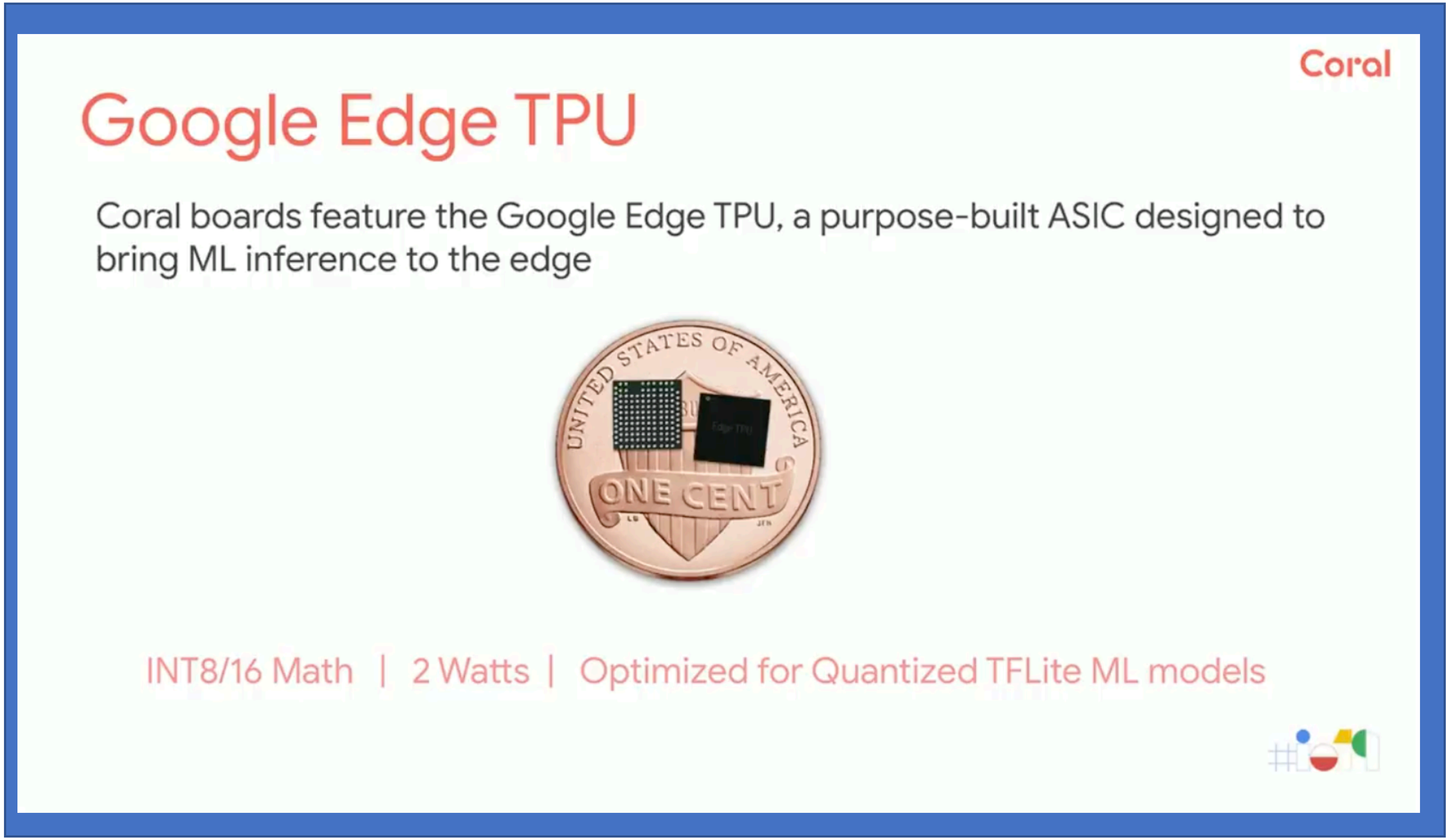


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Edge TPU

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - …

Coral

Edge TPU performance

Edge TPU coprocessor has the capability of running up to 4 Trillion operations per second (TOPS)

Performance comparison in running various ML models for on-device inferencing

Model architecture	Desktop CPU*	Desktop CPU * + USB Accelerator (USB 3.0) *with Edge TPU*	Embedded CPU **	Dev Board † *with Edge TPU*
MobileNet v1	47 ms	2.2 ms	179 ms	2.2 ms
MobileNet v2	45 ms	2.3 ms	150 ms	2.5 ms
Inception v1	92 ms	3.6 ms	406 ms	3.9 ms
Inception v4	792 ms	100 ms	3,463 ms	100 ms

** Desktop CPU: 64-bit Intel(R) Xeon(R) E5-1650 v4 @ 3.60GHz ** Embedded CPU: Quad-core Cortex-A53 @ 1.5GHz*
† Dev Board: Quad-core Cortex-A53 @ 1.5GHz + Edge TPU

Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Dev Board

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

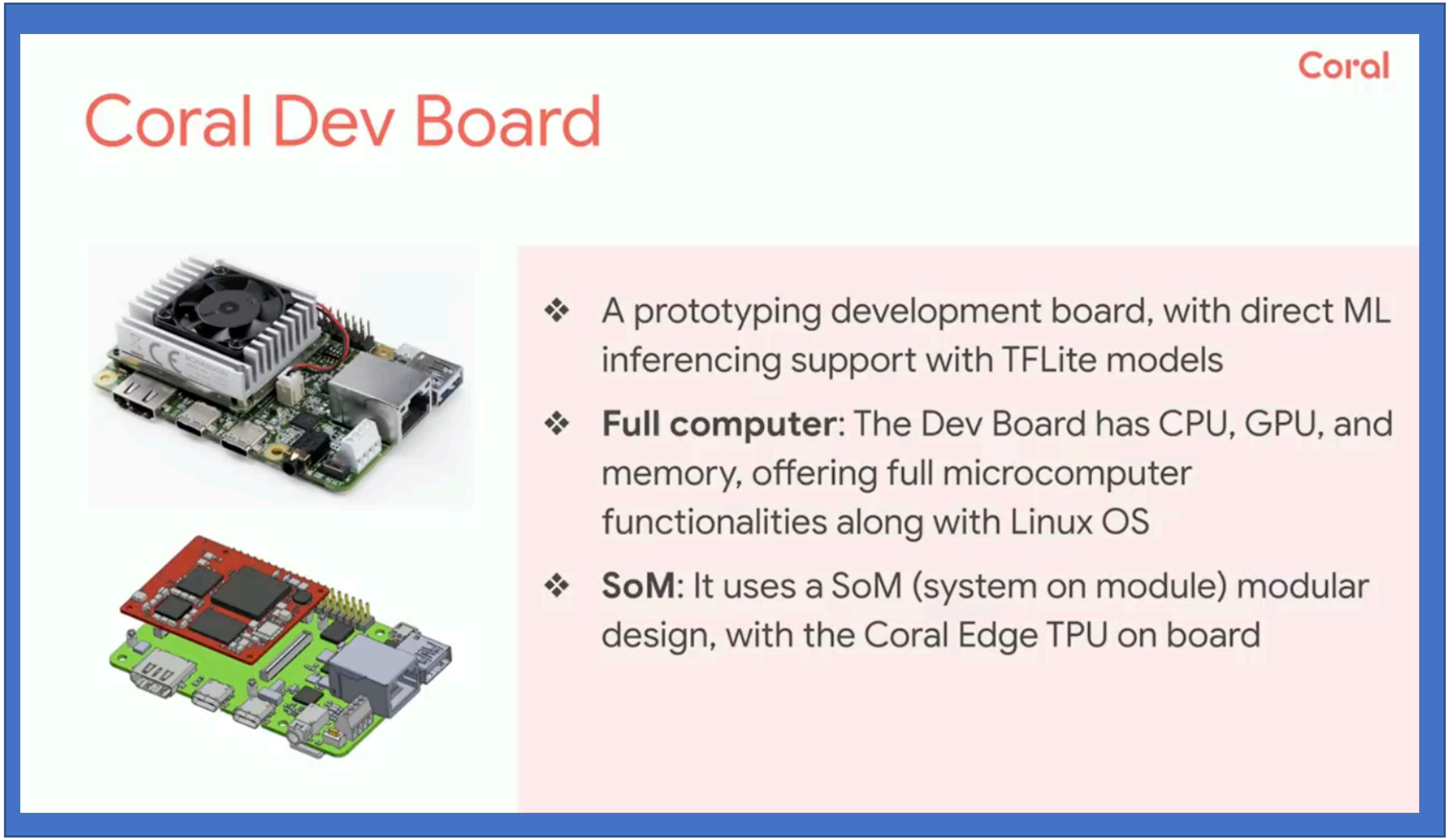


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Dev Board

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

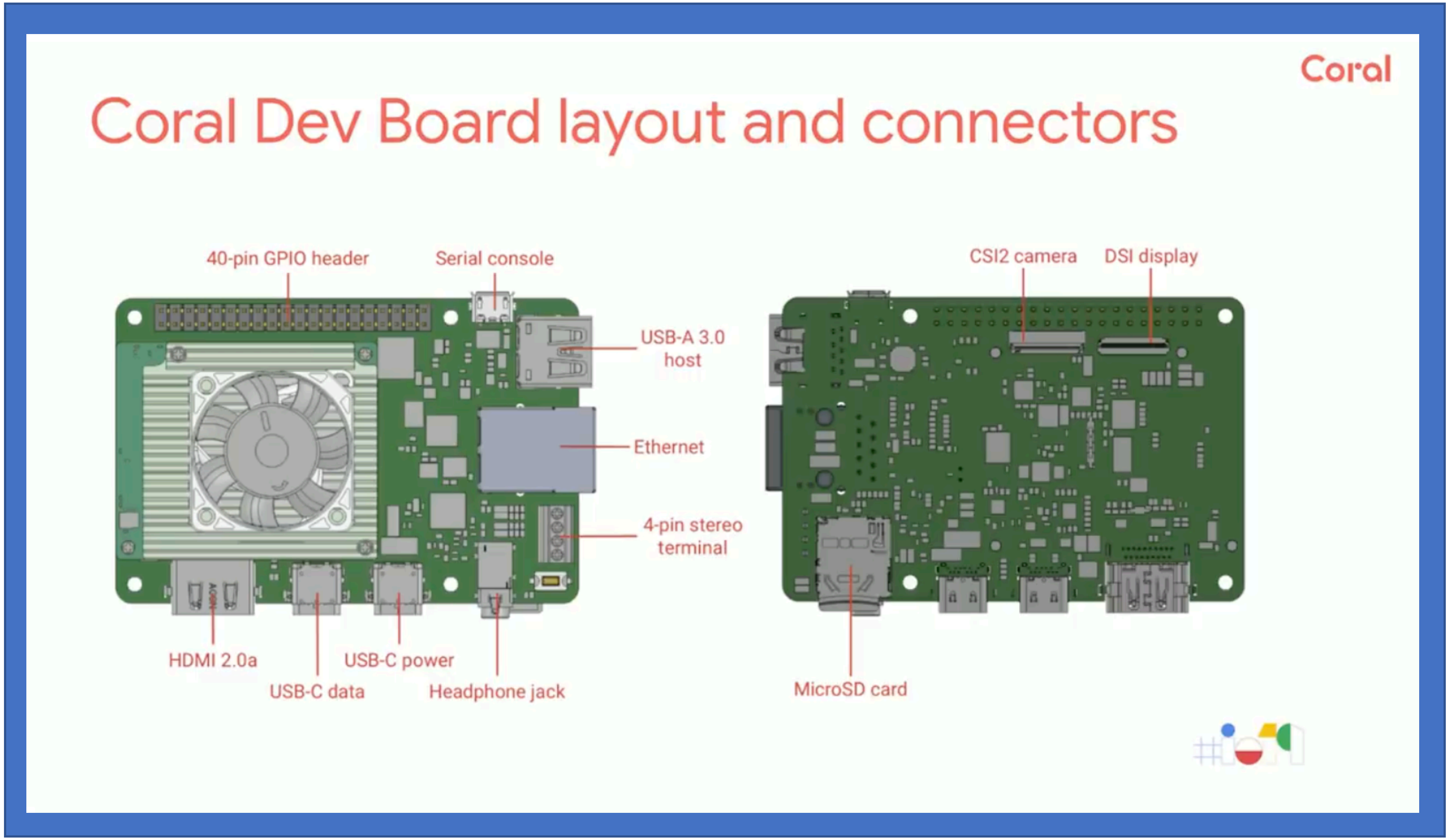


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Dev Board

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

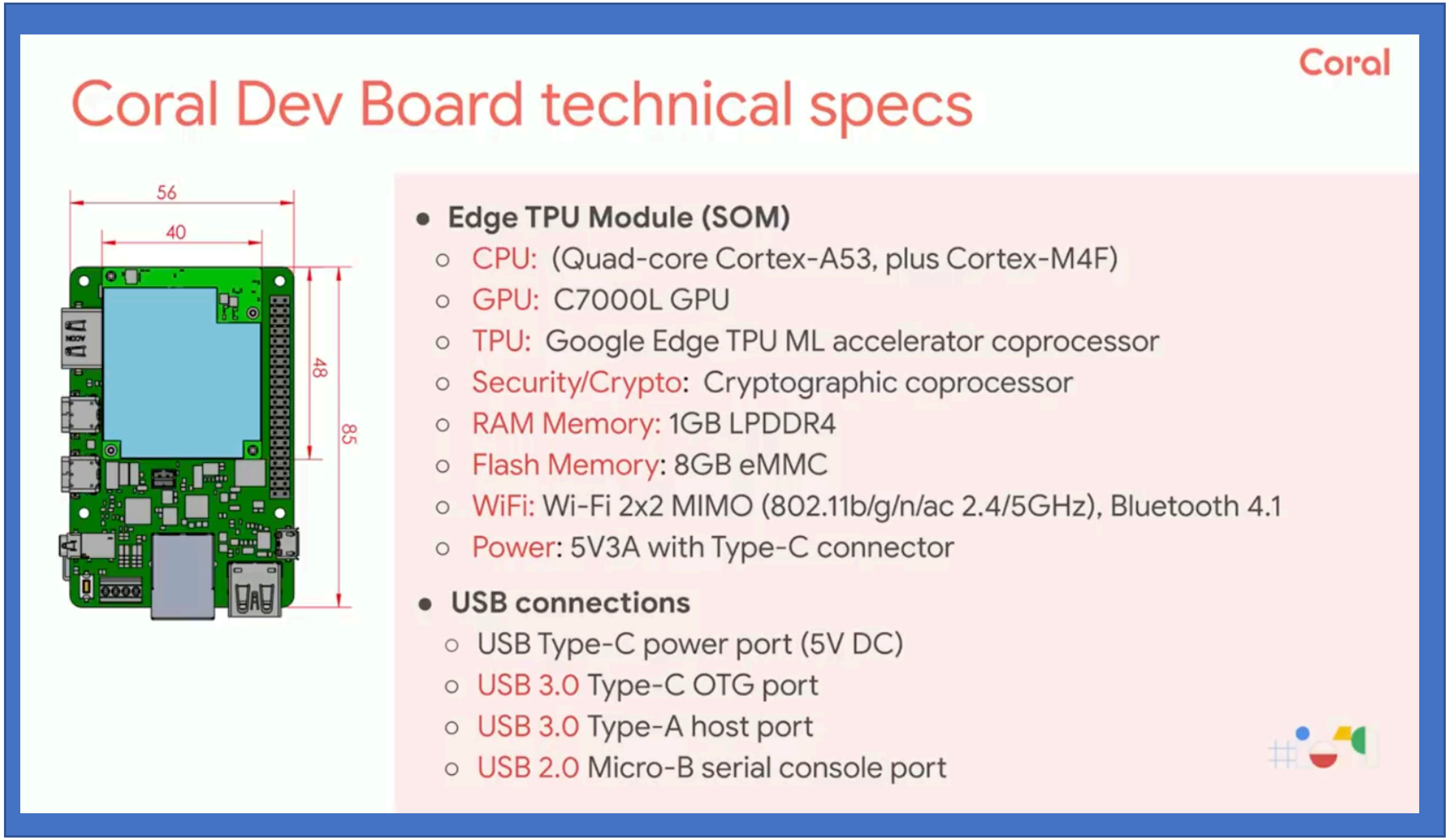


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Dev Board

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

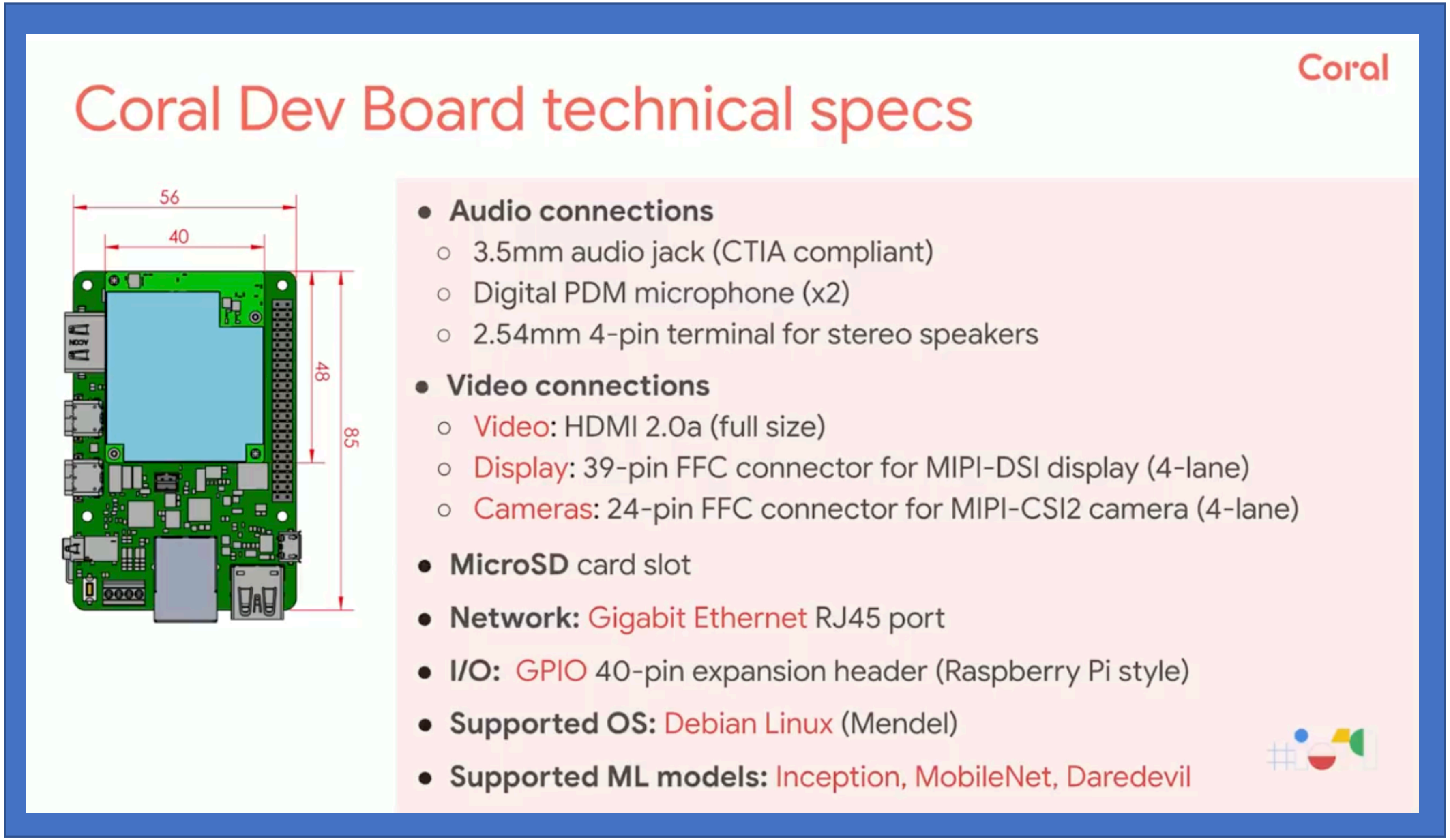


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – Dev Board

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

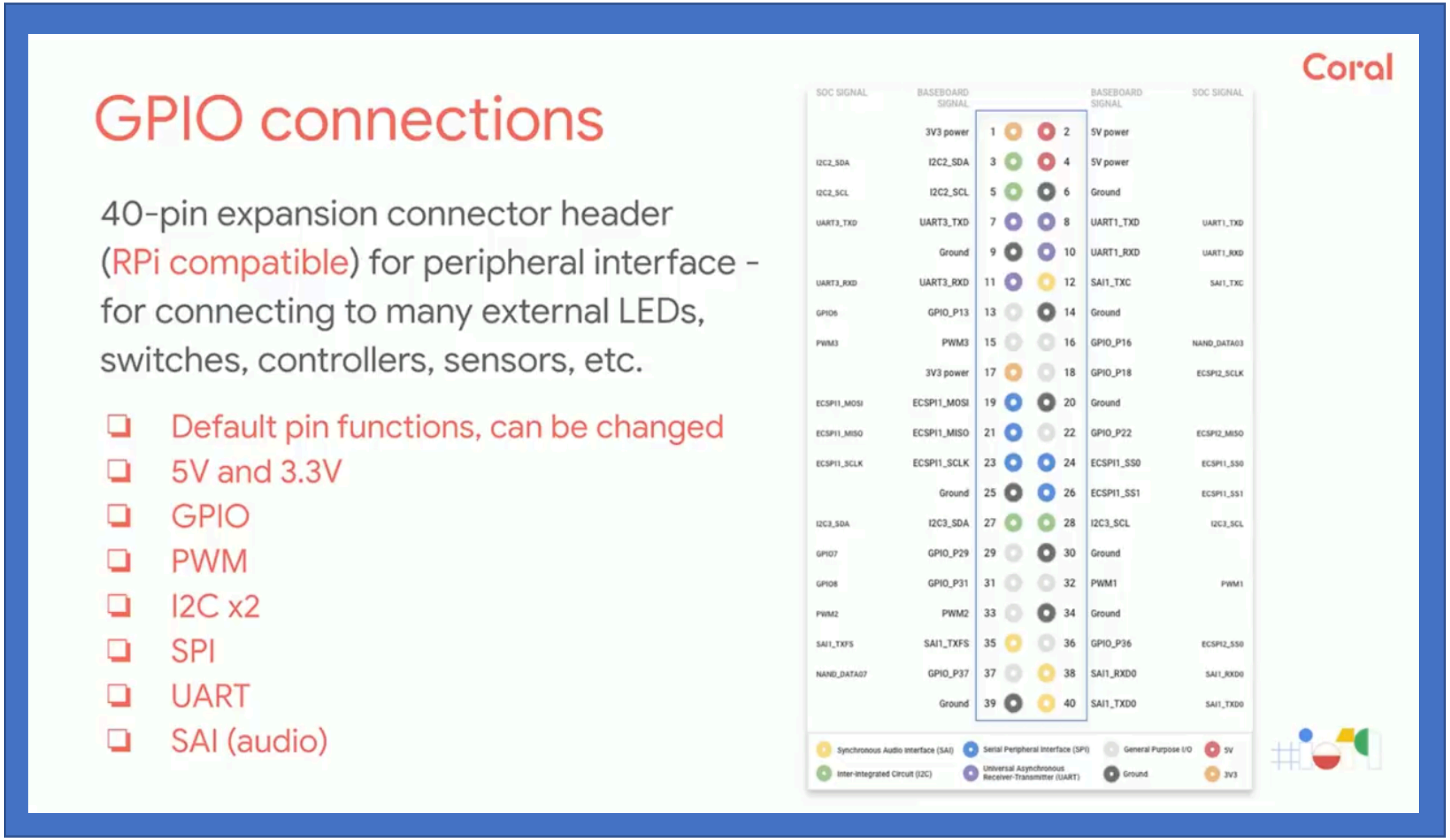
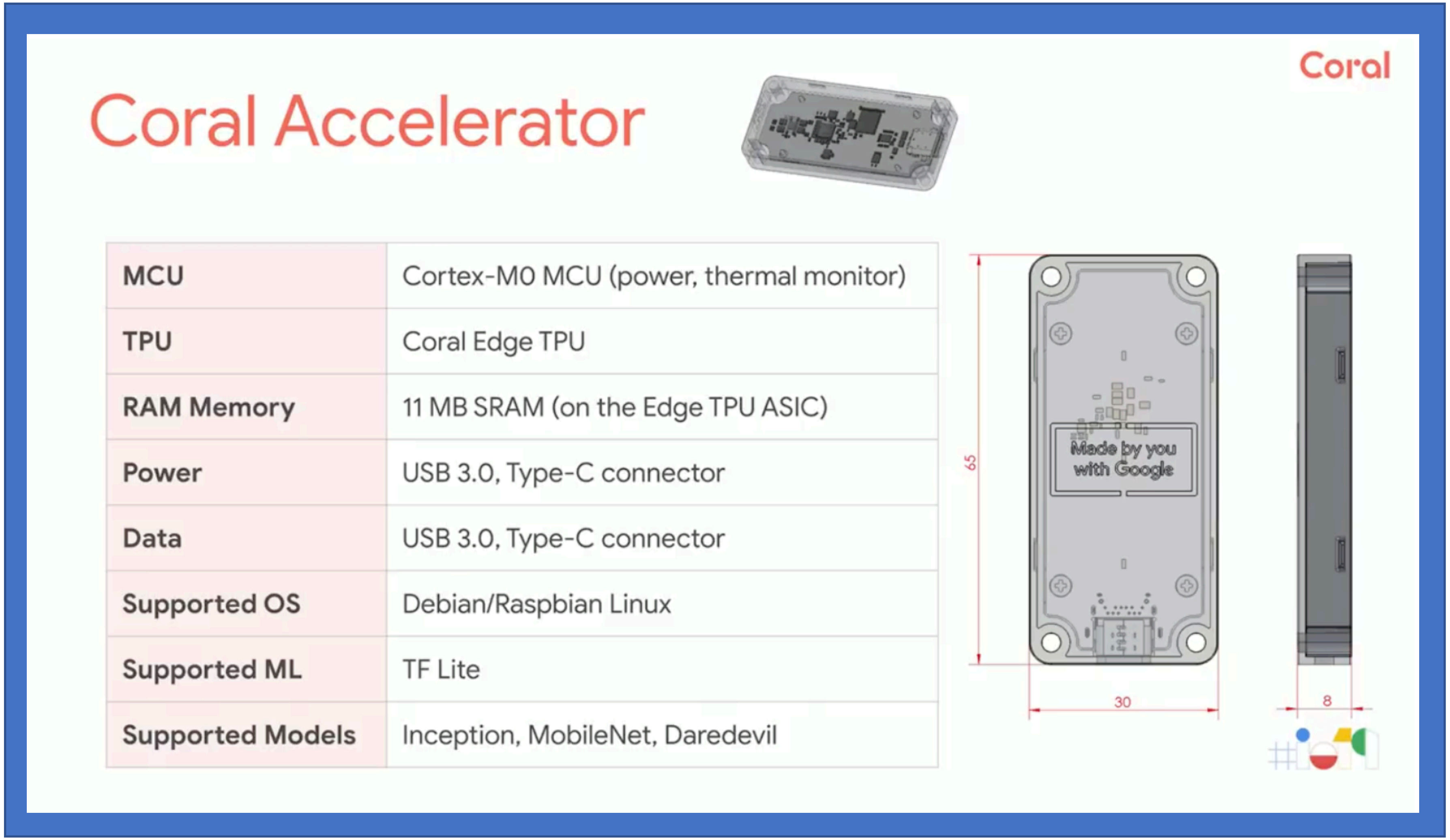


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – USB Stick

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...



MCU	Cortex-M0 MCU (power, thermal monitor)
TPU	Coral Edge TPU
RAM Memory	11 MB SRAM (on the Edge TPU ASIC)
Power	USB 3.0, Type-C connector
Data	USB 3.0, Type-C connector
Supported OS	Debian/Raspbian Linux
Supported ML	TF Lite
Supported Models	Inception, MobileNet, Daredevil

Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google Coral – USB Stick

- Edge TPU module
- Compute
  - CPU: 4x ARM A53
  - MCU: 1x ARM M4F
  - GPU: C7000L
  - TPU: Google Edge 4 TOPS (2W for the Edge TPU chip)
- Memory
  - DRAM: 1GB LPDDR4
  - Flash: 8GB eMMC
- I/O
  - WiFi 2x2 MIMO 802.11 b/g/n/ac 2.4/5 GHz
  - Bluetooth 4.1
  - Gigabit Ethernet
  - USB 3.0 type A and C
  - USB 2.0 micro B
  - Audio: 3.5 mm and digital PDM
  - Video: HDMI 2.0a
  - Display: MIPI-DSI 4 lane
  - Camera: MIPI-CSI2 4 lane
  - ...

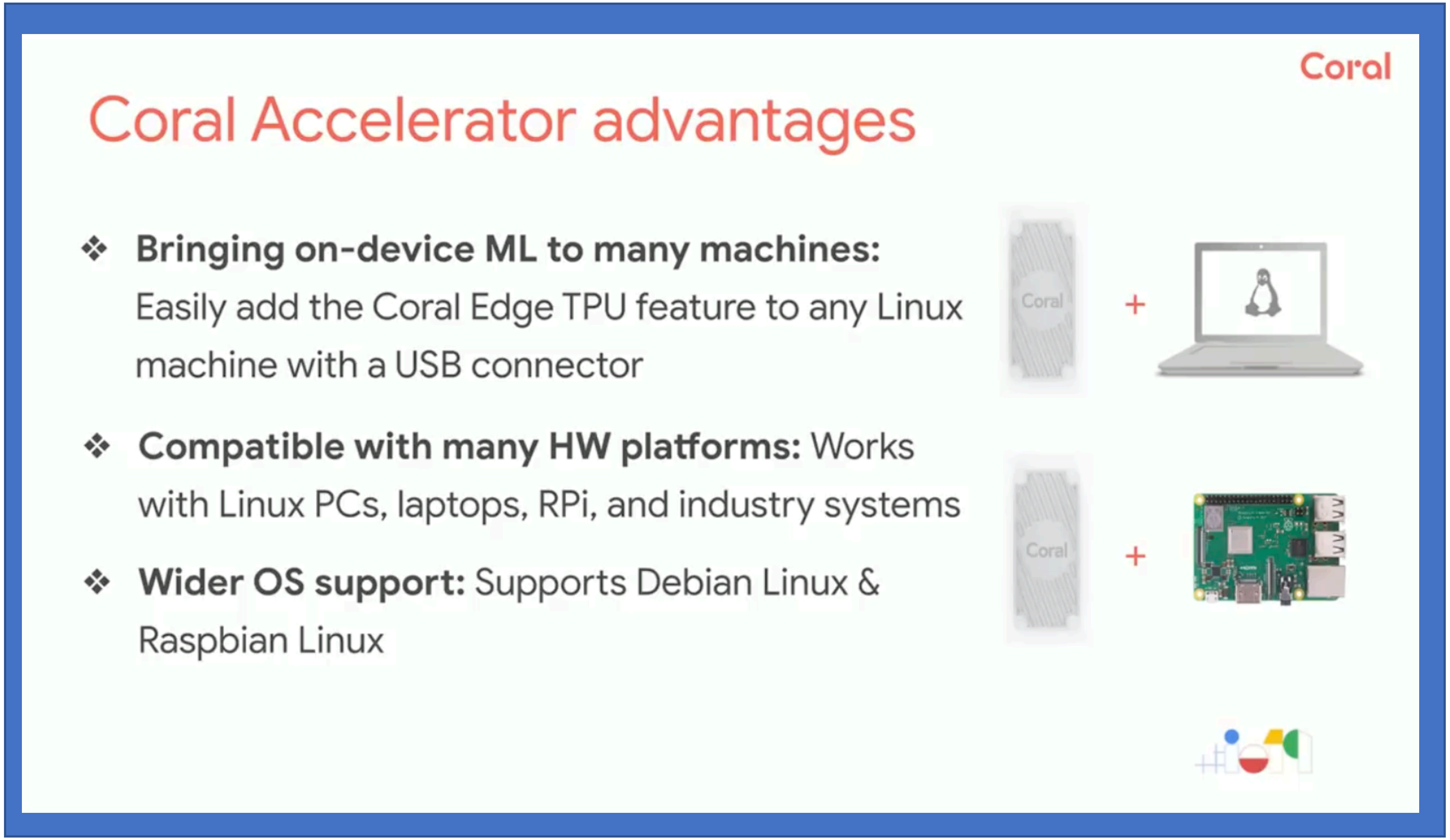


Figure from https://www.youtube.com/watch?v=Jgm25QdF90A

Google TPU V1

- Notes
  - Focused on inference
  - 256 x 256 matrix multiplier with 8b int operands running at 700 MHz for ~ 91.75 TOPS performance
  - 30 GB/s of memory bandwidth
  - 36? MB on device memory

- Links
  - https://arxiv.org/abs/1704.04760

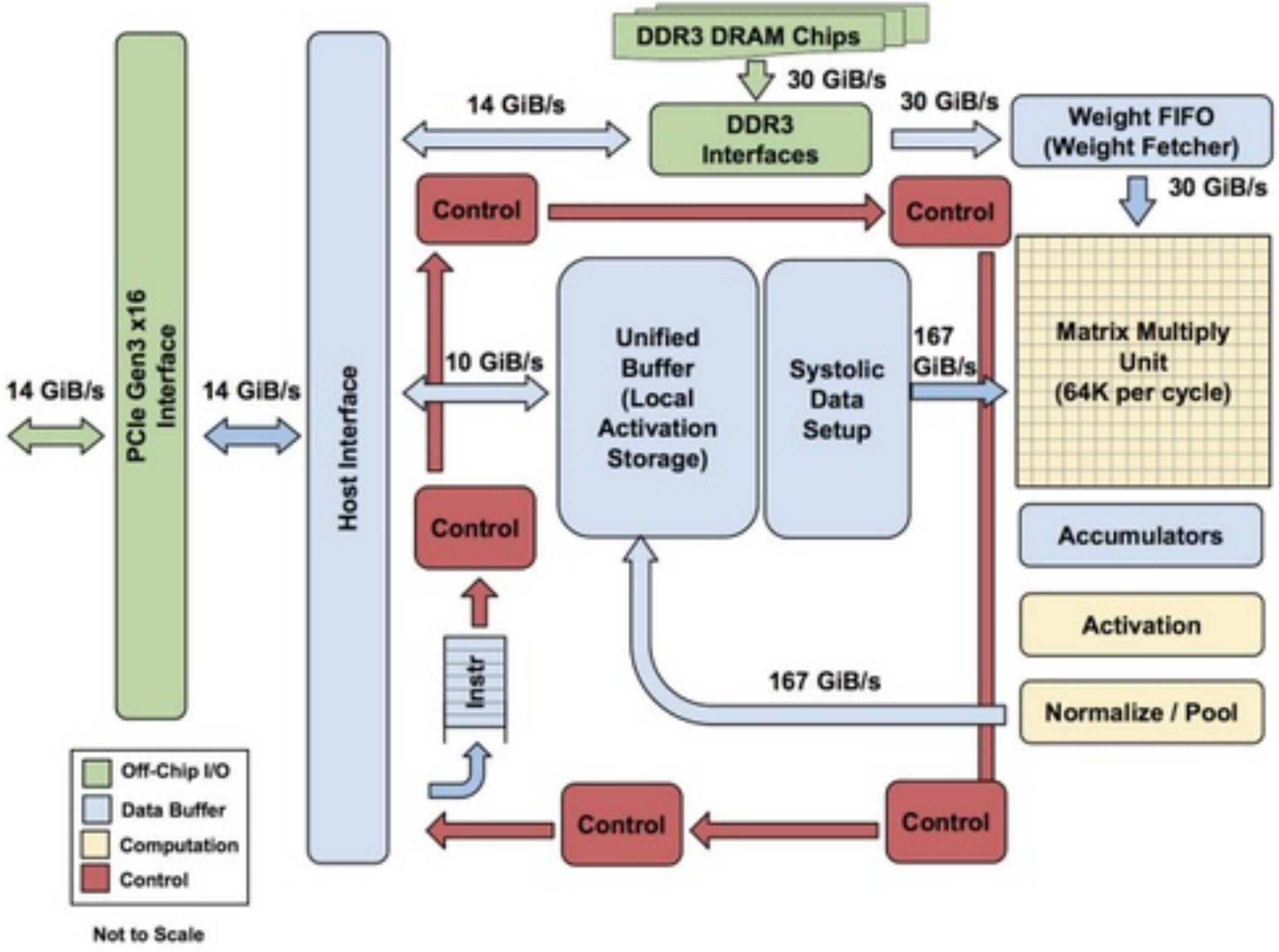


Figure from https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/

Google TPU V2

- Notes
  - Focused on training and inference
  - Each chip contains 2 cores
  - Each core contains a 128 x 128 matrix multiplier with 16 bfloat operands and 32b float accumulation running at ~ 1.375 GHz for ~ 22.5 TFLOPS performance along with scalar and vector compute
  - Each core has access to 8 GB of HBM with 300 GB/s of memory bandwidth
  - Total per chip of 45 TFLOPS and 16 GB HBM with 600 GB/s of memory bandwidth
  - 4 chips per board
  - 64 boards per TPU pod
  - Likely connected in a 2D torus
- Links
  - http://learningsys.org/nips17/assets/slides/dean-nips17.pdf
  - https://www.theregister.co.uk/2017/12/14/google_tpu2_specs_ish/
  - https://www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/

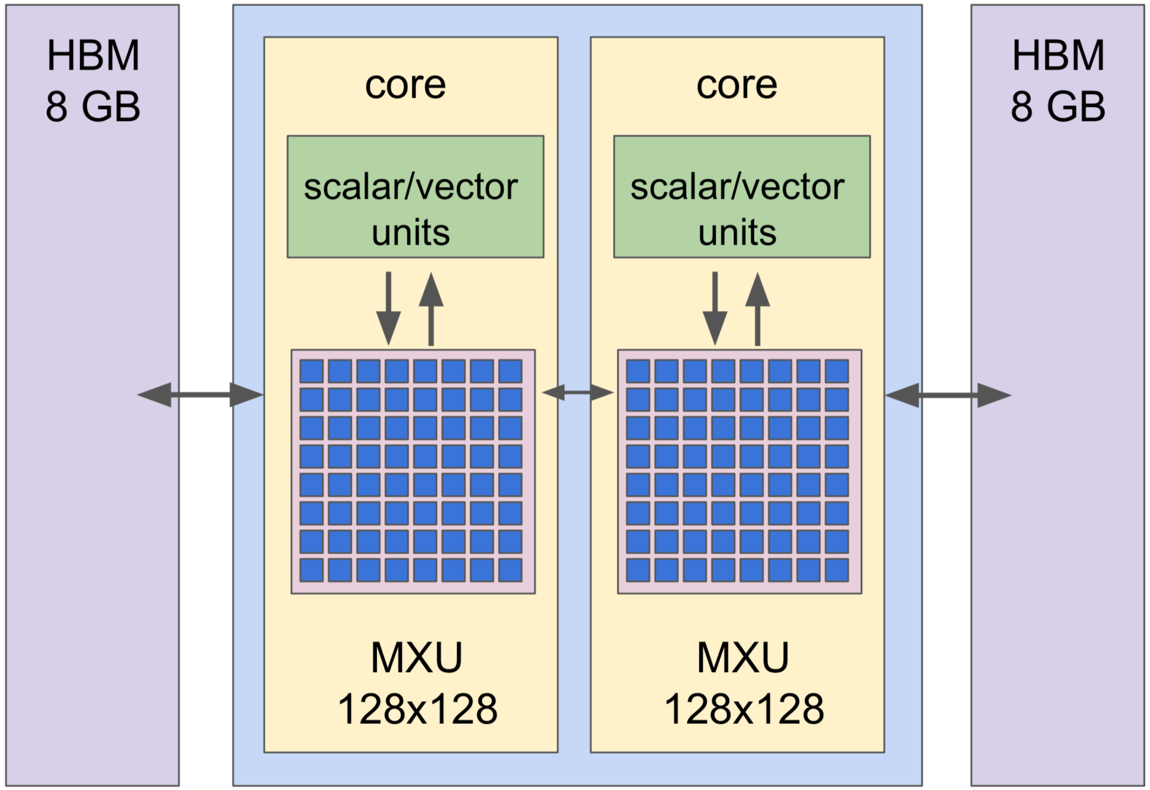
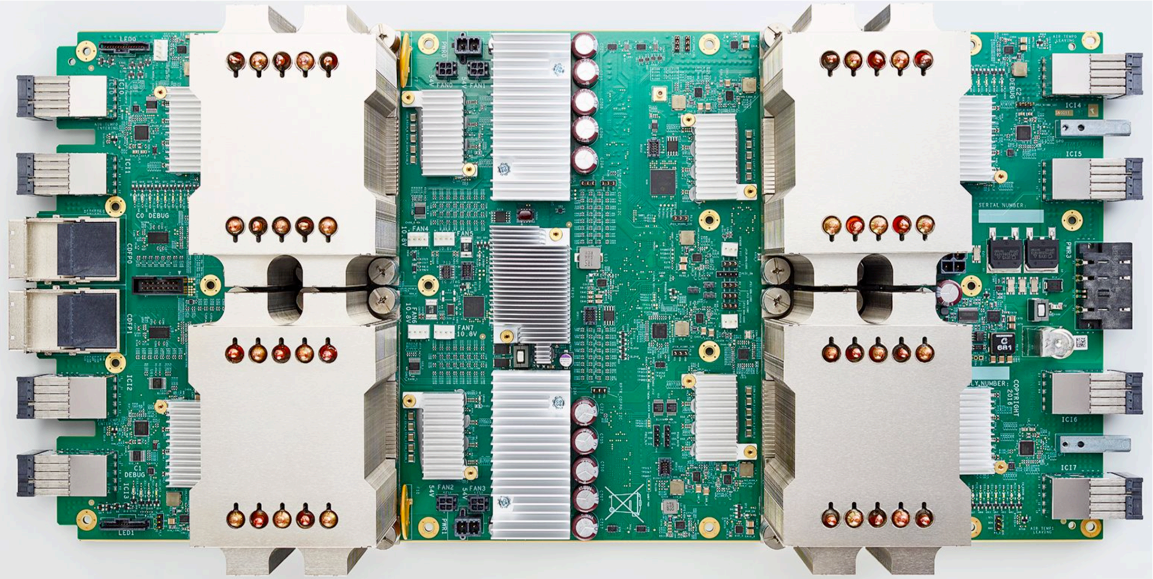


Figure from http://learningsys.org/nips17/assets/slides/dean-nips17.pdf

Google TPU V2

- Notes
  - Focused on training and inference
  - Each chip contains 2 cores
  - Each core contains a 128 x 128 matrix multiplier with 16 bfloat operands and 32b float accumulation running at ~ 1.375 GHz for ~ 22.5 TFLOPS performance along with scalar and vector compute
  - Each core has access to 8 GB of HBM with 300 GB/s of memory bandwidth
  - Total per chip of 45 TFLOPS and 16 GB HBM with 600 GB/s of memory bandwidth
  - 4 chips per board
  - 64 boards per TPU pod
  - Likely connected in a 2D torus



- Links
  - http://learningsys.org/nips17/assets/slides/dean-nips17.pdf
  - https://www.theregister.co.uk/2017/12/14/google_tpu2_specs_ish/
  - https://www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/

Figure from http://learningsys.org/nips17/assets/slides/dean-nips17.pdf

Google TPU V2

- Notes
  - Focused on training and inference
  - Each chip contains 2 cores
  - Each core contains a 128 x 128 matrix multiplier with 16 bfloat operands and 32b float accumulation running at ~ 1.375 GHz for ~ 22.5 TFLOPS performance along with scalar and vector compute
  - Each core has access to 8 GB of HBM with 300 GB/s of memory bandwidth
  - Total per chip of 45 TFLOPS and 16 GB HBM with 600 GB/s of memory bandwidth
  - 4 chips per board
  - 64 boards per TPU pod
  - Likely connected in a 2D torus

- Links
  - http://learningsys.org/nips17/assets/slides/dean-nips17.pdf
  - https://www.theregister.co.uk/2017/12/14/google_tpu2_specs_ish/
  - https://www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/

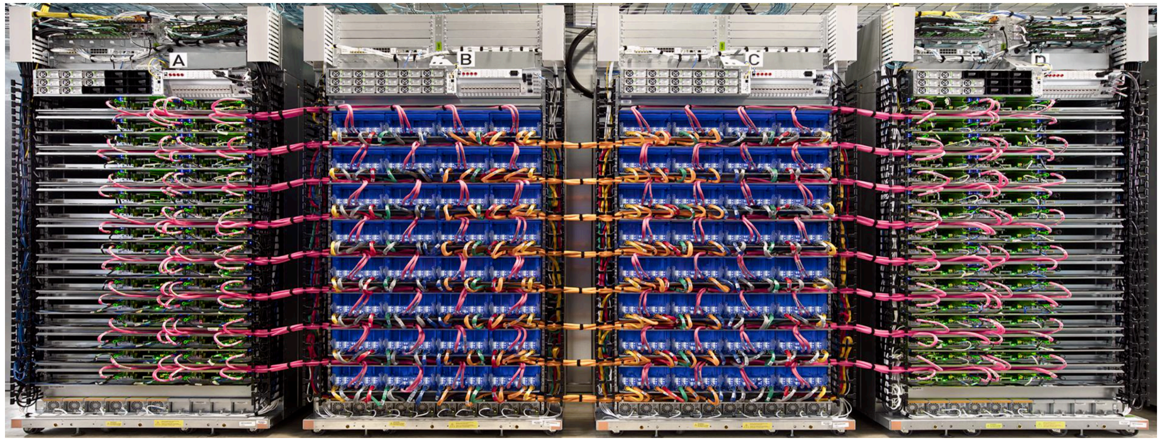


Figure from http://learningsys.org/nips17/assets/slides/dean-nips17.pdf

Google TPU V3

- Notes
  - Each chip contains 2 cores, each core contains 2x 128x128 matrix multipliers with a set of shared scalar and vector units
  - Each core has access to 16 GB of HBM
  - So effectively 2x the compute and 2x the memory vs V2
  - Additional modifications made to the pod configuration and connectivity are also likely
- Links
  - https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/
  - https://pliss2019.github.io/albert_cohen_slides.pdf

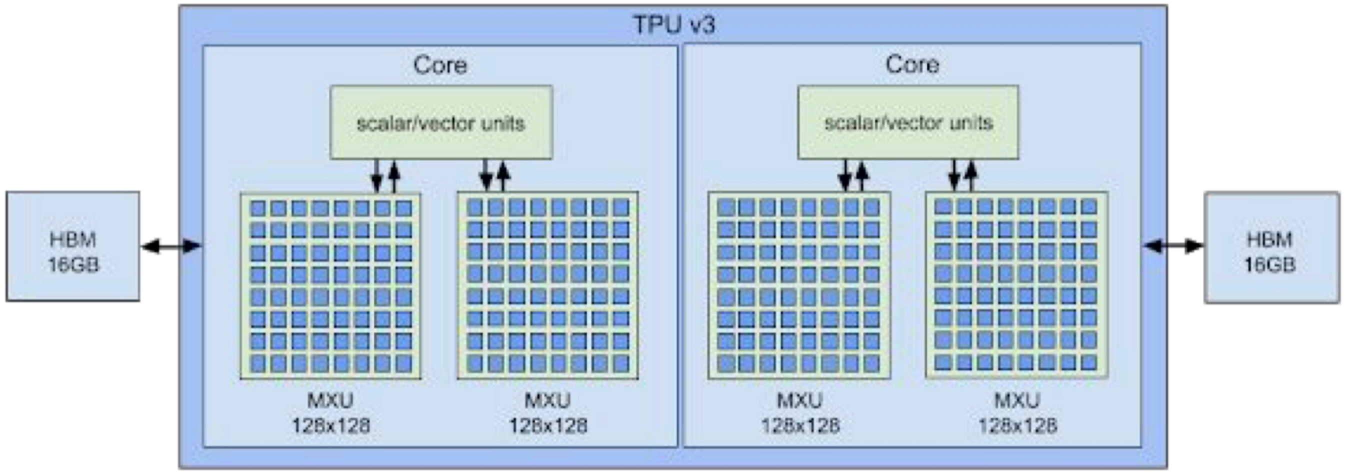
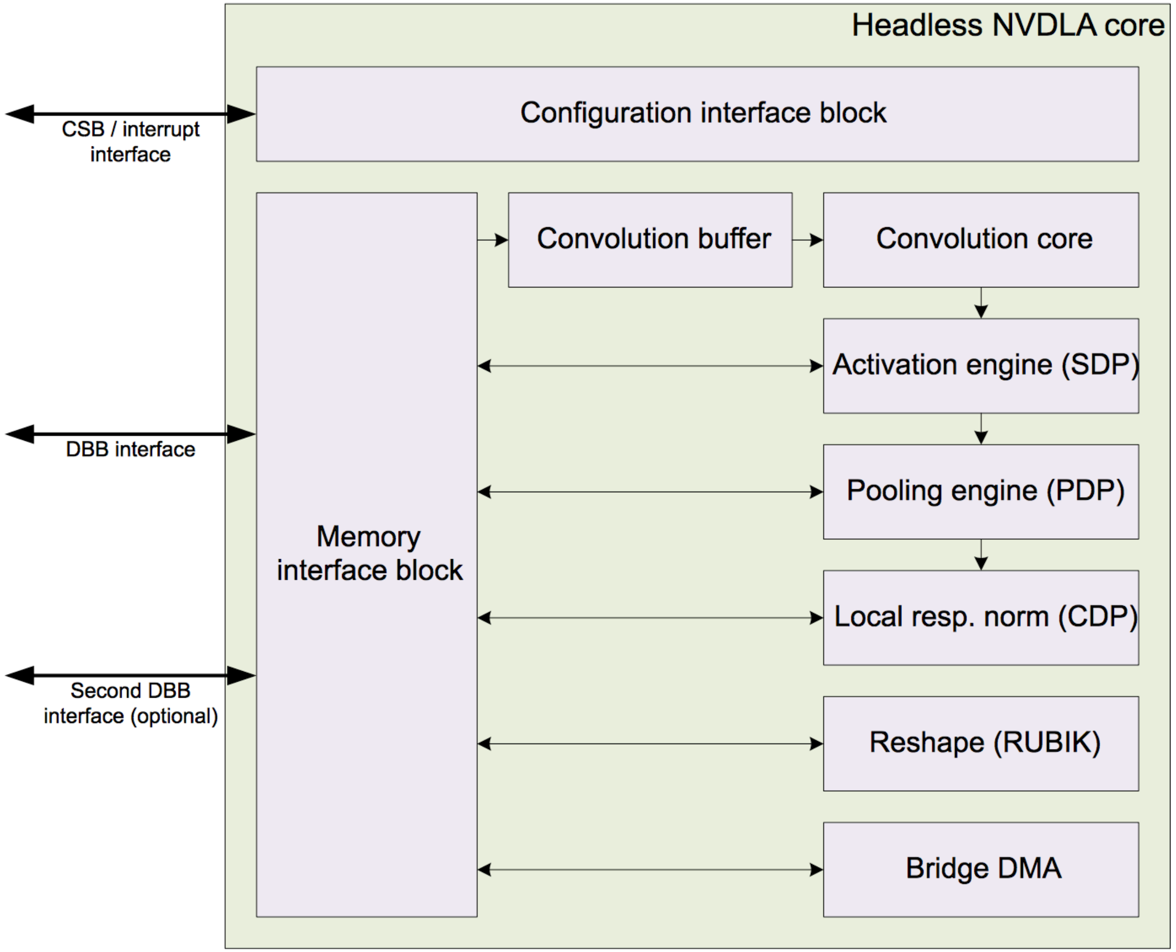


Figure from https://pliss2019.github.io/albert_cohen_slides.pdf

Nvidia NVDLA



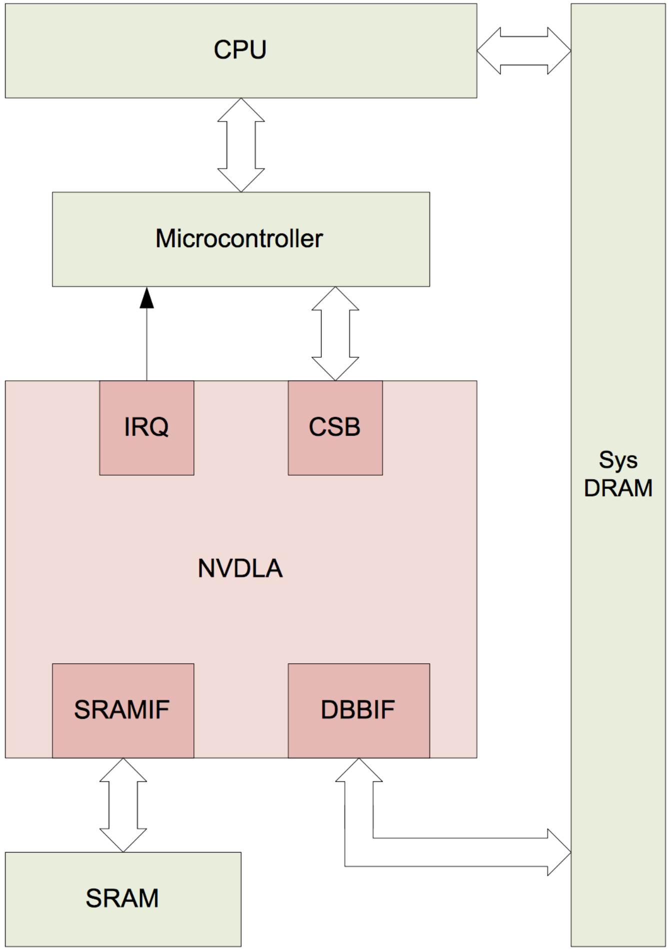
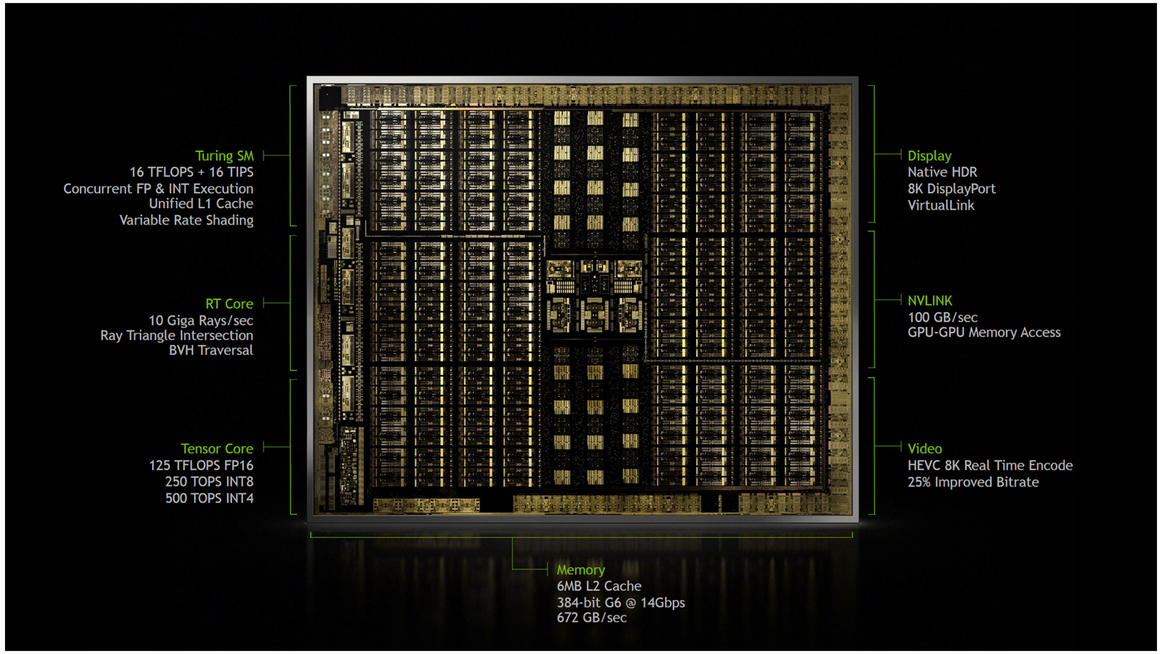


Figure from http://nvdla.org/primer.html

Nvidia Titan RTX

- Partial specs
  - 24 GB VRAM connected via GDDR6 at 672 GB/s
  - Device clock 1.350 / 1.770 GHz
  - 6 MB L2 cache
  - 576 tensor cores with 125 TFLOPS FP16 (FP32 accumulation), 250 TOPS INT8, 500 TOPS INT4
  - 16.3 TFLOPS FP32 and 0.51 FP64
  - 18.6 B transistors and 280 W power in 12 nm FFN



- Links
  - https://www.anandtech.com/show/13668/nvidia-unveils-rtx-titan-2500-top-turing

Figure from https://www.anandtech.com/show/13668/nvidia-unveils-rtx-titan-2500-top-turing

Nvidia Turing

Figure from https://www.anandtech.com/print/13282/nvidia-turing-architecture-deep-dive

Nvidia Turing

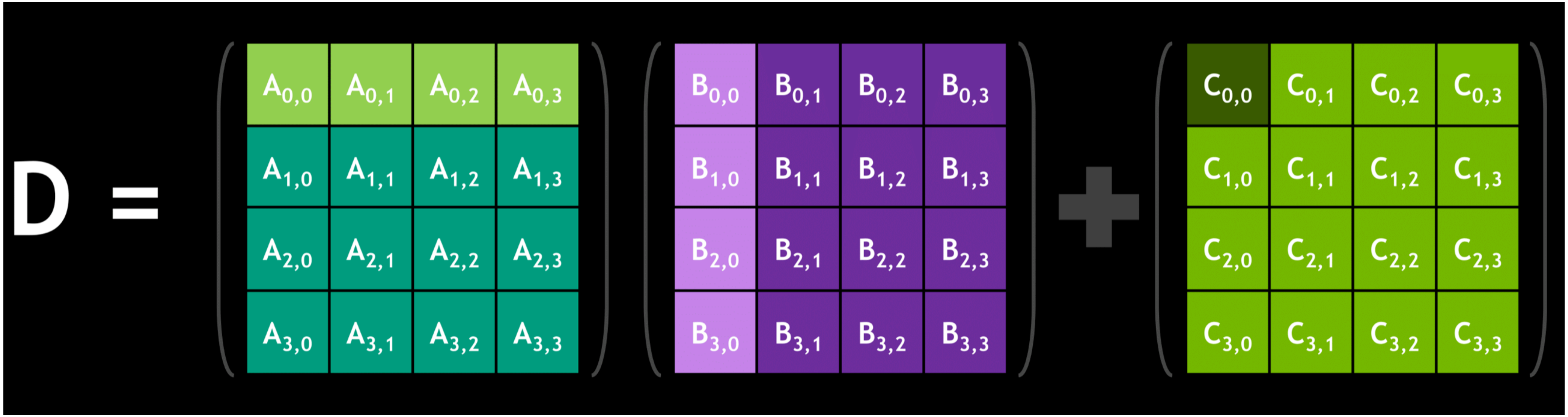


Figure from https://www.anandtech.com/print/13282/nvidia-turing-architecture-deep-dive

Nvidia Turing

NVIDIA GeForce x80 Ti Specification Comparison				
	RTX 2080 Ti Founder's Edition	RTX 2080 Ti	GTX 1080 Ti	GTX 980 Ti
CUDA Cores	4352	4352	3584	2816
ROPs	88	88	88	96
Core Clock	1350MHz	1350MHz	1481MHz	1000MHz
Boost Clock	1635MHz	1545MHz	1582MHz	1075MHz
Memory Clock	14Gbps GDDR6	14Gbps GDDR6	11Gbps GDDR5X	7Gbps GDDR5
Memory Bus Width	352-bit	352-bit	352-bit	384-bit
VRAM	11GB	11GB	11GB	6GB
Single Precision Perf.	14.2 TFLOPs	13.4 TFLOPs	11.3 TFLOPs	6.1 TFLOPs
"RTX-OPS"	78T	78T	N/A	N/A
TDP	260W	250W	250W	250W
GPU	TU102	TU102	GP102	GM200
Architecture	Turing	Turing	Pascal	Maxwell
Manufacturing Process	TSMC 12nm "FFN"	TSMC 12nm "FFN"	TSMC 16nm	TSMC 28nm
Launch Date	09/20/2018	09/20/2018	03/10/2017	06/01/2015
Launch Price	$1199	$999	MSRP: $699 Founders: $699	$649

NVIDIA Turing GPU Comparison				
	TU102	TU104	TU106	GP102
CUDA Cores	4608	3072	2304	3840
SMs	72	48	36	30
Texture Units	288	192	144	240
RT Cores	72	48	36	N/A
Tensor Cores	576	384	288	N/A
ROPs	96	64	64	96
Memory Bus Width	384-bit	256-bit	256-bit	384-bit
L2 Cache	6MB	4MB	4MB	3MB
Register File (Total)	18MB	12MB	9MB	7.5MB
Architecture	Turing	Turing	Turing	Pascal
Manufacturing Process	TSMC 12nm "FFN"	TSMC 12nm "FFN"	TSMC 12nm "FFN"	TSMC 16nm
Die Size	754mm2	545mm2	445mm2	471mm2

Figure from https://www.anandtech.com/print/13282/nvidia-turing-architecture-deep-dive

Nvidia Turing

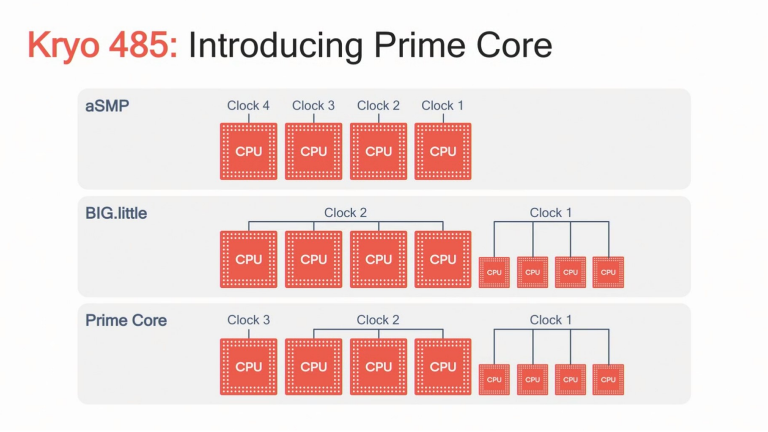
NVIDIA Memory Bandwidth per FLOP (In Bits)			
GPU	**Bandwidth/FLOP**	**Total CUDA FLOPs**	**Total Bandwidth**
RTX 2080	0.36 bits	10.06 TFLOPs	448GB/sec
GTX 1080	0.29 bits	8.87 TFLOPs	320GB/sec
GTX 980	0.36 bits	4.98 TFLOPs	224GB/sec
GTX 680	0.47 bits	3.25 TFLOPs	192GB/sec
GTX 580	0.97 bits	1.58 TFLOPs	192GB/sec

GPU Memory Math: GDDR6 vs. HBM2 vs. GDDR5X						
	NVIDIA GeForce RTX 2080 Ti (GDDR6)	NVIDIA GeForce RTX 2080 (GDDR6)	NVIDIA Titan V (HBM2)	NVIDIA Titan Xp	NVIDIA GeForce GTX 1080 Ti	NVIDIA GeForce GTX 1080
Total Capacity	11 GB	8 GB	12 GB	12 GB	11 GB	8 GB
B/W Per Pin	14 Gb/s		1.7 Gb/s	11.4 Gbps	11 Gbps	
Chip capacity	1 GB (8 Gb)		4 GB (32 Gb)	1 GB (8 Gb)		
No. Chips/KGSDs	11	8	3	12	11	8
B/W Per Chip/Stack	56 GB/s		217.6 GB/s	45.6 GB/s	44 GB/s	
Bus Width	352-bit	256-bit	3092-bit	384-bit	352-bit	256-bit
Total B/W	616 GB/s	448GB/s	652.8 GB/s	547.7 GB/s	484 GB/s	352 GB/s
DRAM Voltage	1.35 V		1.2 V (?)	1.35 V		

Figure from https://www.anandtech.com/print/13282/nvidia-turing-architecture-deep-dive

Qualcomm Snapdragon 855

- Partial specs
  - CPU
    - 1x A76 derivative at 2.84 GHz with 1x 512 KB L2 (prime core)
    - 3x A76 derivative at 2.42 GHz with 3x 256 KB L2
    - 4x A55 derivative at 1.80 GHz with 4x 128 KB L2
    - 2 MB L3
  - GPU
    - Adreno 640
  - DSP
    - Hexagon 690 with ~ 7 TOPS total
    - 4x scalar threads
    - 4x vector of 1024b each
    - 1 tensor accelerator that can work in parallel
  - Memory
    - 4x 16b LPDDR4x at 2133 MHz for 34.1 GB/s
    - 3 MB system level cache
- Links
  - https://www.qualcomm.com/products/snapdragon-855-mobile-platform
  - https://www.qualcomm.com/media/documents/files/snapdragon-855-mobile-platform-product-brief.pdf
  - https://www.anandtech.com/print/13680/snapdragon-855-going-into-detail
  - https://www.anandtech.com/print/13786/snapdragon-855-performance-preview



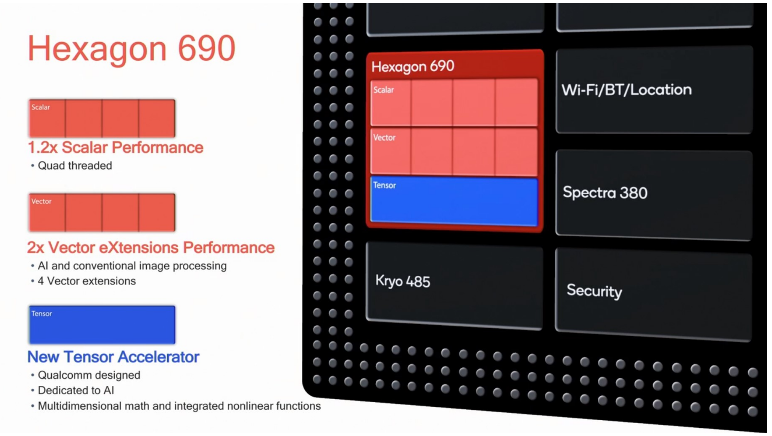


Figure from https://www.anandtech.com/print/13786/snapdragon-855-performance-preview

UW VTA

- Versatile tensor accelerator (VTA)
  - Programmable accelerator that exposes RISC like abstractions for compute and memory at the tensor level
  - https://tvm.ai/vta and https://arxiv.org/abs/1807.04188

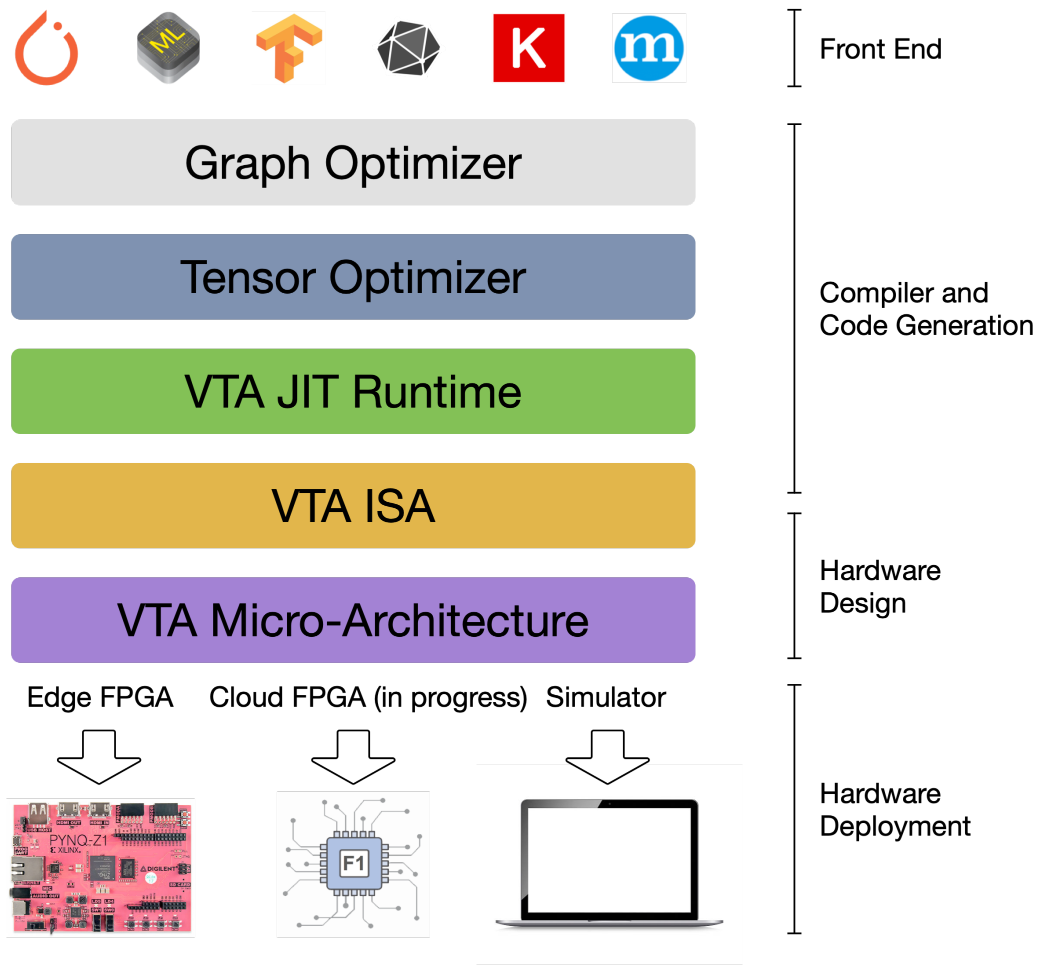


Figure from https://arxiv.org/abs/1807.04188

UW VTA

- Versatile tensor accelerator (VTA)
  - Programmable accelerator that exposes RISC like abstractions for compute and memory at the tensor level
  - https://tvm.ai/vta and https://arxiv.org/abs/1807.04188

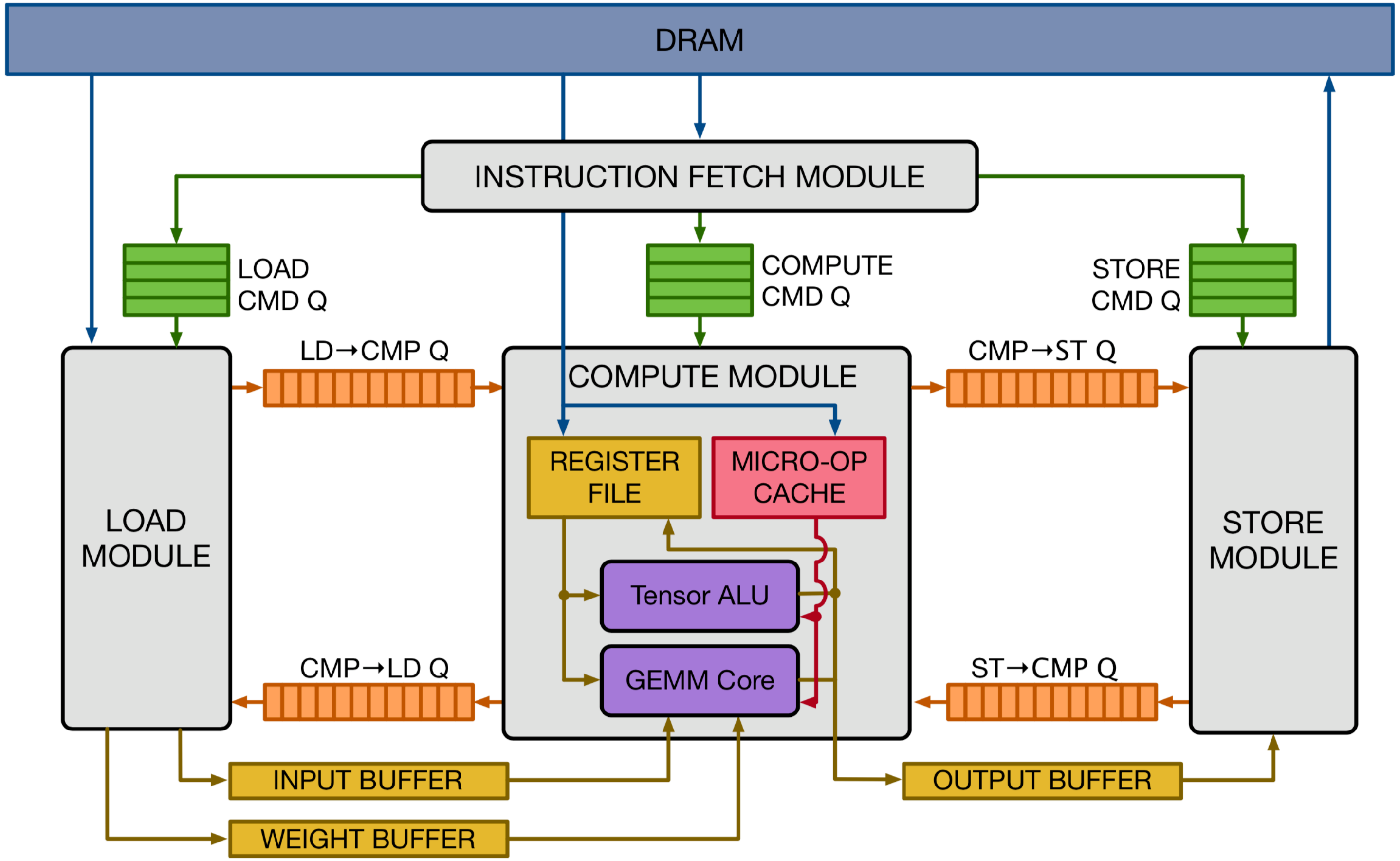


Figure from https://arxiv.org/abs/1807.04188

Xilinx Versal SoC Architecture

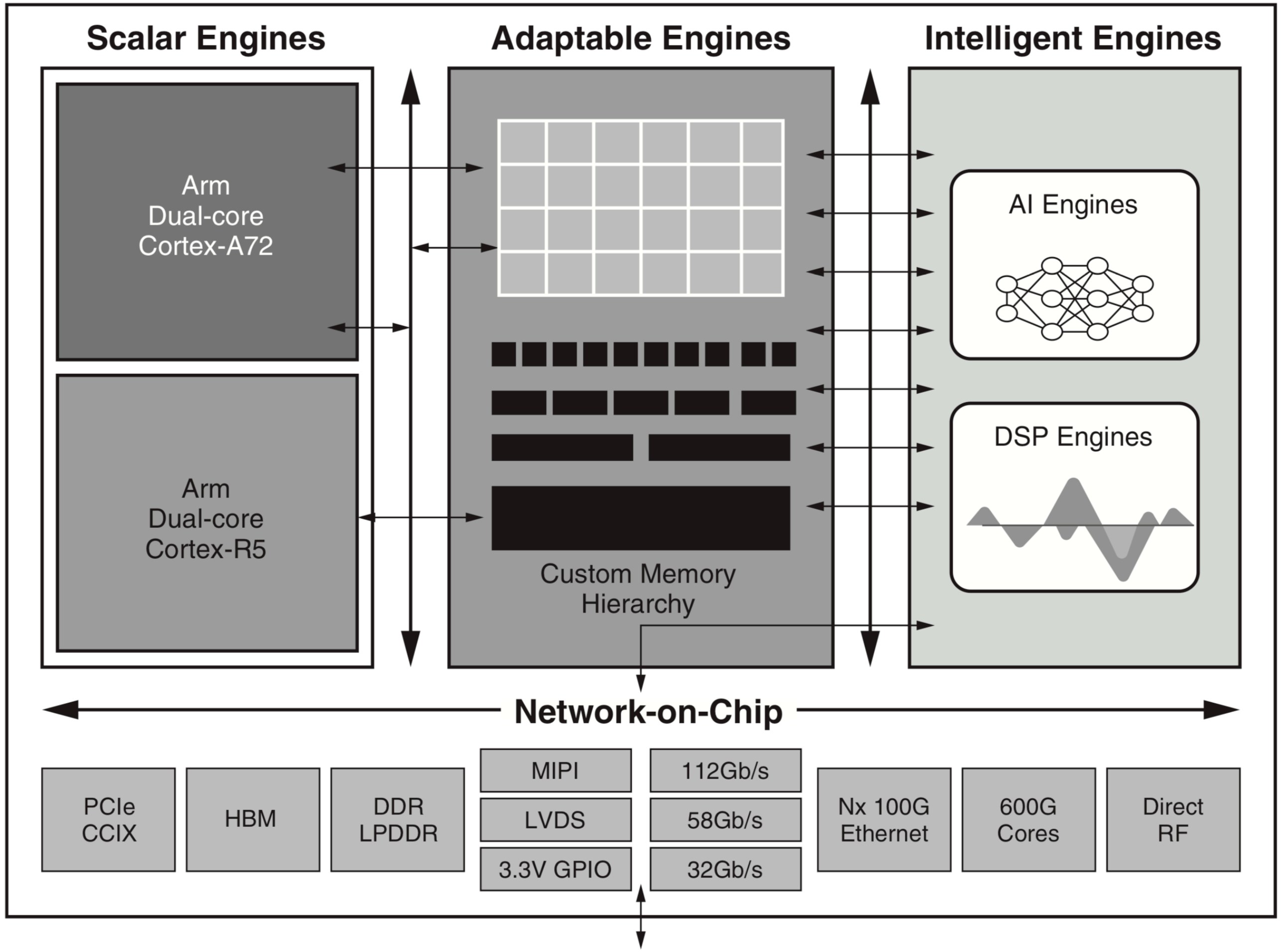


Figure from https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

Xilinx 2D Array Of AI Engines

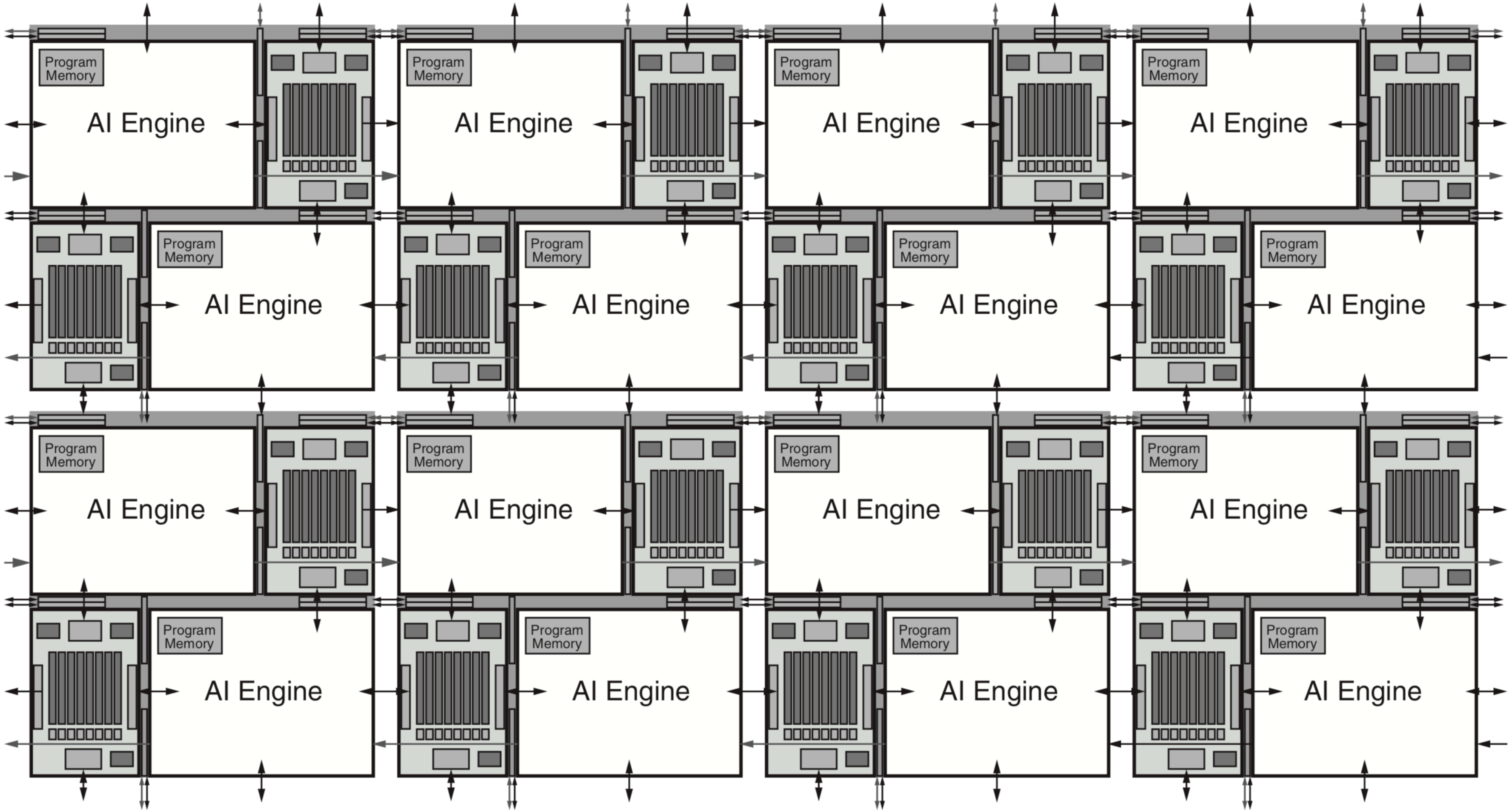


Figure from https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf

Xilinx AI Engine

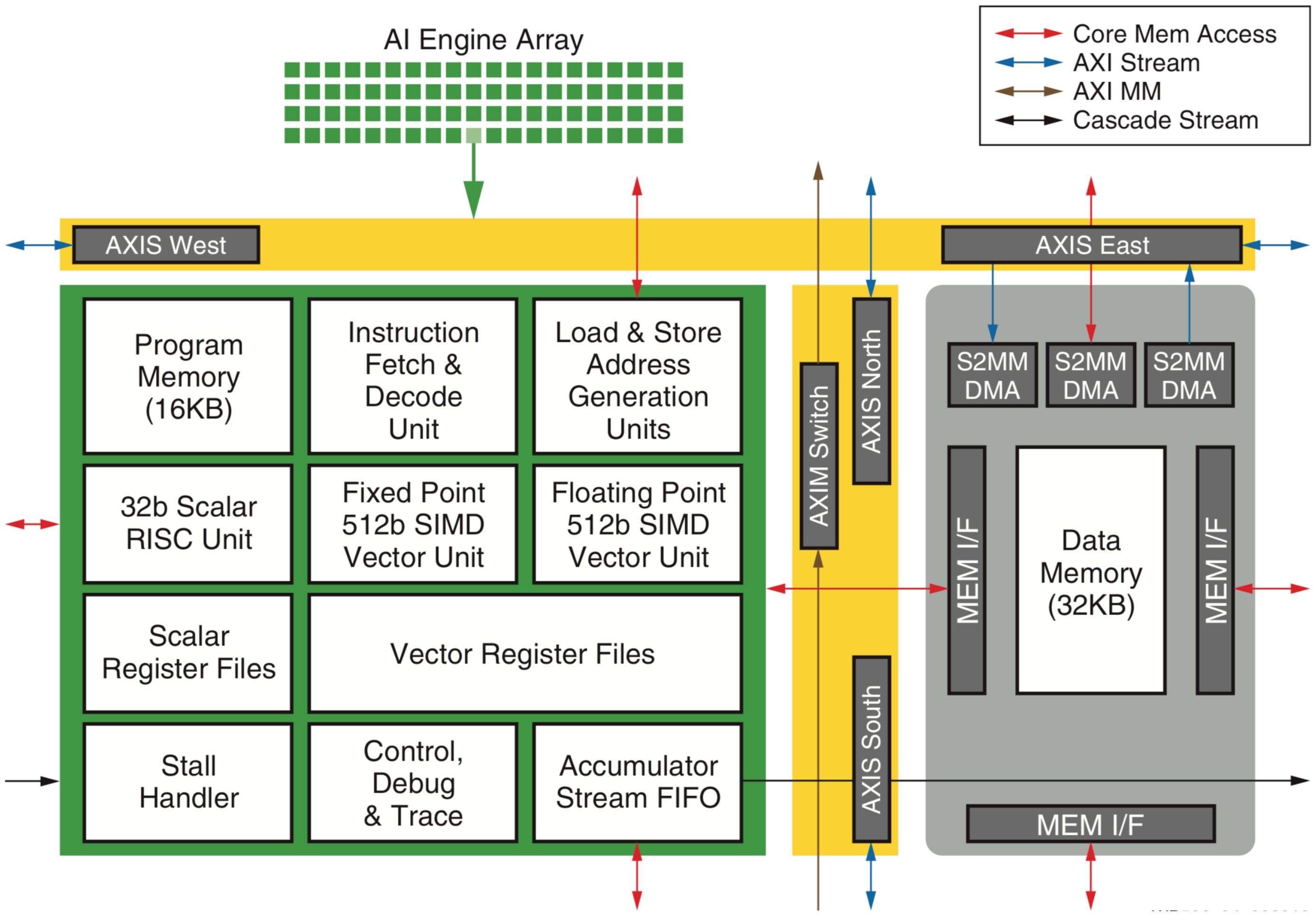


Figure from https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

References

Design

- PACT: parameterized clipping activation for quantized neural networks
  - https://arxiv.org/abs/1805.06085
- Scaling neural machine translation
  - https://arxiv.org/abs/1806.00187
- Rethinking floating point for deep learning
  - https://arxiv.org/abs/1811.01721
- HAQ: hardware-aware automated quantization with mixed precision
  - https://arxiv.org/abs/1811.08886
- On periodic functions as regularizers for quantization of neural networks
  - https://arxiv.org/abs/1811.09862
- SQuantizer: simultaneous learning for both sparse and low-precision neural networks
  - https://arxiv.org/pdf/1812.08301v1.pdf
- Defensive quantization: when efficiency meets robustness
  - https://arxiv.org/abs/1904.08444
- Data-free quantization through weight equalization and bias correction
  - https://arxiv.org/abs/1906.04721

Design

- Ternary weight networks
    - https://arxiv.org/abs/1605.04711
- Ternary neural networks for resource-efficient AI applications
    - https://arxiv.org/abs/1609.00222
- Trained ternary quantization
    - https://arxiv.org/abs/1612.01064
- Ternary neural networks with fine-grained quantization
    - https://arxiv.org/abs/1705.01462
- GXNOR-Net: training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework
    - https://arxiv.org/abs/1705.09283
- Ternary hybrid neural-tree networks for highly constrained IoT applications
    - https://arxiv.org/abs/1903.01531
- Unrolling ternary neural networks
    - https://arxiv.org/abs/1909.04509

Design

- High-efficiency convolutional ternary neural networks with custom adder trees and weight compression
  - https://hal.archives-ouvertes.fr/hal-01686718/document
- Fixed-point feedforward deep neural network design using weights +1, 0, and -1
  - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6986082

- Data-free quantization through weight equalization and bias correction
  - https://arxiv.org/abs/1906.04721
- Table-based neural units: fully quantizing networks for multiply-free inference
  - https://arxiv.org/abs/1906.04798
- Fixed-point optimization of deep neural networks with adaptive step size retraining
  - http://150.162.46.34:8080/icassp2017/pdfs/0001203.pdf

Design

- TensorFlow Lite model optimization
  - https://www.tensorflow.org/lite/performance/model_optimization
- Making floating point math highly efficient for AI hardware
  - https://code.fb.com/ai-research/floating-point-math/
- 8-bit precision for training deep learning systems
  - https://www.ibm.com/blogs/research/2018/12/8-bit-precision-training/
- Highly accurate deep learning inference with 2-bit precision
  - https://www.ibm.com/blogs/research/2019/04/2-bit-precision/

Design

- MorphNet: fast & simple resource-constrained structure learning of deep networks
  - https://arxiv.org/abs/1711.06798
  - https://ai.googleblog.com/2019/04/morphnet-towards-faster-and-smaller.html
- AMC: AutoML for model compression and acceleration on mobile devices
  - https://arxiv.org/abs/1802.03494
- ProxylessNAS: direct neural architecture search on target task and hardware
  - https://arxiv.org/abs/1812.00332
- Network slimming by slimmable networks: towards one-shot architecture search for channel numbers
  - https://arxiv.org/abs/1903.11728

Design

- Estimating or propagating gradients through stochastic neurons for conditional computation
  - https://arxiv.org/abs/1308.3432
- Training deep neural networks with low precision multiplications
  - https://arxiv.org/abs/1412.7024
- Hardware-oriented approximation of convolutional neural networks
  - https://arxiv.org/abs/1604.03168
- Quantized neural networks: training neural networks with low precision weights and activations
  - https://arxiv.org/abs/1609.07061
- Mixed precision training
  - https://arxiv.org/abs/1710.03740
- Quantization and training of neural networks for efficient integer-arithmetic-only inference
  - https://arxiv.org/abs/1712.05877
- Mixed precision training of convolutional neural networks using integer operations
  - https://arxiv.org/abs/1802.00930
- Quantizing deep convolutional networks for efficient inference: A whitepaper
  - https://arxiv.org/abs/1806.08342

Design

- Neural network approximation
  - https://zsc.github.io/megvii-pku-dl-course/slides/Lecture5(Neural%20Network%20Approximation).pdf

- Ristretto: a framework for empirical study of resource-efficient inference in convolutional neural networks
  - http://lepsucd.com/?page_id=621
  - http://lepsucd.com/?page_id=637
  - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8318896
- High performance ultra-low-precision convolutions on mobile devices
  - https://arxiv.org/abs/1712.02427

- IEEE 754
  - https://en.wikipedia.org/wiki/IEEE_754
- bfloat16 floating-point format
  - https://en.wikipedia.org/wiki/Bfloat16_floating-point_format

Design

- XNOR-Net: ImageNet classification using binary convolutional neural networks
    - https://arxiv.org/abs/1603.05279
    - https://github.com/allenai/XNOR-Net
- Towards accurate binary convolutional neural network
    - https://arxiv.org/abs/1711.11294
- Bi-Real Net: enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm
    - https://arxiv.org/abs/1808.00278
    - https://github.com/liuzechun/Bi-Real-net
- Structured binary neural networks for accurate image classification and semantic segmentation
    - https://arxiv.org/abs/1811.10413
    - https://github.com/bohanzhuang/Group-Net-image-classification
    - https://github.com/bohanzhuang/Group-Net-semantic-segmentation

Design

- MeliusNet: can binary neural networks achieve MobileNet-level accuracy?
  - https://arxiv.org/abs/2001.05936
  - https://github.com/hpi-xnor/BMXNet-v2-examples
- ReActNet: towards precise binary neural network with generalized activation functions
  - https://arxiv.org/abs/2003.03488
- BiDet: an efficient binarized object detector
  - https://arxiv.org/abs/2003.03961
  - https://github.com/ZiweiWangTHU/BiDet
- Training binary neural networks with real-to-binary convolutions
  - https://arxiv.org/abs/2003.11535
  - https://github.com/brais-martinez/real2binary
- BCNN: a Binary CNN with all matrix ops quantized to 1 bit precision
  - https://arxiv.org/abs/2010.00704

Design

- Ternary weight networks
  - https://arxiv.org/abs/1605.04711
- Trained ternary quantization
  - https://arxiv.org/abs/1612.01064
- Ternary residual networks
  - https://arxiv.org/abs/1707.04679

Design

- How to train a compact binary neural network with high accuracy?
  - https://www.ganghua.org/publication/AAAI17.pdf
- Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1
  - https://arxiv.org/abs/1602.02830
- Quantized neural networks: training neural networks with low precision weights and activations
  - https://arxiv.org/abs/1609.07061
- Deep learning with low precision by half-wave Gaussian quantization
  - https://arxiv.org/abs/1702.00953
- Incremental network quantization: towards lossless CNNs with low-precision weights
  - https://arxiv.org/abs/1702.03044
  - https://github.com/AojunZhou/Incremental-Network-Quantization
- ReBNet: residual binarized neural network
  - https://arxiv.org/abs/1711.01243
- Regularized binary network training
  - https://arxiv.org/abs/1812.11800

Design

- Self-binarizing networks
  - https://arxiv.org/abs/1902.00730
- Learning low-precision neural networks without straight-through estimator (STE)
  - https://arxiv.org/abs/1903.01061
- Regularizing activation distribution for training binarized deep networks
  - https://arxiv.org/abs/1904.02823
- Improved training of binary networks for human pose estimation and image recognition
  - https://arxiv.org/abs/1904.05868
- Forward and backward information retention for accurate binary neural networks
  - https://arxiv.org/abs/1909.10788
  - https://github.com/htqin/IR-Net
- Quantization networks
  - https://arxiv.org/abs/1911.09464
  - https://github.com/aliyun/alibabacloud-quantization-networks

Design

- Awesome pruning
  - https://github.com/he-y/Awesome-Pruning
- PyTorch pruning tutorial
  - https://pytorch.org/tutorials/intermediate/pruning_tutorial.html
- Efficient and sparse neural networks by pruning weights in a multiobjective learning approach
  - https://arxiv.org/abs/2008.13590
- Understanding diversity based pruning of neural networks -- statistical mechanical analysis
  - https://arxiv.org/abs/2006.16617
- ESPN: extremely sparse pruned networks
  - https://arxiv.org/abs/2006.15741
- Movement pruning: adaptive sparsity by fine-tuning
  - https://arxiv.org/abs/2005.07683
- Pruned neural networks are surprisingly modular
  - https://arxiv.org/abs/2003.04881
- What is the state of neural network pruning?
  - https://arxiv.org/abs/2003.03033

Design

- Comparing rewinding and fine-tuning in neural network pruning
  - https://arxiv.org/abs/2003.02389
  - https://github.com/lottery-ticket/rewinding-iclr20-public
- Fast sparse ConvNets
  - https://arxiv.org/abs/1911.09723
- Gate decorator: global filter pruning method for accelerating deep convolutional neural networks
  - https://arxiv.org/abs/1909.08174
- Importance estimation for neural network pruning
  - https://arxiv.org/abs/1906.10771
- SpArSe: sparse architecture search for CNNs on resource-constrained microcontrollers
  - https://arxiv.org/abs/1905.12107
- Rethinking the value of network pruning
  - https://arxiv.org/abs/1810.05270
- Learning sparse neural networks through L0 regularization
  - https://arxiv.org/abs/1712.01312

Design

- To prune, or not to prune: exploring the efficacy of pruning for model compression
  - https://arxiv.org/abs/1710.01878
- Channel pruning for accelerating very deep neural networks
  - https://arxiv.org/abs/1707.06168
- Exploring the regularity of sparse structure in convolutional neural networks
  - https://arxiv.org/abs/1705.08922
- Faster CNNs with direct sparse convolutions and guided pruning
  - https://arxiv.org/abs/1608.01409
- Optimal brain surgeon and general network pruning
  - https://authors.library.caltech.edu/54981/1/Optimal%20Brain%20Surgeon%20and%20general%20network%20pruning.pdf
- Optimal brain damage
  - http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf

Design

- Gradient flow in sparse neural networks and how lottery tickets win
  - https://arxiv.org/abs/2010.03533
- Training sparse neural networks using compressed sensing
  - https://arxiv.org/abs/2008.09661
- SparseTrain: exploiting dataflow sparsity for efficient convolutional neural networks training
  - https://arxiv.org/abs/2007.13595
- Learning sparse filters in deep convolutional neural networks with a l1/l2 pseudo-norm
  - https://arxiv.org/abs/2007.10022
- Pruning neural networks without any data by iteratively conserving synaptic flow
  - https://arxiv.org/abs/2006.05467
- What needles do sparse neural networks find in nonlinear haystacks
  - https://arxiv.org/abs/2006.04041
- Dynamic sparse training: find efficient sparse network from scratch with trainable masked layers
  - https://arxiv.org/abs/2005.06870

Design

- On the transferability of winning tickets in non-natural image datasets
  - https://arxiv.org/abs/2005.05232
- Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture
  - https://arxiv.org/abs/2005.03454
- Pruning untrained neural networks: principles and analysis
  - https://arxiv.org/abs/2002.08797
- NeuroFabric: identifying ideal topologies for training a priori sparse networks
  - https://arxiv.org/abs/2002.08339
- Picking winning tickets before training by preserving gradient flow
  - https://arxiv.org/abs/2002.07376
- Sparse weight activation training
  - https://arxiv.org/abs/2001.01969
- l0 regularized structured sparsity convolutional neural networks
  - https://arxiv.org/abs/1912.07868

Design

- Winning the lottery with continuous sparsification
  - https://arxiv.org/abs/1912.04427
- Rigging the lottery: making all tickets winners
  - https://arxiv.org/abs/1911.11134
  - https://github.com/google-research/rigl
- Sparse networks from scratch: faster training without losing performance
  - https://arxiv.org/abs/1907.04840
- The difficulty of training sparse neural networks
  - https://arxiv.org/abs/1906.10732
- Deconstructing lottery tickets: zeros, signs, and the supermask
  - https://arxiv.org/abs/1905.01067
- Stabilizing the lottery ticket hypothesis
  - https://arxiv.org/abs/1903.01611
- The state of sparsity in deep neural networks
  - https://arxiv.org/abs/1902.09574

Design

- SNIP: single-shot network pruning based on connection sensitivity
  - https://arxiv.org/abs/1810.02340
- The lottery ticket hypothesis: finding sparse, trainable neural networks
  - https://arxiv.org/abs/1803.03635
- Learning both weights and connections for efficient neural networks
  - https://arxiv.org/abs/1506.02626

Software

- Netscope CNN analyzer
  - https://dgschwend.github.io/netscope/quickstart.html

- Model zoo
  - https://modelzoo.co

- Python review
  - http://web.stanford.edu/class/cs224n/lectures/python-review.pdf
- Python numpy tutorial
  - http://cs231n.github.io/python-numpy-tutorial/

Software

- An introduction to TensorFlow
  - http://web.stanford.edu/class/cs224n/lectures/lecture6.pdf
  - http://web.stanford.edu/class/cs224n/readings/tensorflow_tutorial_code.zip
- Introduction to TensorFlow
  - https://www.youtube.com/watch?v=PicxU81owCs#t=3m16s
- TensorFlow tutorial
  - http://web.stanford.edu/class/cs224s/lectures/Tensorflow-tutorial.pdf
  - https://github.com/pbhatnagar3/cs224s-tensorflow-tutorial
- TensorFlow quantization aware training
  - https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/quantize
- TensorFlow models research slim
  - https://github.com/tensorflow/models/tree/master/research/slim
  - https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md
- TensorFlow models official
  - https://github.com/tensorflow/models/tree/master/official

Software

- Introduction to PyTorch code examples
    - https://cs230-stanford.github.io/pytorch-getting-started.html
- Hardware and software
    - http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture08.pdf
- Welcome to PyTorch tutorials
    - https://pytorch.org/tutorials/
- PyTorch examples
    - https://github.com/jcjohnson/pytorch-examples
- The incredible PyTorch
    - https://github.com/ritchieng/the-incredible-pytorch
- PyTorch docs torchvision models
    - https://pytorch.org/docs/stable/torchvision/models.html
- Pretrained models PyTorch
    - https://github.com/Cadene/pretrained-models.pytorch
- Pytorch mobilenet
    - https://github.com/marvis/pytorch-mobilenet

Software

- Glow: graph lowering compiler techniques for neural networks
  - https://arxiv.org/abs/1805.00907
- Compiler for neural network hardware accelerators
  - https://github.com/pytorch/glow
- Glow
  - https://facebook.ai/developers/tools/glow
- Halide a language for fast, portable computation on images and tensors
  - http://halide-lang.org
- Loopy: transformation-based generation of high-performance CPU/GPU code
  - https://github.com/inducer/loopy
- TVM: an automated end-to-end optimizing compiler for deep learning
  - https://arxiv.org/abs/1802.04799
- End to end deep learning compiler stack for CPUs, GPUs and specialized accelerators
  - https://tvm.ai
- Designing computer systems for software 2.0
  - https://iscaconf.org/isca2018/docs/Kunle-ISCA-Keynote-2018.pdf

Software

- Learning to optimize tensor programs
  - https://arxiv.org/abs/1805.08166

- TensorFlow XLA overview
  - https://www.tensorflow.org/extend/xla/
- Tensor comprehensions: framework-agnostic high-performance machine learning abstractions
  - https://arxiv.org/abs/1802.04730
- Tensor comprehensions
  - https://facebook.ai/developers/tools/tensorcomprehensions

- ONNX
  - https://facebook.ai/developers/tools/onnx
- Open neural network exchange
  - https://github.com/onnx

Software

- The LLVM Compiler Infrastructure
  - https://llvm.org
- LLVM: a compilation framework for lifelong program analysis & transformation
  - https://llvm.org/pubs/2004-01-30-CGO-LLVM.html

- OpenMPI: open source high performance computing
  - https://www.open-mpi.org
- OpenMP
  - https://www.openmp.org
- OpenCL
  - https://www.khronos.org/opencl/
- CUDA
  - https://developer.nvidia.com/cuda-zone

Software

- GPipe: efficient training of giant neural networks using pipeline parallelism
    - https://arxiv.org/abs/1811.06965

- The state of machine learning frameworks in 2019
    - https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/
- NVDLA deep learning inference compiler is now open source
    - https://devblogs.nvidia.com/nvdla/
- Compilers for machine learning
    - https://www.c4ml.org

Software

- BLAS (basic linear algebra subprograms)
  - http://www.netlib.org/blas/
- Automatically tuned linear algebra software (ATLAS)
  - http://math-atlas.sourceforge.net
- BLAS-like library instantiation software framework
  - https://github.com/flame/blis
- High-performance object-based library for DLA computations
  - https://github.com/flame/libflame
- FFTW
  - http://www.fftw.org
- Spiral software / hardware generation for performance
  - http://www.spiral.net/index.html
- cuDNN
  - https://developer.nvidia.com/cudnn

Hardware

- MIT 6.004 Computation Structures
  - https://computationstructures.org/index.html

- John Hennessy and David Patterson 2017 ACM A.M. Turing award lecture
  - https://www.youtube.com/watch?v=3LVeEjsn8Ts
- The future of computing: a conversation with John Hennessy (Google I/O '18)
  - https://www.youtube.com/watch?v=Azt8Nc-mtKM
- A new golden age for computer architecture
  - https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- Amdahl's law and its proof
  - https://www.geeksforgeeks.org/computer-organization-amdahls-law-and-its-proof/
- Performance modeling
  - http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2013/Lect07_short.pdf
- Roofline: an insightful visual performance model for multicore architectures
  - https://dl.acm.org/citation.cfm?id=1498785

Hardware

- Moore's law and Dennard scaling
  - http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture15.pdf
- The future of microprocessors
  - https://cacm.acm.org/magazines/2011/5/107702-the-future-of-microprocessors/
- Computing's energy problem (and what we can do about it)
  - http://eecs.oregonstate.edu/research/vlsi/teaching/ECE471_WIN15/mark_horowitz_ISSCC_2014.pdf
- The accelerator store: a shared memory framework for accelerator-based systems
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.994&rep=rep1&type=pdf
- Dark silicon and the end of multicore scaling
  - https://www.cc.gatech.edu/~hadi/doc/paper/2012-toppicks-dark_silicon.pdf
- Is dark silicon useful?
  - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6241647&tag=1
- Dark memory and accelerator-rich system optimization in the dark silicon era
  - https://arxiv.org/abs/1602.04183
- Co-designing accelerators and soc interfaces using gem5-aladdin
  - http://vlsiarch.eecs.harvard.edu/wp-content/uploads/2016/08/shao_micro2016.pdf

Hardware

- The impact of Moore’s law and loss of Dennard scaling
  - https://indico.cern.ch/event/397113/contributions/1837780/attachments/1215934/1775678/Talk_2016-01-21.pdf
- The scaling of MOSFETS, Moore’s law, and ITRS
  - http://userweb.eng.gla.ac.uk/fikru.adamu-lema/Chapter_02.pdf
- Integrated power management, leakage control and process compensation technology for advanced processes
  - https://www.design-reuse.com/articles/20296/power-management-leakage-control-process-compensation.html
- An introduction to computation technologies in deep learning
  - https://zsc.github.io/megvii-pku-dl-course/slides18/dl-comp-tech.pdf

Hardware

- Dynamic random-access memory
  - https://en.wikipedia.org/wiki/Dynamic_random-access_memory
- Static random-access memory
  - https://en.wikipedia.org/wiki/Static_random-access_memory

Hardware

- Vector-matrix multiply and winner-take-all as an analog classifier
  - https://ieeexplore.ieee.org/document/6519956
- Evaluation of an analog accelerator for linear algebra
  - https://ieeexplore.ieee.org/document/7551423
- Charge-mode parallel architecture for vector-matrix multiplication
  - https://ieeexplore.ieee.org/document/974781
- Analysis and design of a passive switched-capacitor matrix multiplier for approximate computing
  - https://ieeexplore.ieee.org/document/7579580
- SysML 18: Jonathan Binas, Analog electronic deep networks for fast and efficient inference
  - https://www.youtube.com/watch?reload=9&v=8t0Yunt5kE4

Hardware

- Fast algorithms for convolutional neural networks
  - https://arxiv.org/abs/1509.09308
- Efficient processing of deep neural networks: a tutorial and survey
  - https://arxiv.org/abs/1703.09039
- Efficient sparse-Winograd convolutional neural networks
  - https://arxiv.org/abs/1802.06367
- Accelerating CNN inference on FPGAs: a survey
  - https://arxiv.org/abs/1806.01683
- Design automation for efficient deep learning computing
  - https://arxiv.org/abs/1904.10616
- Tutorial on hardware architectures for deep neural networks
  - http://eyeriss.mit.edu/tutorial.html
- Understanding the limitations of existing energy-efficient design approaches for deep neural networks
  - http://www.rle.mit.edu/eems/wp-content/uploads/2018/02/2018_SysML_final.pdf

Hardware

- ARM system IP
  - https://developer.arm.com/products/system-ip
- ARM details ”Project Trillium" machine learning processor architecture
  - https://www.anandtech.com/show/12791/arm-details-project-trillium-mlp-architecture
- Arm's new Mali-G77 & Valhall GPU architecture: a major leap
  - https://www.anandtech.com/print/14385/arm-announces-malig77-gpu
- Cadence announces the Tensilica DNA 100 IP: bigger artificial intelligence
  - https://www.anandtech.com/show/13377/cadence-announces-tensilica-dna-100-a-bigger-nn-ip
- Deep learning inference in Facebook data centers: characterization, performance optimizations and hardware implications
  - https://arxiv.org/abs/1811.09886
- Machine learning at Facebook: understanding inference at the edge
  - https://research.fb.com/publications/machine-learning-at-facebook-understanding-inference-at-the-edge/
- An in-depth look at Google’s first tensor processing unit (TPU)
  - https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu
- In-datacenter performance analysis of a tensor processing unit
  - https://arxiv.org/abs/1704.04760

Hardware

- First in-depth look at Google's TPU architecture
  - https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/
- Under the hood of Google's TPU2 machine learning clusters
  - https://www.nextplatform.com/2017/05/22/hood-googles-tpu2-machine-learning-clusters/
- Tearing apart Google's TPU 3.0 AI coprocessor
  - https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/
- Cloud TPU system architecture
  - https://cloud.google.com/tpu/docs/system-architecture

Hardware

- Eyeriss v2: a flexible and high-performance accelerator for emerging deep neural networks
    - https://arxiv.org/abs/1807.07928
- NNgen: a fully-customizable hardware synthesis compiler for deep neural network
    - https://github.com/NNgen/nngen
- NVDLA primer
    - http://nvdla.org/primer.html
- The Nvidia Turing GPU architecture deep dive: prelude to GeForce RTX
    - https://www.anandtech.com/print/13282/nvidia-turing-architecture-deep-dive
- Xilinx AI engines and their applications
    - https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf
- Versal: the first adaptive compute acceleration platform (ACAP)
    - https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

Performance

- MLPerf (machine learning performance benchmarking suite)
  - Links
    - https://mlperf.org
    - https://mlperf.org/assets/static/media/MLPerf-User-Guide.pdf
    - https://github.com/mlperf/reference
    - https://github.com/mlperf/submissions
  - Tests
    - Image_classification - Resnet-50 v1 applied to Imagenet
    - Object_detection - Mask R-CNN applied to COCO
    - Speech_recognition - DeepSpeech2 applied to Librispeech
    - Translation - Transformer applied to WMT English-German
    - Recommendation - Neural Collaborative Filtering applied to MovieLens 20 Million (ml-20m)
    - Sentiment_analysis - Seq-CNN applied to IMDB dataset
    - Reinforcement - Mini-go applied to predicting pro game moves

Performance

- MLPerf releases first inference benchmark results; Nvidia touts its showing
  - https://www.hpcwire.com/2019/11/06/mlperf-releases-first-inference-benchmark-results-nvidia-touts-its-showing/
- Stanford data analytics for what's next (DAWN) project (includes a deep learning benchmark)
  - https://dawn.cs.stanford.edu/benchmark/
  - http://dawn.cs.stanford.edu/2018/04/30/dawnbench-v1-results/
- Efficient processing of deep neural networks: a tutorial and survey
  - https://arxiv.org/abs/1703.09039
- Benchmark analysis of representative deep neural network architectures
  - https://arxiv.org/abs/1810.00736