GAP*flow*

Presenter:
Marco Fariselli
Embedded ML Engineer
marco.fariselli@greenwaves-technologies.com

GREENWAVES TECHNOLOGIES

GreenWaves Technologies

Best hardware product
Embedded World 2023

- Fabless semiconductor startup founded in 2014.
- We design and sell **extreme performance** processors for **energy constrained devices**
- 45 people, HQ in **Grenoble,** France
- Offices in **Bologna**, Italy, **Shanghai**, China, **Copenhagen**, Denmark. Global sales footprint.

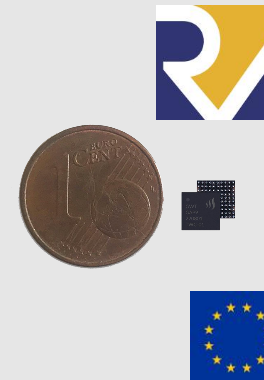
Embedded Technologies Award 2023
Les Assises de l'Embarqué

GAP8

In production since 2020 one of the very first commercially available RISC-V processor and AI microcontroller

GAP9

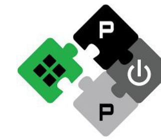
Second generation ultra-low power AI and DSP enabled Microcontroller



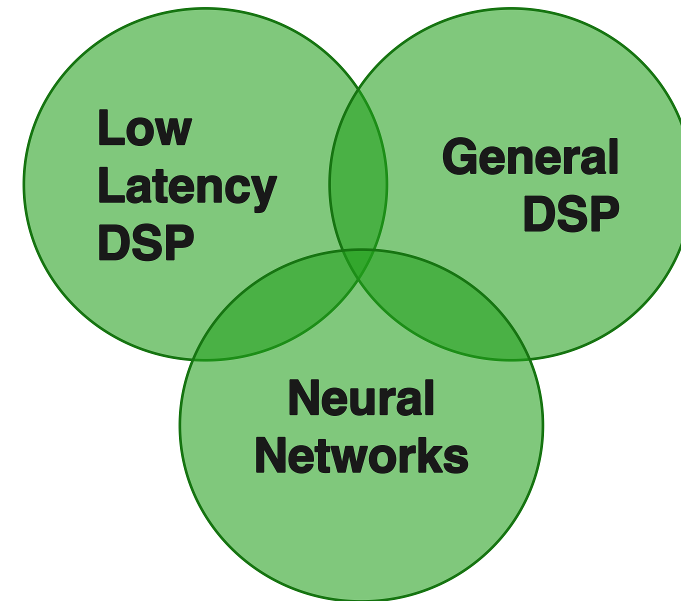
Gartner
COOL VENDOR 2019

Cool Vendors in AI Semiconductors, Alan Priestley, Saniye Alaybeyi, April 29, 2019.

Foundations of GAP



- Born from research groups:
  - **PULP** (Unibo & ETH Zurich)
  - **RISC-V**
- Push **Edge AI** to the limits
  - Specialized HW
  - Optimized SW
  - 10-100 GOPS @ <10mW always-on
- **Programmability**
  - RISC-V open source ISA
  - High level tools



Company Proprietary

GREENWAVES TECHNOLOGIES

Outline

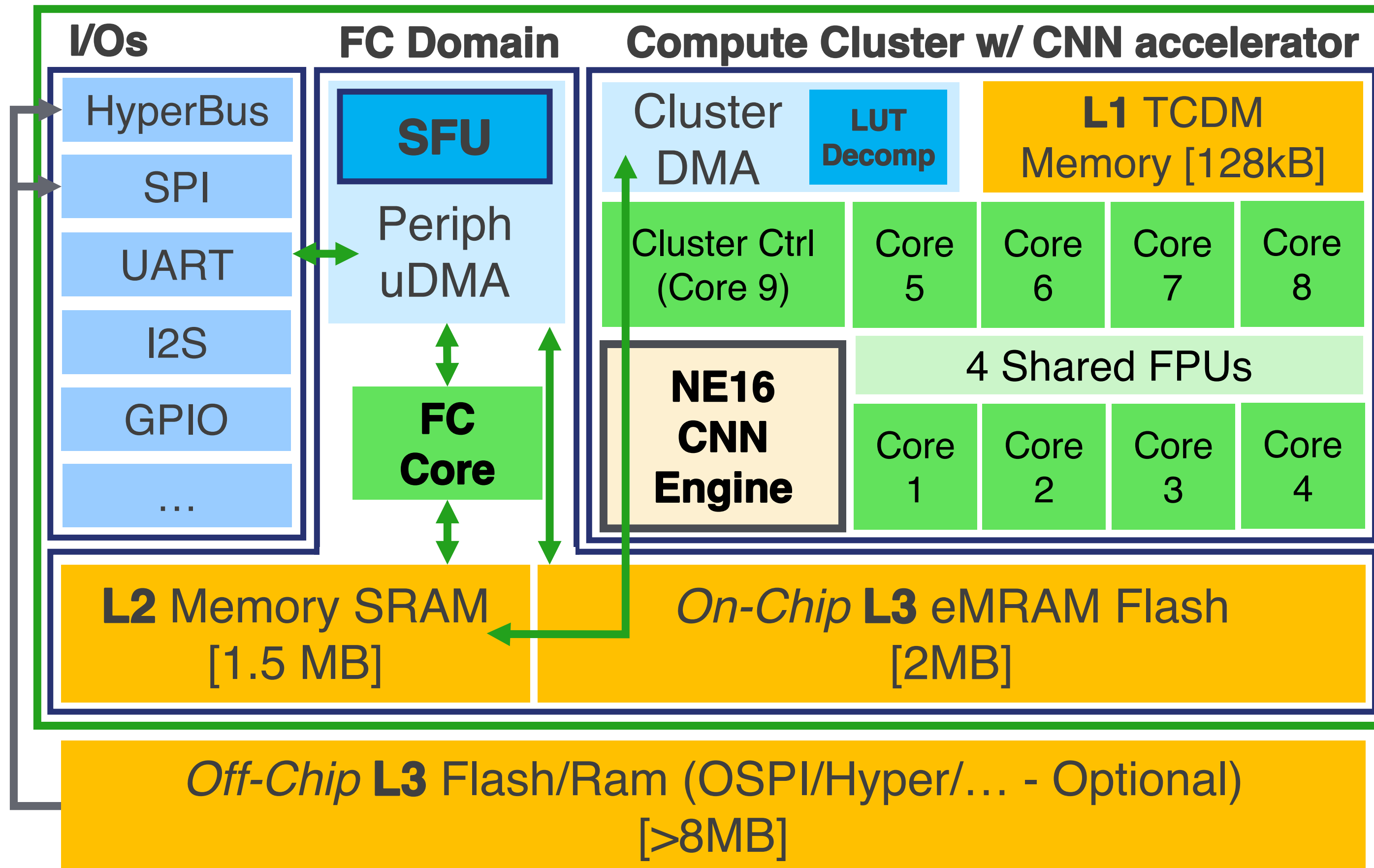
- **GAP Architecture**
- **GAP*flow***
    - **NNtool**: Graph optimizations / Quantization
    - **Autotiler**: Memory management
- **NE16**
- **Hands-on**

Company Proprietary

GREENWAVES TECHNOLOGIES

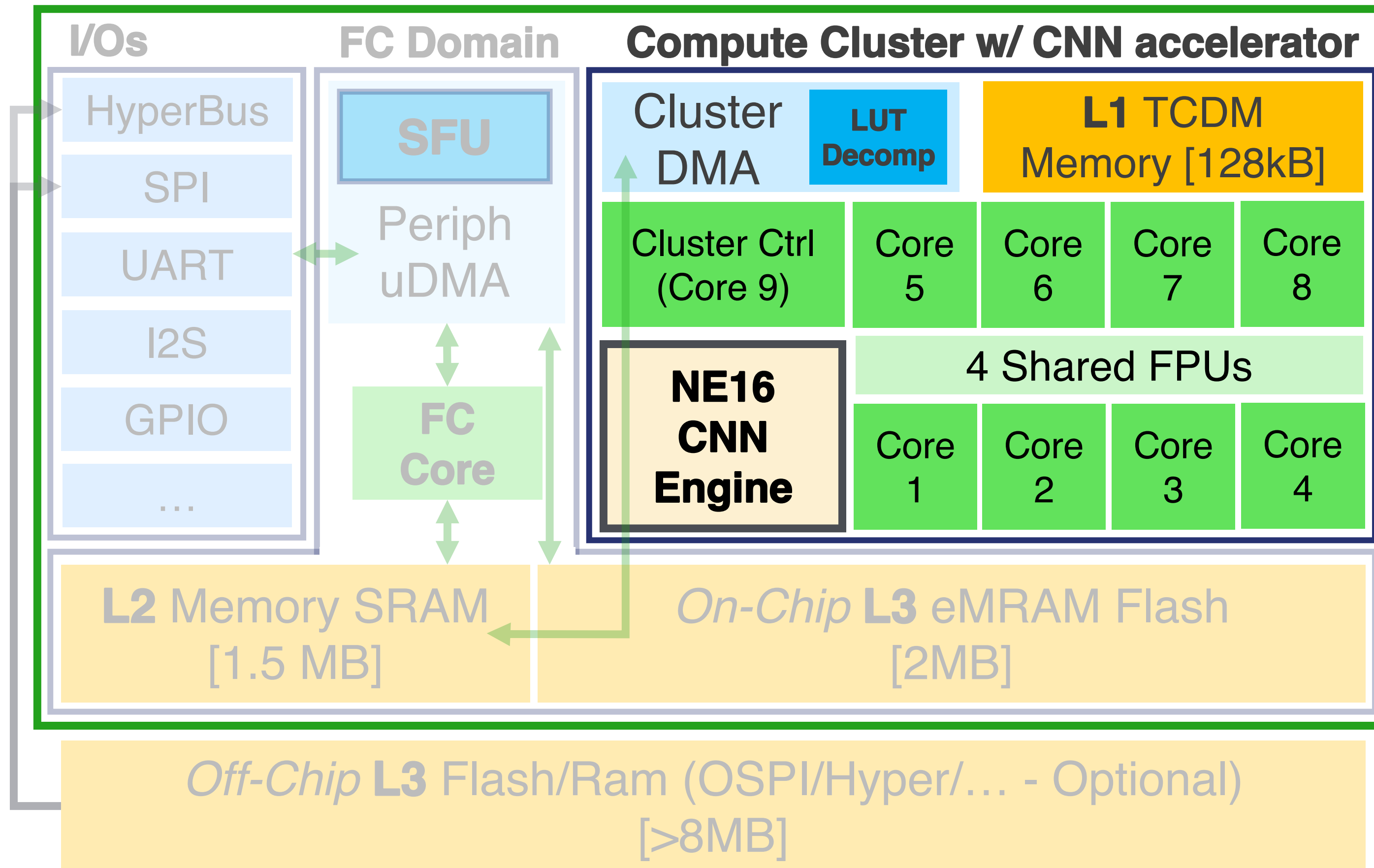
GAP Architecture

GAP9 Architecture



Company Proprietary
GREENWAVES TECHNOLOGIES

GAP9 Architecture

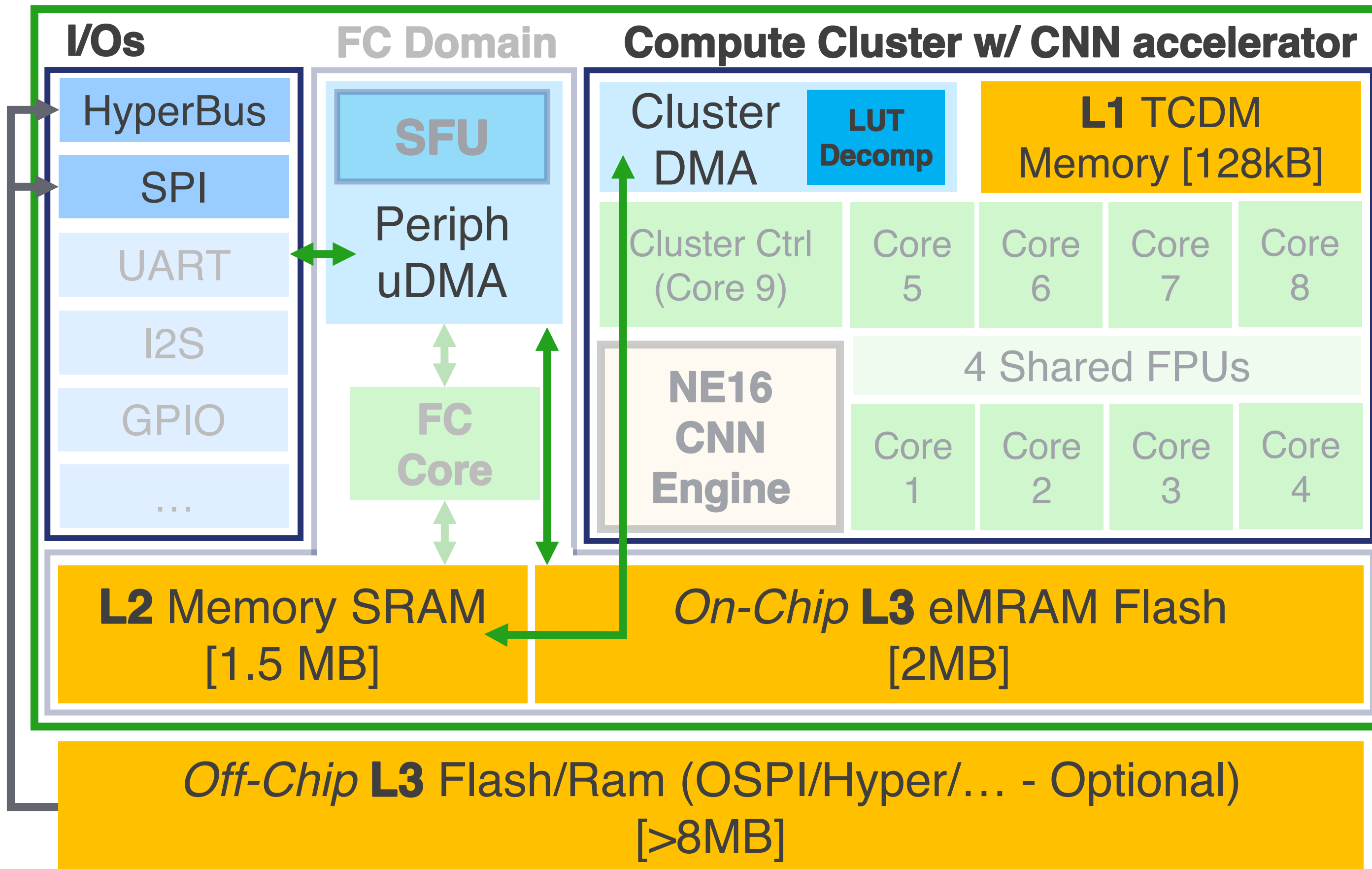


Hierarchical Compute paradigm

- **4** independent frequency domains: **FC - I/Os - Cluster - SFU**
- *"Turn-on when you need"*

Company Proprietary
GREENWAVES TECHNOLOGIES

GAP9 Architecture



Hierarchical Compute paradigm

- **4** independent frequency domains: **FC - I/Os - Cluster - SFU**
- *"Turn-on when you need"*

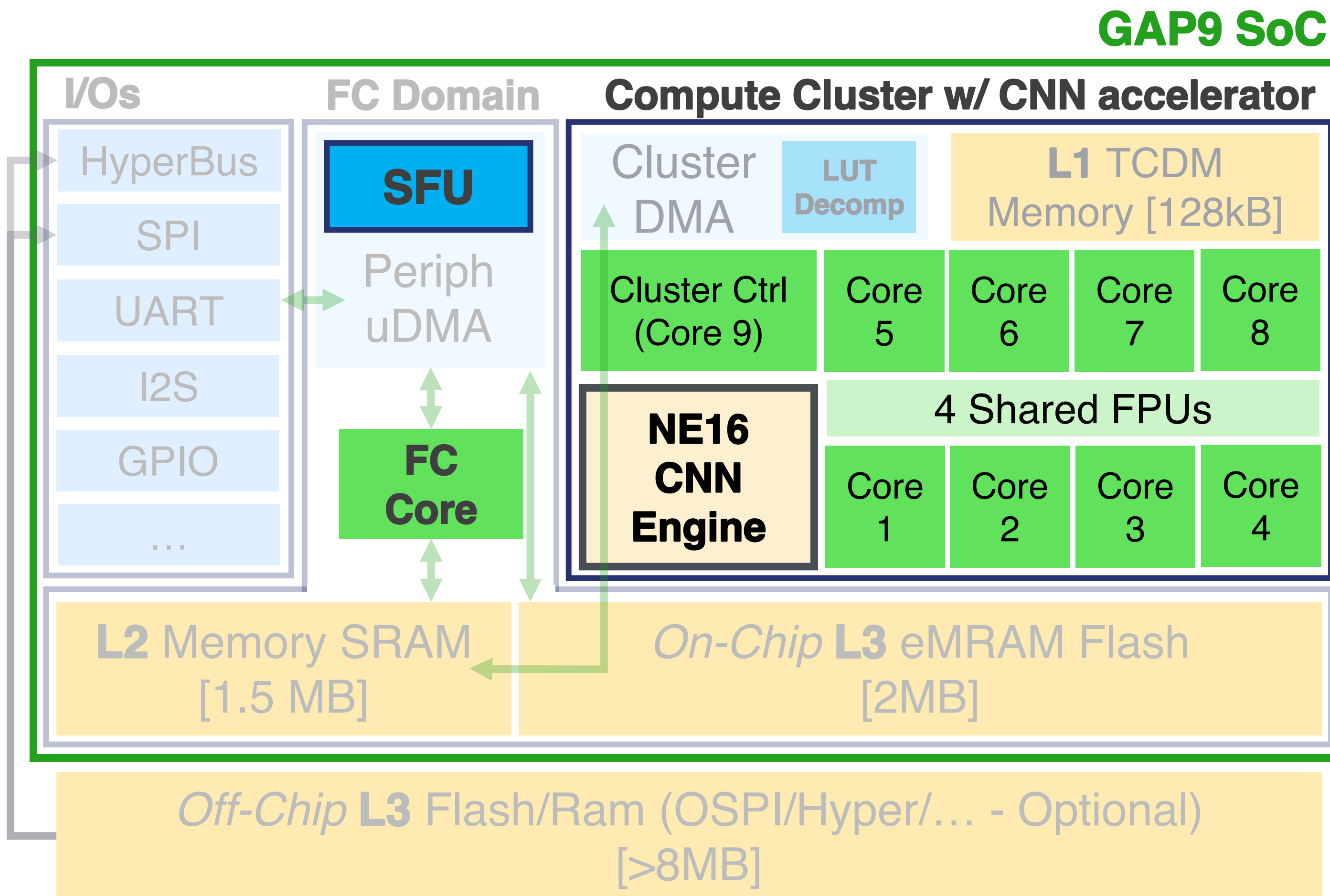
Hierarchical Memory Architecture (w/o D-Cache)

- L1: 128kB – 1 Cyc/Access
- L2: 1.5MB – 10-100 Cyc/Access
- L3: >2MB – 100-1000 Cyc/Access
- DMA and uDMA for background copies+decompression

Company Proprietary

GREENWAVES TECHNOLOGIES

GAP9 Architecture



Hierarchical Compute paradigm

- **4** independent frequency domains: **FC - I/Os - Cluster - SFU**
- *“Turn-on when you need”*

Hierarchical Memory Architecture (w/o D-Cache)

- L1: 128kB – 1 Cyc/Access
- L2: 1.5MB – 10-100 Cyc/Access
- L3: >2MB – 100-1000 Cyc/Access
- DMA and uDMA for background copies+decompression

Heterogeneous Compute Units

- 10 General Purpose RISC-V Cores
- 4 Shared FPUs (half/single precision)
- Conv/MatMul HW Accelerator (NE16)
- Low-Latency time-domain DSP Accelerator (SFU)

Company Proprietary

GREENWAVES TECHNOLOGIES

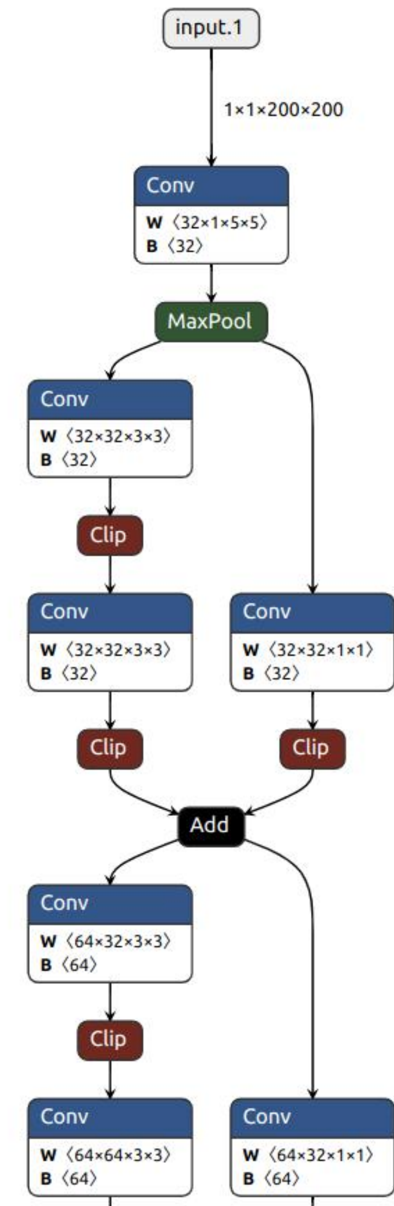
GAPflow

GREENWAVES
TECHNOLOGIES

Neural Network from GAPflow POV

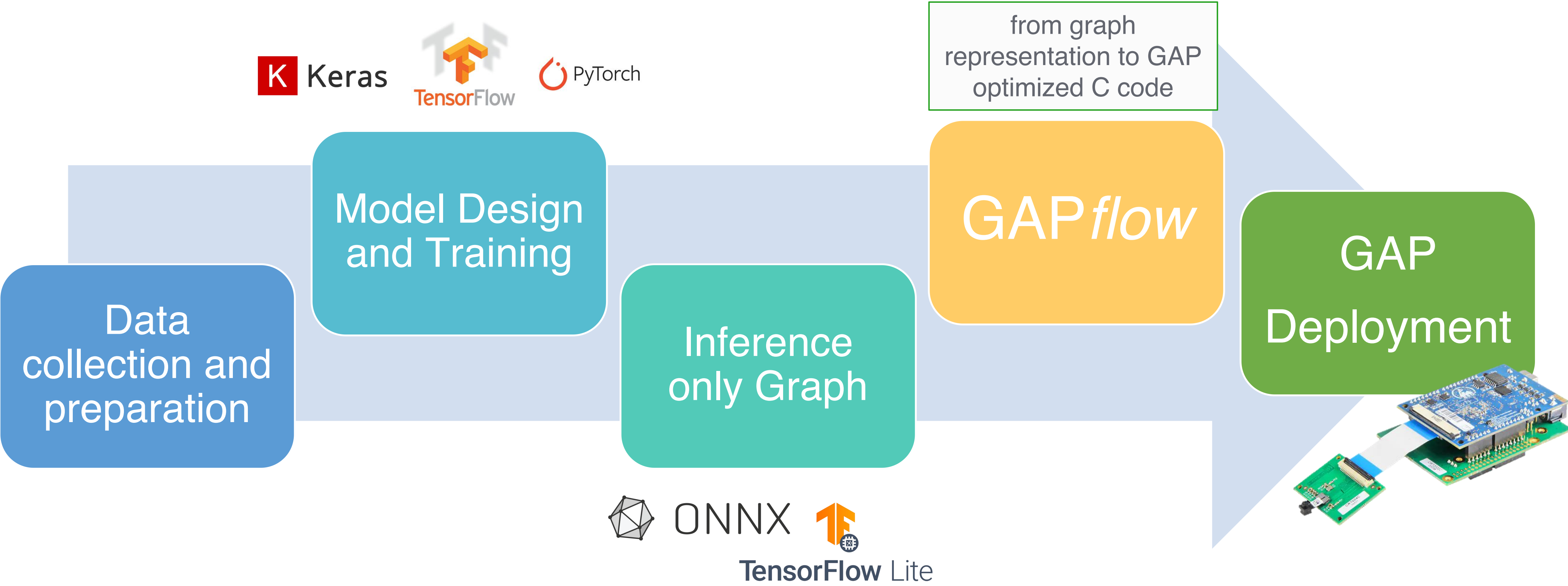
Computational Graph:

- ***Nodes = Computational Layers***
  - e.g. Conv2D, MatMul, MatAdd, Pooling, ...
  - *COMPUTATIONAL COST*
- ***Edges = Tensors***
  - e.g. Input, Output, Weights, ...
  - *MEMORY COST*
    - *Static*: constant at every network run (weights/bias)
    - *Dynamic*: different depending on the network inputs (in/out)



Company Proprietary
GREENWAVES TECHNOLOGIES

GAP*flow* enables DNN inference on Parallel-Ultra Low Power GAP MCUs

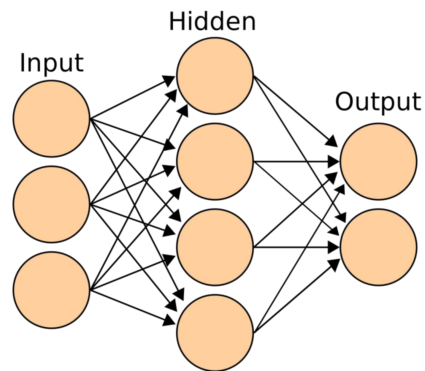


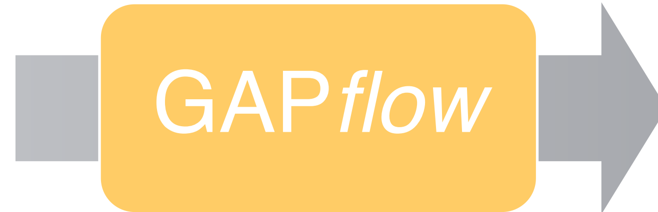
Company Proprietary

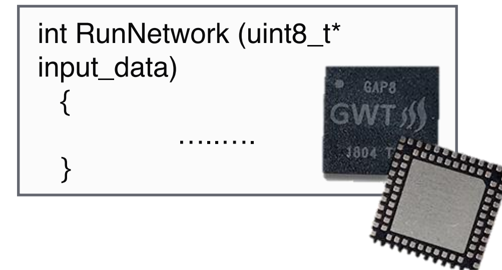
GREENWAVES TECHNOLOGIES

What is GAP*flow*?

MAIN GOAL: turn complex DSP/NN computational graphs into optimized C code for GAP9



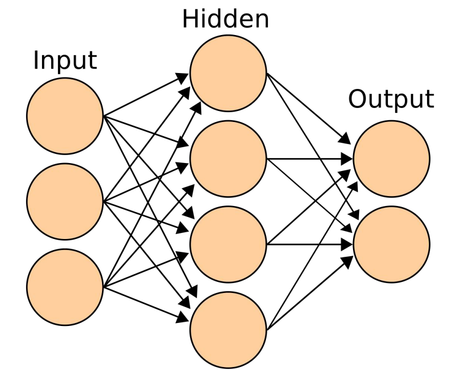




Company Proprietary

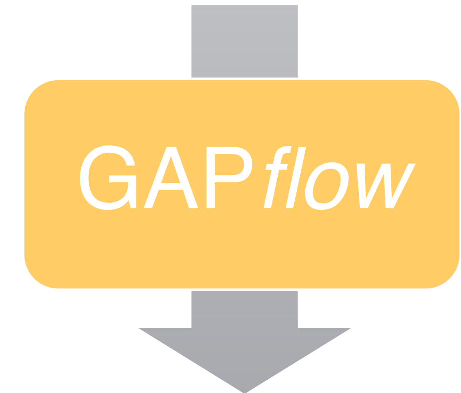
What is GAP*flow*?

MAIN GOAL: turn complex DSP/NN computational graphs into optimized C code for GAP9



GAP9 Graph Optimization

- **Static topology optimization** to minimize number of operations and memory overhead
- **Quantization:** reduce memory usage up to 16x and enable integer only arithmetic when performance is critical



Validate the Solution

- Validate the numeric precision/accuracy of the deployable model in a user-friendly environment (python)

Automatic C Code Generation

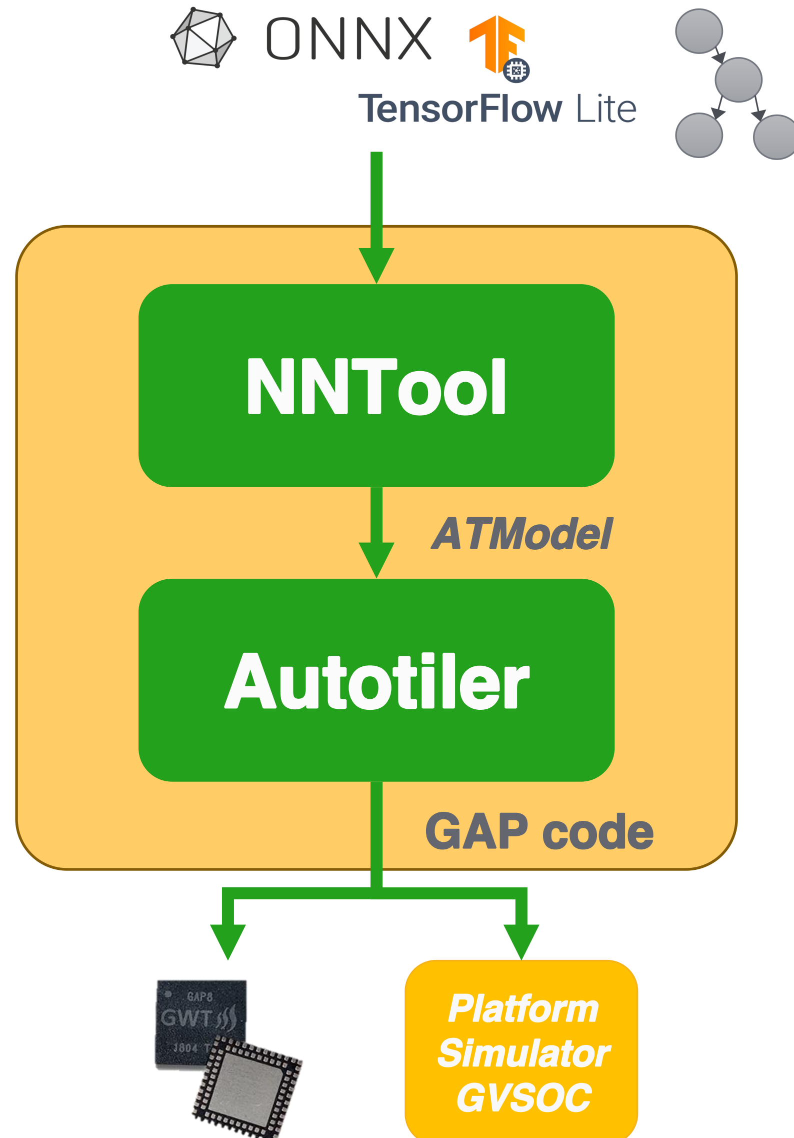
- Map the graph operations into the **Optimized SW library** of GAP9
- **Optimal Memory Management:** automating memory allocation and data transfers

```
int RunNetwork (uint8_t* input_data)
 {
          .........
 }
```


Company Proprietary

GREENWAVES TECHNOLOGIES

GAPflow: Overview



NNTool

- **Static Topology optimizations** (node fusion)
- **Quantization** w/ calibration dataset (optional)
- **Validate** numerically the deployable solution
- Generates an IR of the graph (***ATModel***)

Autotiler

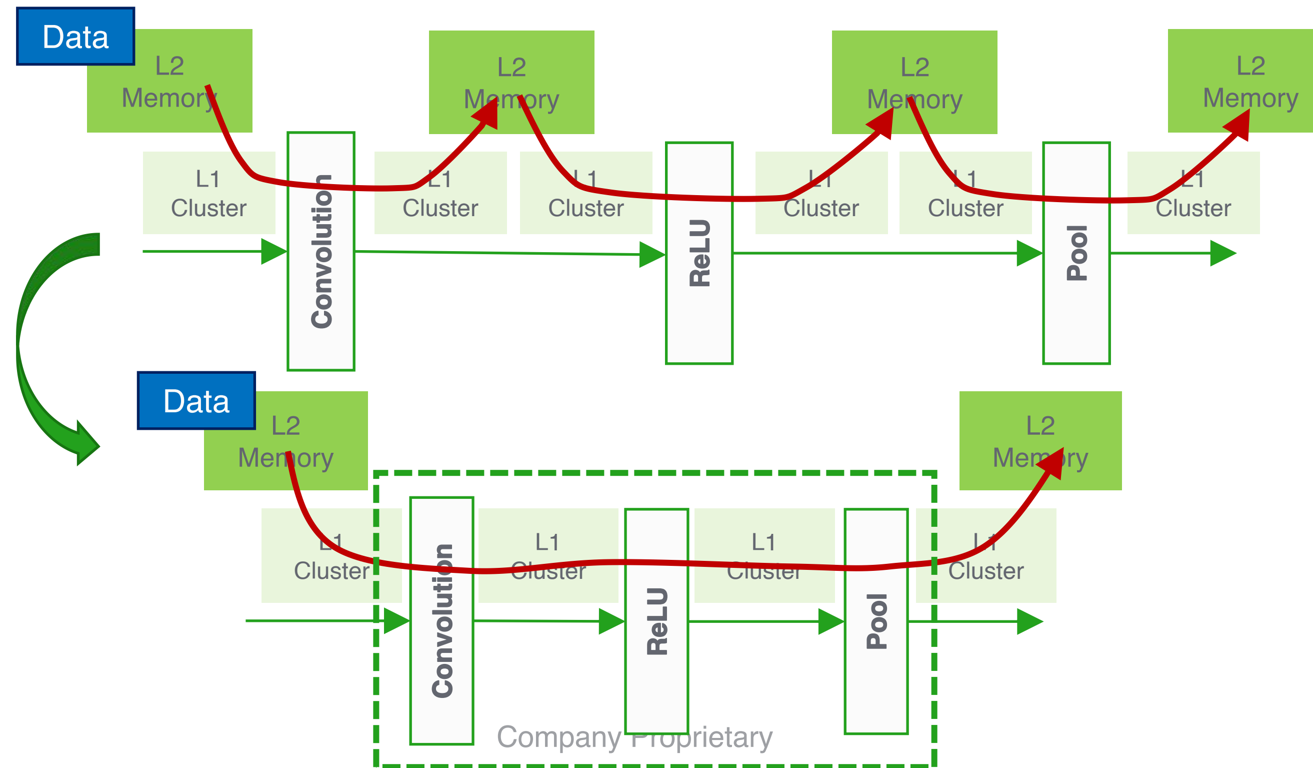
- **Optimizes data movement** across the memory hierarchy
- Computes **optimal tiling sizes**
- Generates **GAP code** with double/triple-buffer mechanism using **optimized SW Library primitives**

Company Proprietary
GREENWAVES TECHNOLOGIES

Topology Optimizations

Minimize number of nodes/edges:

- **Remove** useless reshapes/transpose by moving them accross the graph
- **Layer Fusion:** known sequence of nodes merged together thanks to specialized hand-written backend SW
- **Expressions compiler**: fuses an arbitrary sequence of piecewise/broadcastable operations and dynamically generates GAP C code for it



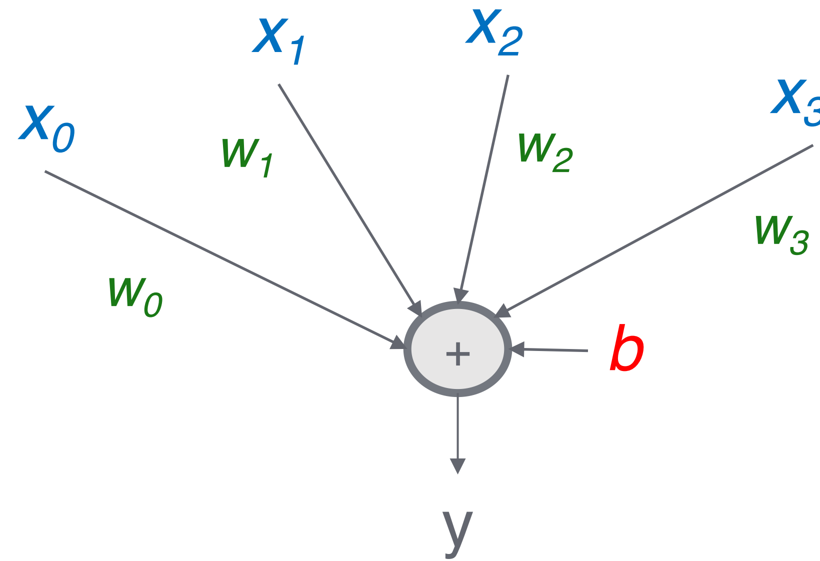
Company Proprietary

GREENWAVES TECHNOLOGIES

Quantization: Background

Convolution Operation

$$\boldsymbol{y} = \sum \boldsymbol{x} \cdot \boldsymbol{w} + \boldsymbol{b}$$



x activation input tensor

w weight parameter tensor

b bias parameter tensor

y activation output tensor

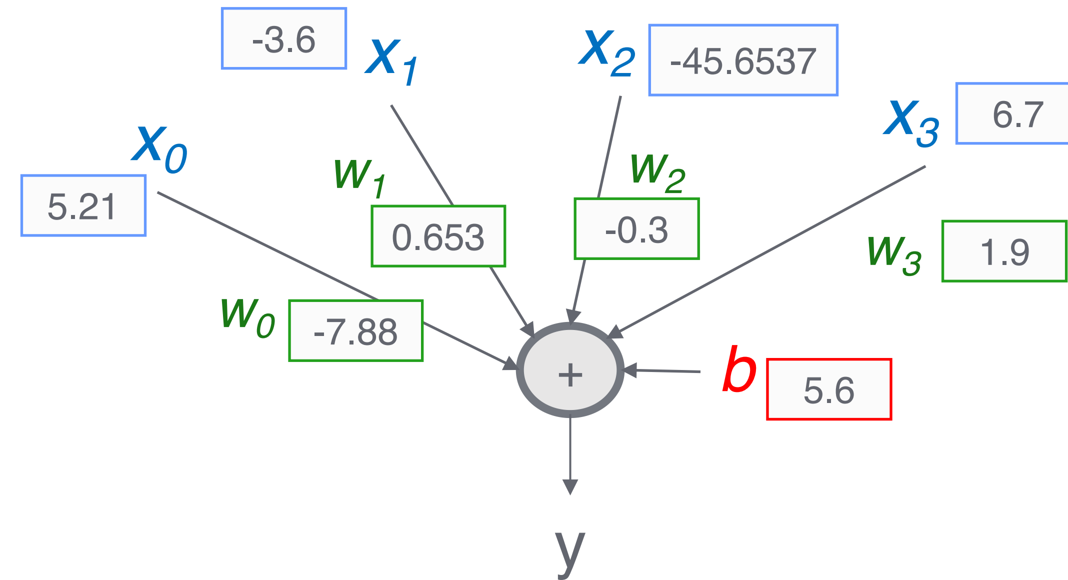
Company Proprietary

GREENWAVES TECHNOLOGIES

Quantization: Background

Convolution Operation

$$y = \sum x \cdot w + b$$



DL frameworks operates with **real numbers**

- Floating-point 32-bit format (FP32)
- Inference requires FPU engines

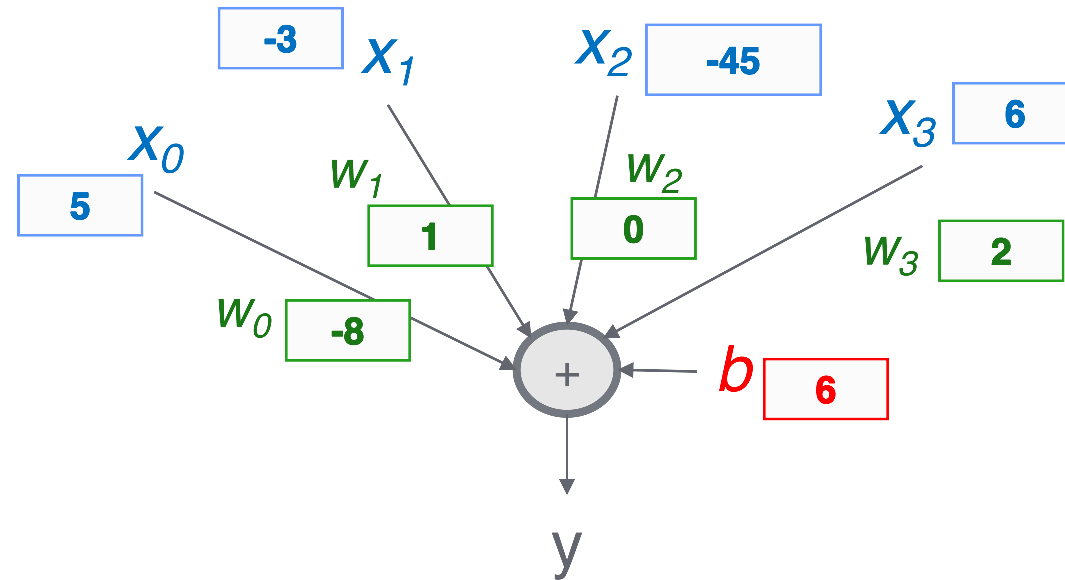
Company Proprietary

GREENWAVES TECHNOLOGIES

Quantization: Background

Convolution Operation

$$y = \sum x \cdot w + b$$



Quantization maps any real value into a set of **integer values**

- **COMPRESSION** to n-bit integer values up to **4x** compression with 8-bit quantization
- **LATENCY** Inference requests integer low-precision operation 8-bit convolution up to **4x** (even more due to the lower BW) faster than FP32 in SW (even more with dedicated HW like NE16)

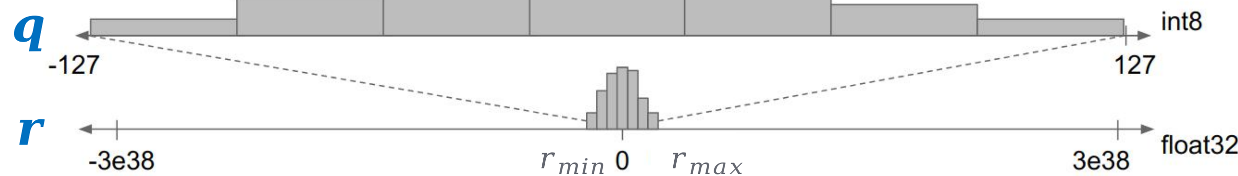
Company Proprietary

GREENWAVES TECHNOLOGIES

Quantization: Background

- 4x memory reduction
- >4x speed up thanks to specialized HW
- But accuracy???

through an affine



$Z = 0$ if symmetric ranges $r_{max} = -r_{min}$

$$r = S\,(q - Z)$$

$$\frac{r_{max} - r_{min}}{2^{nbits}}$$

Convolution Operation

X is a quantized value, x is a real value

$$y = \sum x \cdot w$$

Affine transformation

$$S_y Y = \sum S_x X \cdot S_w W$$

Integer only MACs

$$Y = \frac{S_x S_w}{S_y} \sum X \cdot W$$

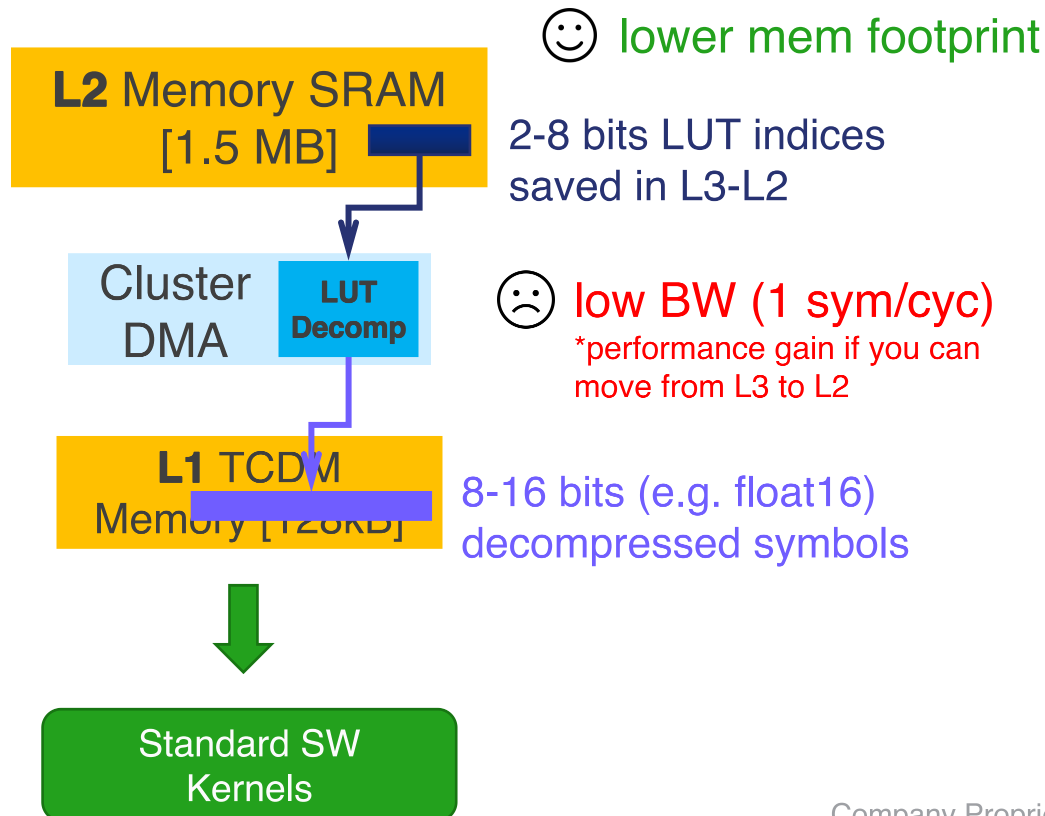
INT8

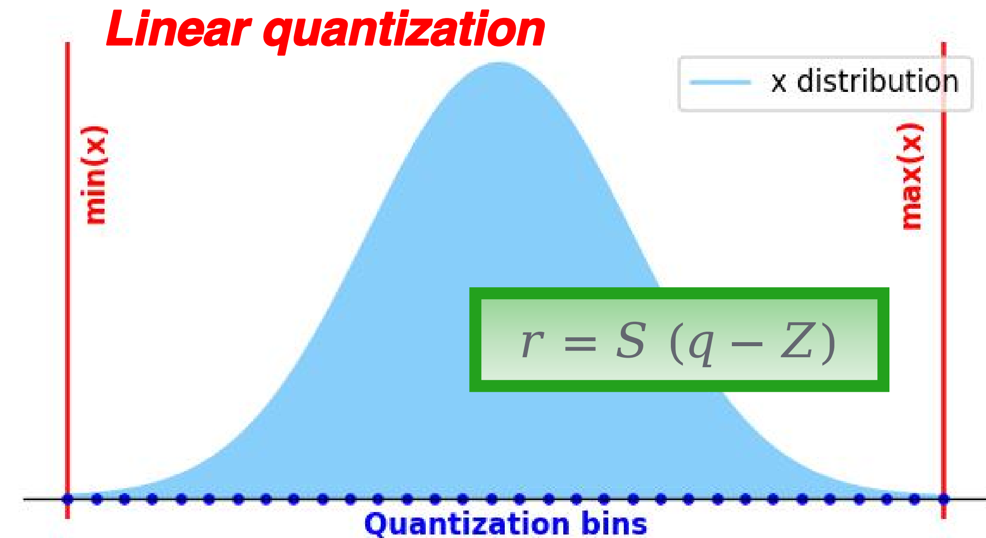
Company Proprietary

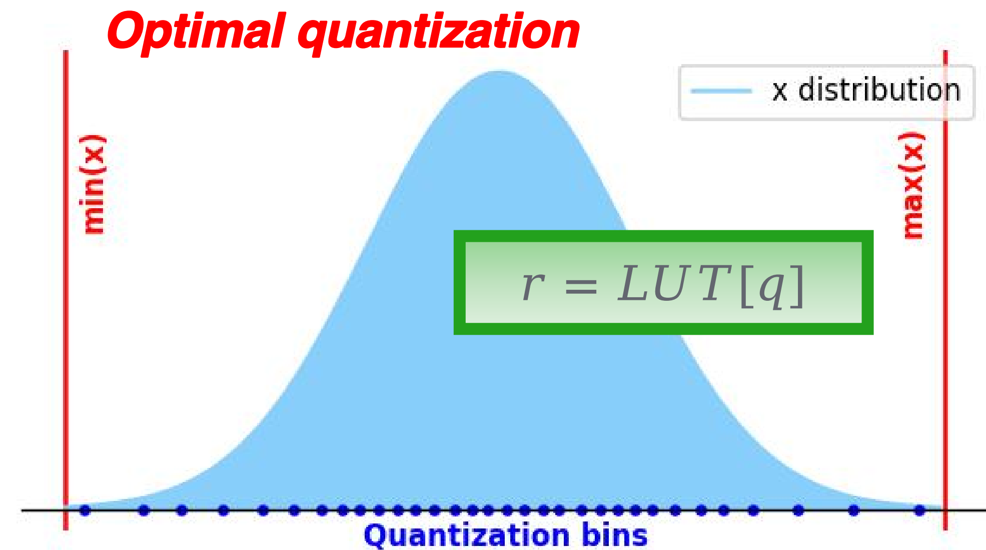
GREENWAVES TECHNOLOGIES

Optimal quantization: Decompressor

- LUT based quantization allows non-uniform quantization
- Clustering (KMeans) can lead to better approximation (higher compression rates)







Company Proprietary

TECHNOLOGIES

Quantization

$$\frac{r_{max} - r_{min}}{2^{nbits}}$$

$Z = 0$ if symmetric ranges $r_{max} = -r_{min}$

$$r = S\,(q - Z)$$

Scheme	Weights	Activations	Affine Transformation	
POW2 (QX.Y)	16-bit (symmetric, PerTensor)	16-bit (symmetric, PerTensor)	$r = q * 2^{-N}$	(1)
int8	2-8bits (symmetric, PerChannel)	8- or 16bit (asym. PerTensor)	$r = (q - Z) * S$	(1)
Float16	16-bit (bfloat16/f16ieee)	16-bit (bfloat16/f16ieee)	-	(2)
LUT	2-8bits (LUT)	(not supported)	$r = LUT[q]$	

(1) Quantization Ranges obtained:

- From a graph quantized with third-paties tools: *Onnx+NNCF* or *TFLite* quantization
- Using GAP Nntool Post-Training Quantization with a calibration dataset

(2) Cast from FP32 to FP16 done by GAP NNtool. No Calibration dataset required. Typically, lossless.

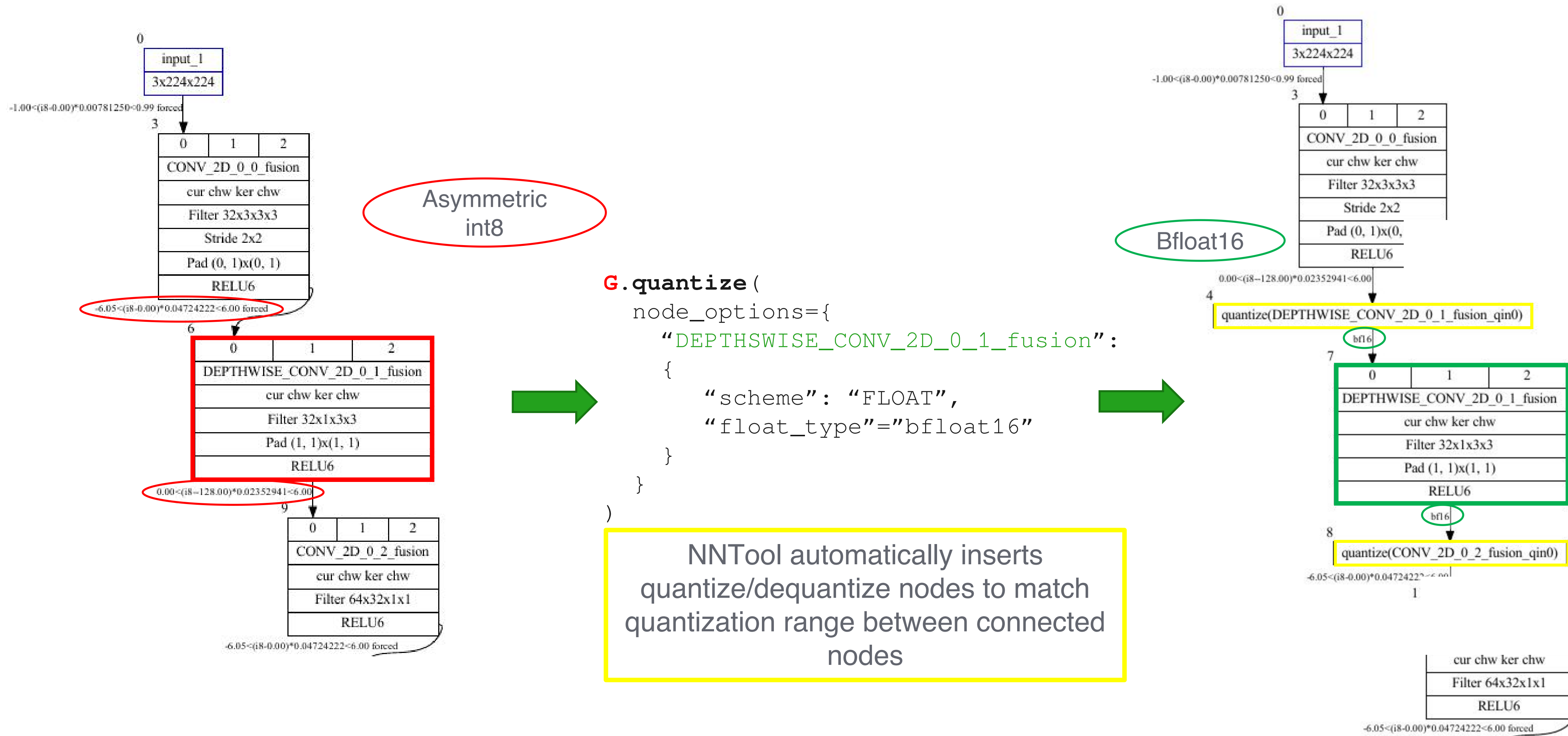
* TFLite quantization, https://arxiv.org/abs/1712.05877

Company Proprietary

GREENWAVES TECHNOLOGIES

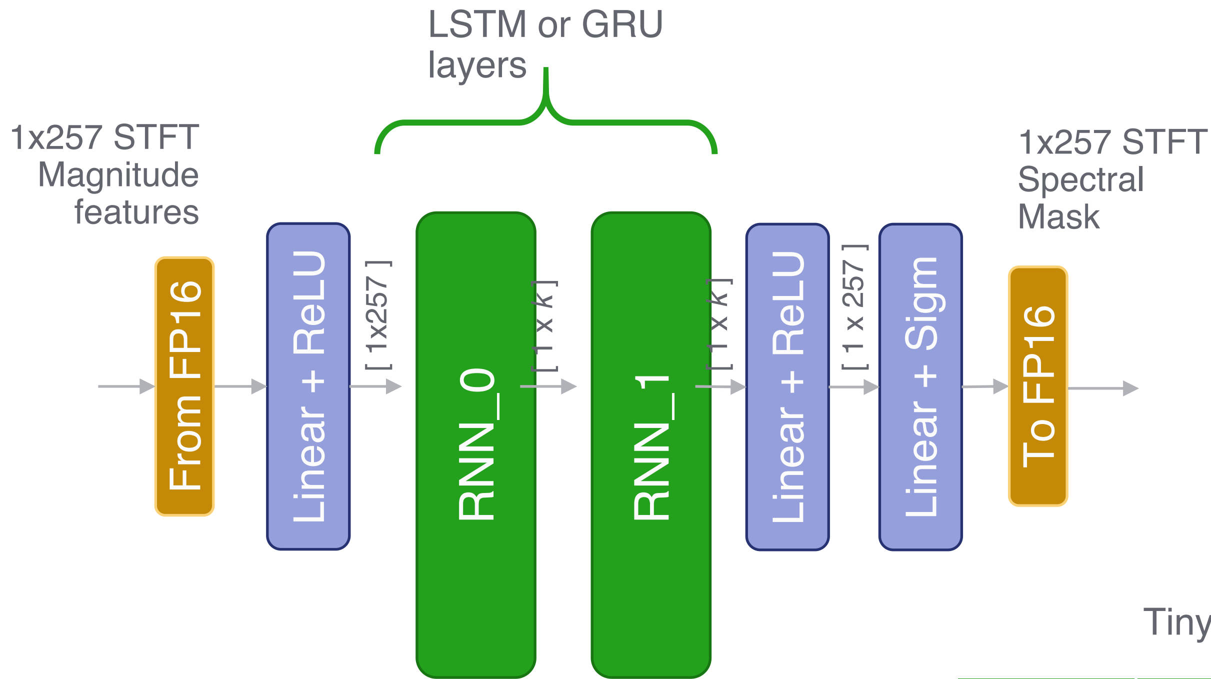
Mixed-Precision Quantization

GAP NNTool enables layer-wise selection of quantization scheme and number of bits



GREENWAVES TECHNOLOGIES

Mixed-Precision Quantization: Use case



	LSTM256	GRU256	LSTM128	GRU128
k	256	256	128	128
RNN_0	LSTM(257,256)	GRU(257, 256)	LSTM(257,128)	GRU(257, 128)
RNN_1	LSTM(257,256)	GRU(257, 256)	LSTM(128 128)	GRU(128, 128)
Param	1.24 M	0.985 M	0.493 M	0.411 M
% rnn params	84%	80%	66.5%	59.8%

The majority of the weights are due to RNN layers!

TinyDenoiser models trained on Valentini dataset: FP32 baseline

Quant Type	LSTM256			GRU256			LSTM128			GRU128		
	PESQ	STOI	Mem	PESQ	STOI	Mem	PESQ	STOI	Mem	PESQ	STOI	Mem
FP32	**2.79**	**0.94**	4.75	**2.78**	**0.94**	3.76	**2.76**	**0.94**	1.88	**2.69**	**0.94**	1.56
FP16	**2.79**	**0.94**	2.37	**2.78**	**0.94**	1.88	**2.76**	**0.94**	0.94	**2.69**	**0.94**	0.78
INT8	**2.42**	**0.92**	1.18	**2.48**	**0.93**	0.93	**2.51**	**0.92**	0.47	**2.36**	**0.93**	0.39

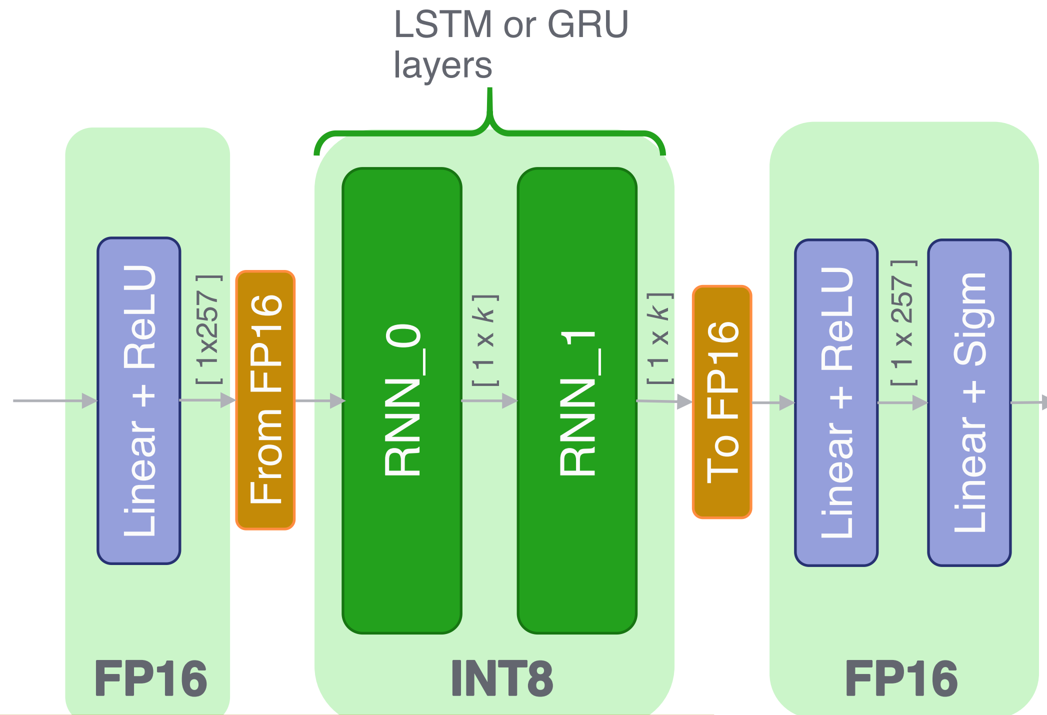
INT8 is lightweight but **LOSSY**

- avg PESQ loss: **-0.3**, STOI loss: **0.015**
- **2x** mem compression

Company Proprietary

GREENWAVES TECHNOLOGIES

Mixed-Precision Quantization: Use case



	LSTM256	GRU256	LSTM128	GRU128
k	256	256	128	128
RNN_0	LSTM(257,256)	GRU(257, 256)	LSTM(257,128)	GRU(257, 128)
RNN_1	LSTM(257,256)	GRU(257, 256)	LSTM(128 128)	GRU(128, 128)
Param	1.24 M	0.985 M	0.493 M	0.411 M
% rnn params	84%	80%	66.5%	59.8%

The majority of the weights are due to RNN layers!

TinyDenoiser models trained on Valentini dataset: FP32 baseline

No need for expensive QAT !!!

Quant Type	LSTM256			GRU256			LSTM128			GRU128		
	PESQ	STOI	Mem	PESQ	STOI	Mem	PESQ	STOI	Mem	PESQ	STOI	Mem
FP32	2.78	0.94	4.75	2.78	0.94	3.76	2.76	0.94	1.88	2.69	0.94	1.56
FP16	2.78	0.94	2.37	2.78	0.94	1.88	2.76	0.94	0.94	2.69	0.94	0.78
INT8	2.42	0.92	1.18	2.48	0.93	0.93	2.51	0.92	0.47	2.36	0.93	0.39
MixFP16-INT8	2.73	0.93	1.37	2.72	0.94	1.13	2.69	0.93	0.67	2.63	0.94	0.55

MixedFP16-INT8 present low accuracy degradation (**PESQ: 0.06, STOI: <0.01**) while **1.4-1.7x** mem compression vs FP16

Company Proprietary

GREENWAVES TECHNOLOGIES

Fast prototyping

- Deploy in few lines
- Complete control of:
  - Graph manipulations
  - Quantization
  - Testing
  - Deployment settings
- Check of consistency

Prepare the graph for deployment

```
G = NNGraph.load_graph("model.onnx")
G.adjust_order()
G.fusions("scaled_match_group")
stats = G.collect_statistics(repr_dataset())
G.quantize(stats, graph_qopts, node_qopts)
```

Test the final graph in python

```
for data in test_data:
  outs = G.execute(data, dequantize=True)
  ok = check(outs)
```

If not satisfied: requantize

```
if not ok:
  # update node_qopts and start over
```

Check consistency on target

```
res = G.execute_on_target(data, model_settings)
check_equal(res.output_tensors, outs)
res.print_basic_mem_infos()
res.print_performance()
...
```

Validation of the solution

- Check deployed model accuracy:
  ☹ On-device: use directly the final platform to test the accuracy (**very slow**)
  ☺ NNTool: bit-accurate numpy backend, the user can test accuracy in a python environment without need of device (**fast**)

```
G = NNGraph.load_graph(file_path)
stats = G.collect_statistics(calibration_dataset)
G.quantize(stats, quantization_options)
G.adjust_order()
G.fusions("scaled_match_group")
# Ready for inference
acc1 = 0
for in_data, target in test_dataset:
    outq = G.execute(in_data, quantize=True)
    acc1 += np.argmax(outq[-1][0]) == target
```

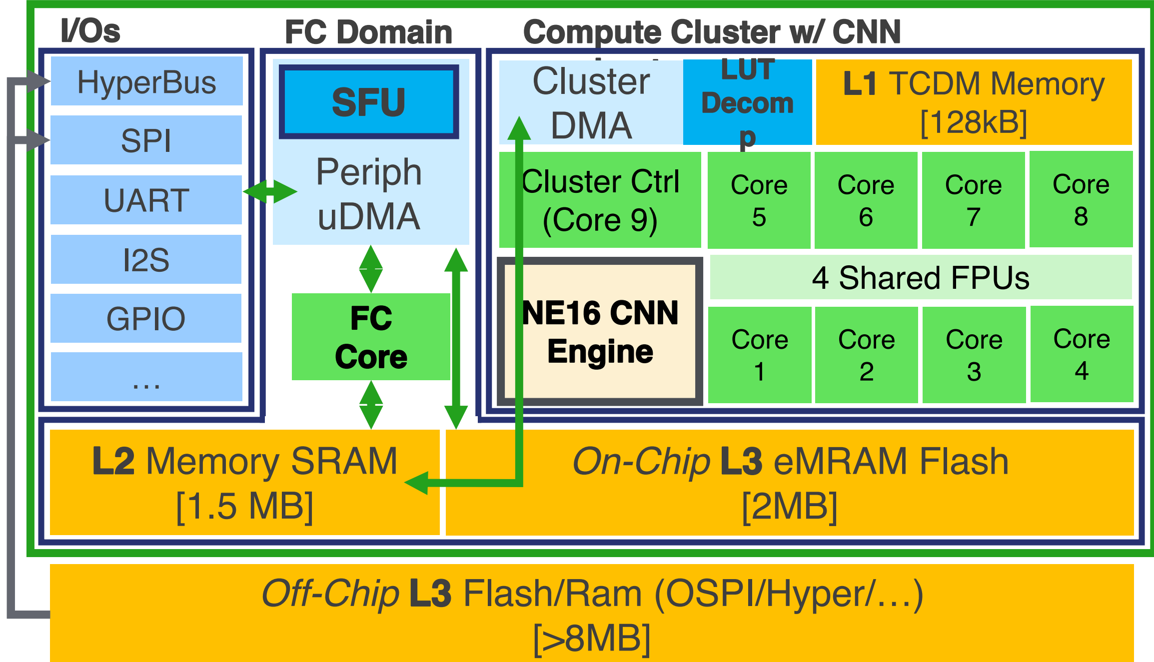
Prepare your model for deployment (quantization+graph manipulation)

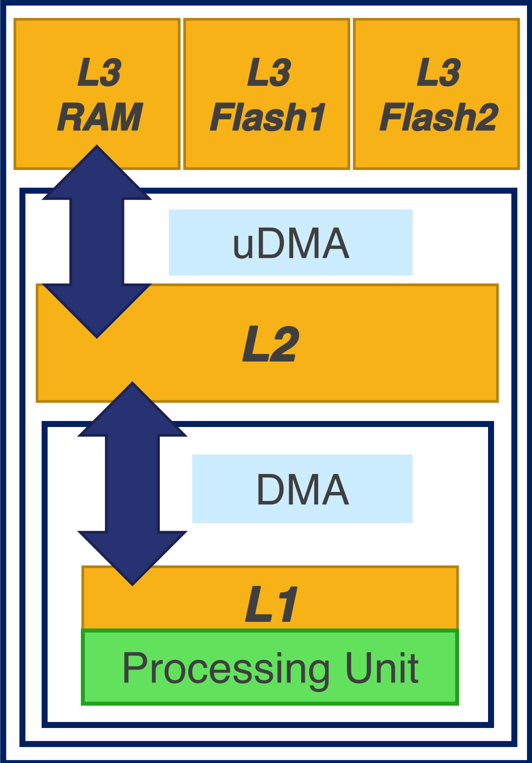
Run inference and check results

GAP Autotiler

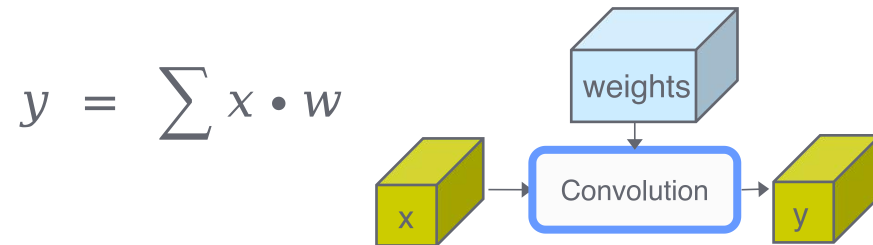
GAP does not have a data CACHE:
- Silicon Area
- Energy Efficiency
- NN/DSP algo have predictable data traffic

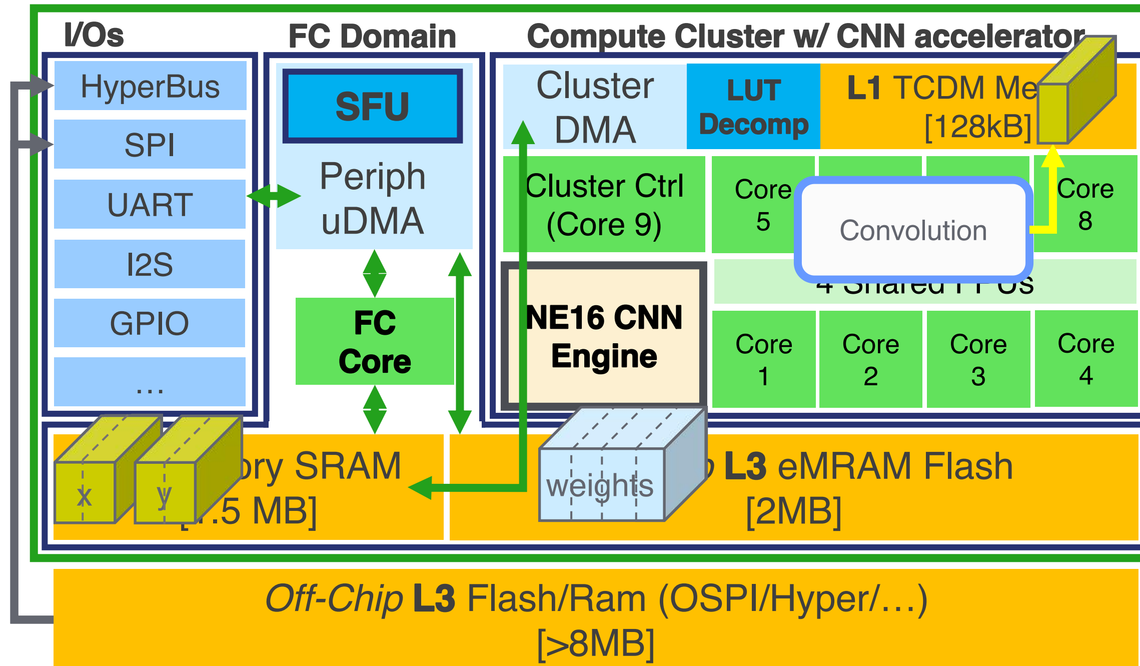
Generate C code for all data movement at compile time





Autotiler User Kernel: NN Node to the GAP architecture





Computation dataflow

Ahead of time

Store data (parameters & input vector) in L2 (or L3)

At run time, for any computational node:

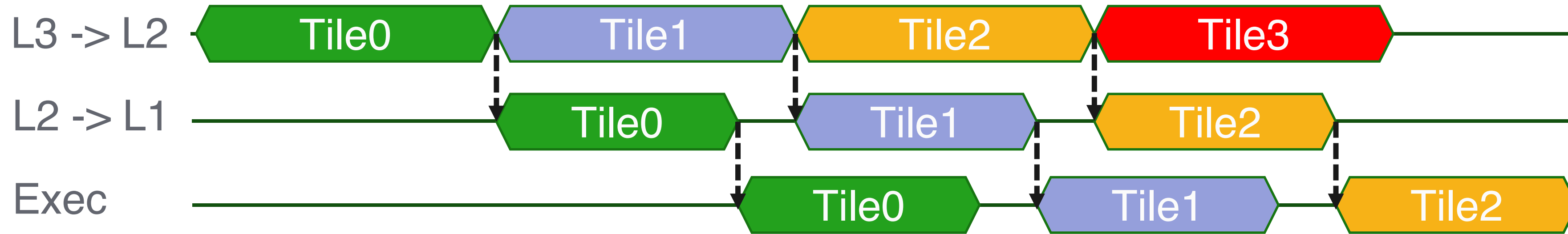
Partition and Load data (parameters & input tensors) to L1

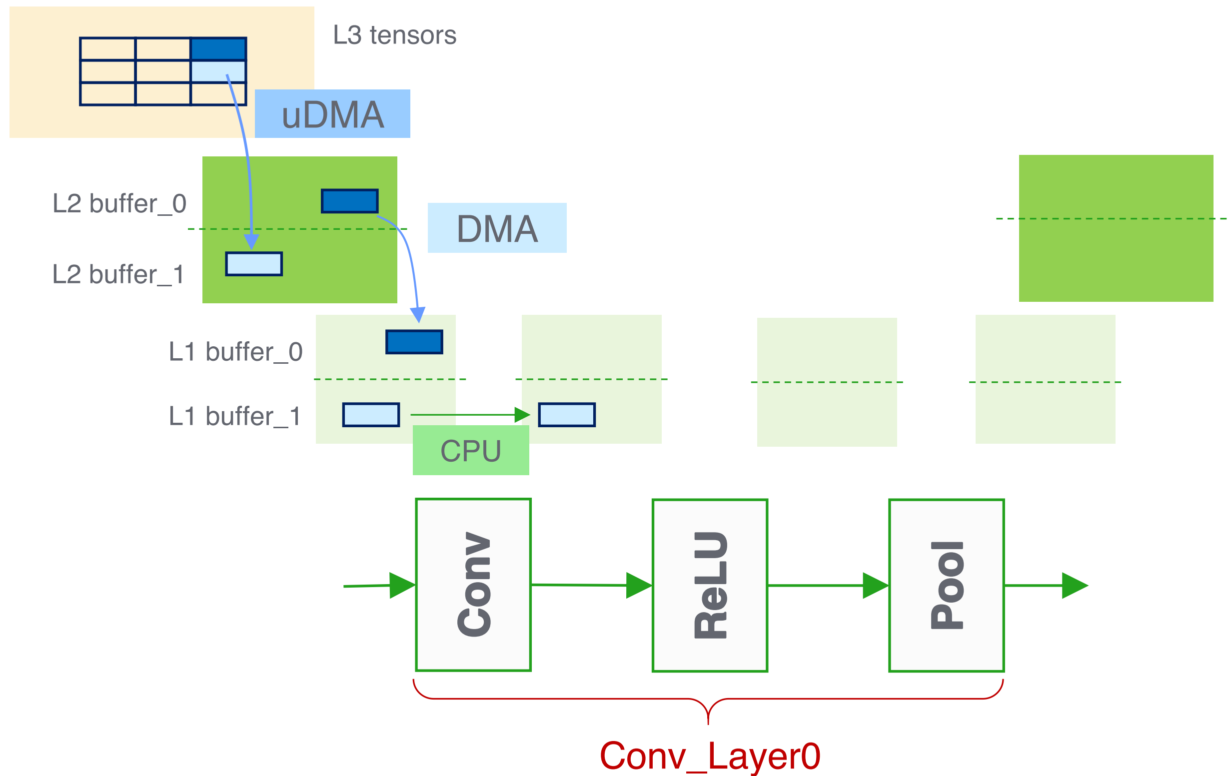
Run data-parallel/CNN Engine computation

Store data (output tensors) back in L2 (or L3)

```
static void Conv_Layer0
(
    signed char * In,       // input L2 vector
    signed char * Weights,  // input L2 vector
    signed char * Bias,     // input L2 vector
    signed char * Out,      // output L2 vector
){

    //tile sizes of In, Weights, Bias computed offline
    //L1 buffer allocated to handle double buffering
    //  two L1 memory buffers for double buffering

    DMA load first tiles to L1 memory buffer

    for any tile of In, Weights, Bias tensors:

      DMA load next tiles to L1 memory buffer

      ParConv() on L1 tile
      ParReLU() on L1 tile
      ParPool() on L1 tile

      DMA write results (Out) to L2

}
```

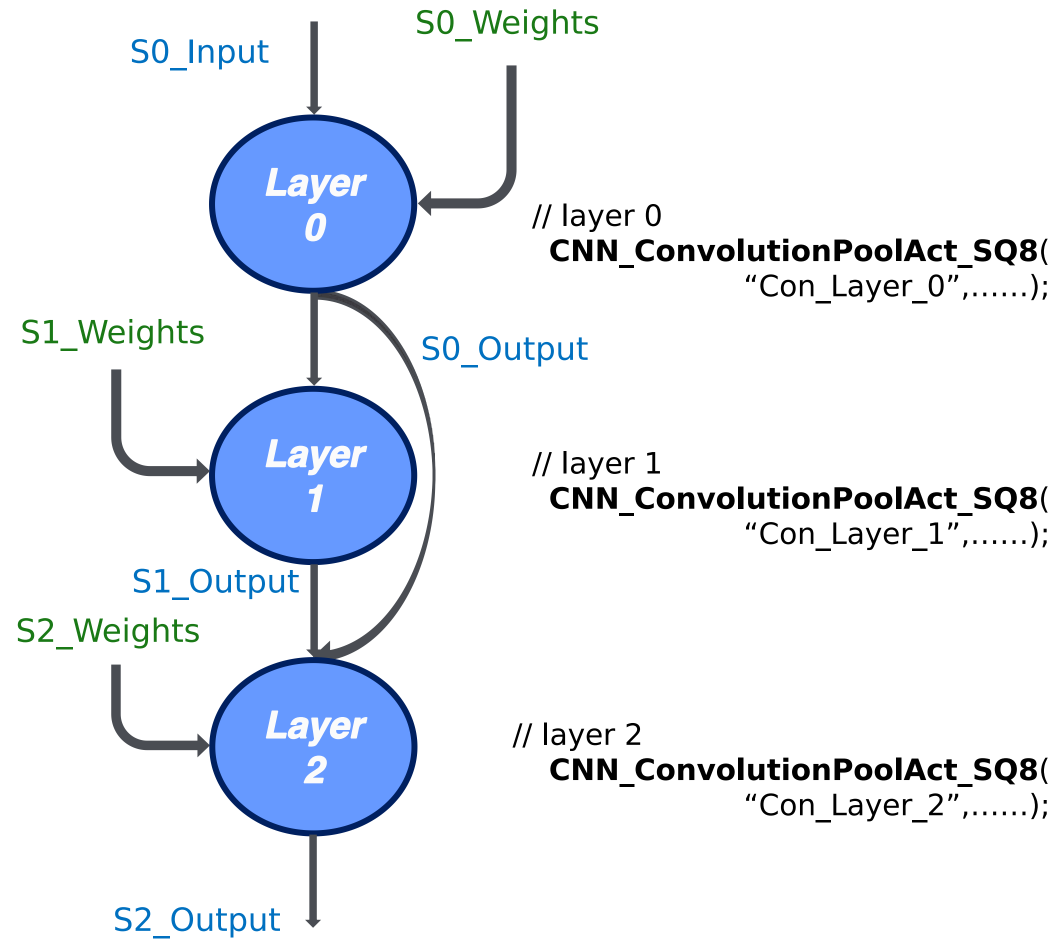
Mapping a NN Node to the GAP HW/SW architecture






```
static void Conv_Layer0
(
    signed char * In,       // input L3 vector
    signed char * Weights,  // input L3 vector
    signed char * Bias,     // input L3 vector
    signed char * Out,      // output L3 vector
){

    //tile sizes of In, Weights, Bias computed offline
    //L1 buffer allocated to handle double buffering
    //  two L1 memory buffers for double buffering

    uDMA load first tiles to L2 memory buffer

    DMA load first tiles to L1 memory buffer

    for any tile of In, Weights, Bias tensors:

        uDMA load next next tiles to L2 memory buffer

        DMA load next tiles to L1 memory buffer

        ParConv() on L1 tile
        ParReLU() on L1 tile
        ParPool() on L1 tile

        DMA write results (Out) to L2

        uDMA write prev results to L3
}
```

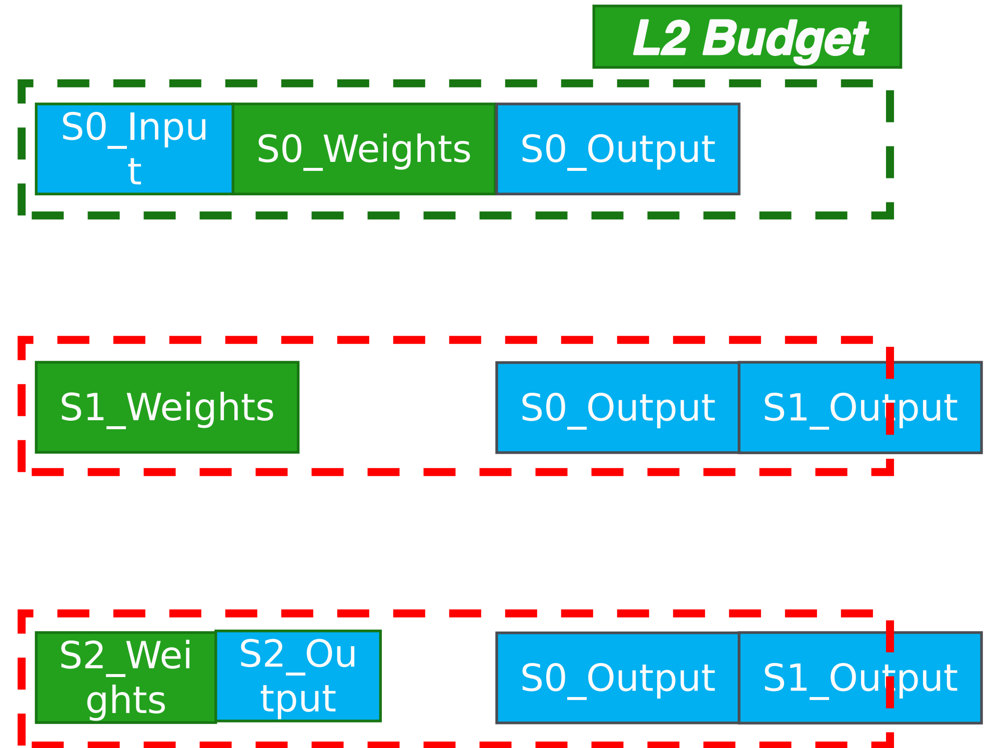
Autotiler Graph: Static allocation



// layer 0
CNN_ConvolutionPoolAct_SQ8(
"Con_Layer_0",……);

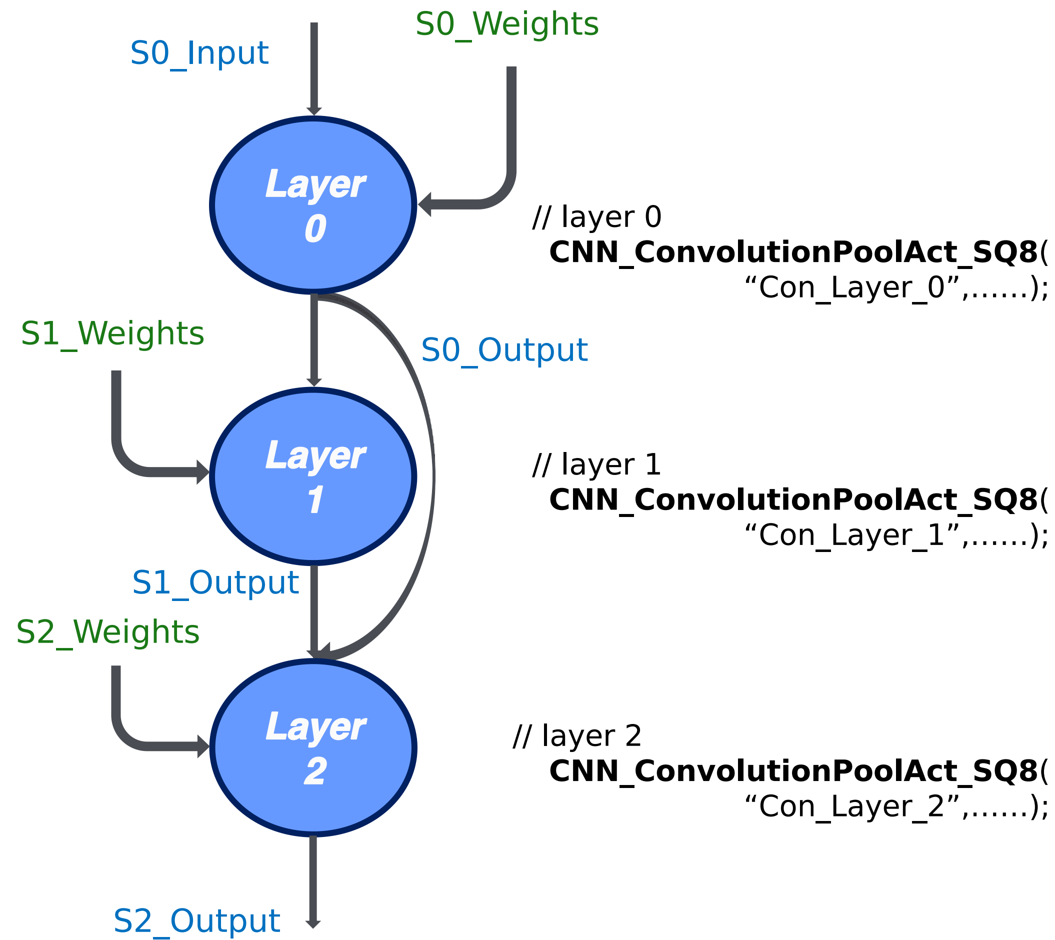
// layer 1
CNN_ConvolutionPoolAct_SQ8(
"Con_Layer_1",……);

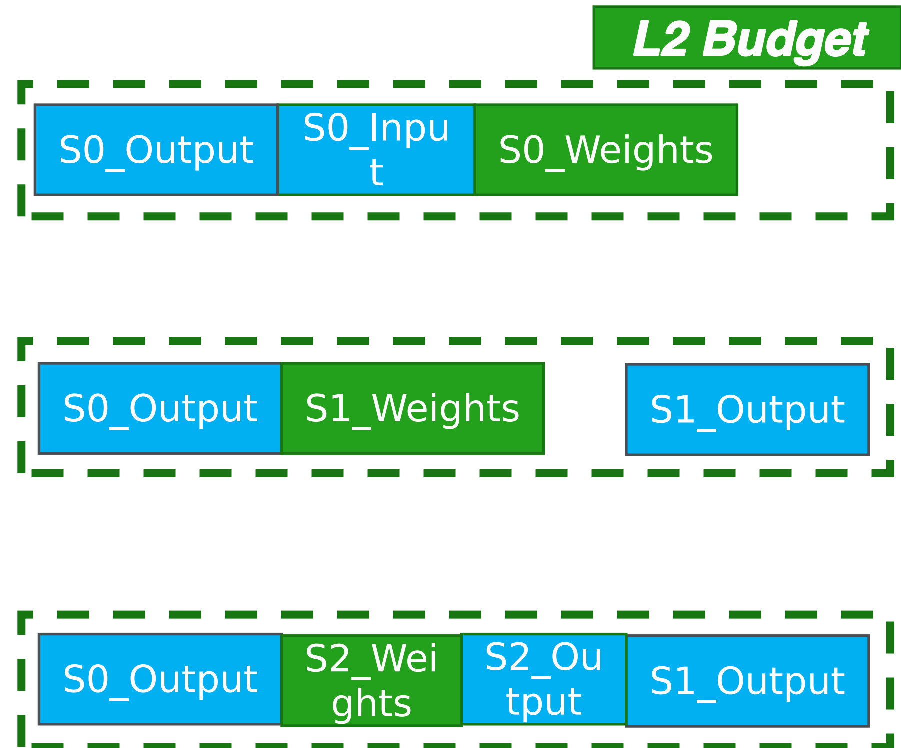
// layer 2
CNN_ConvolutionPoolAct_SQ8(
"Con_Layer_2",……);



Company Proprietary
GREENWAVES TECHNOLOGIES

Autotiler Graph: Static allocation

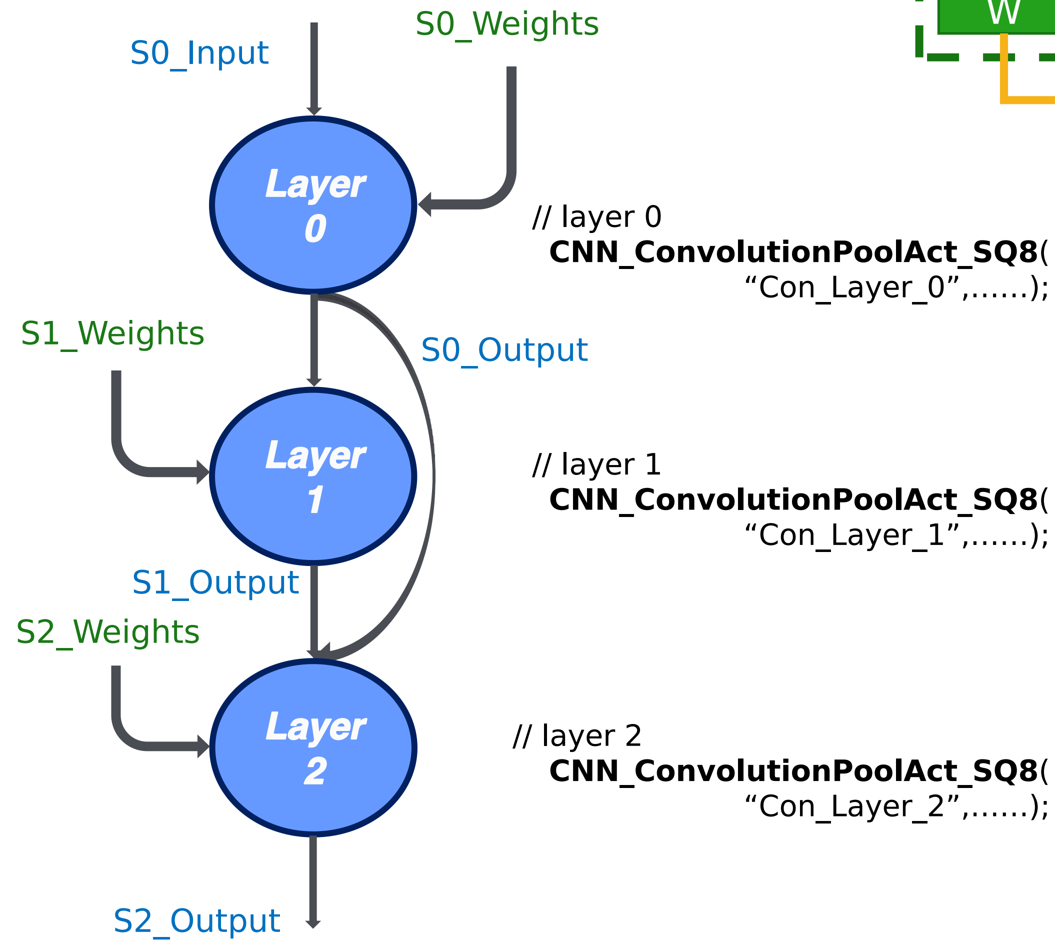


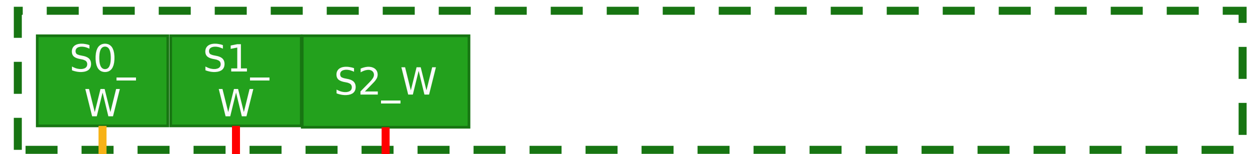


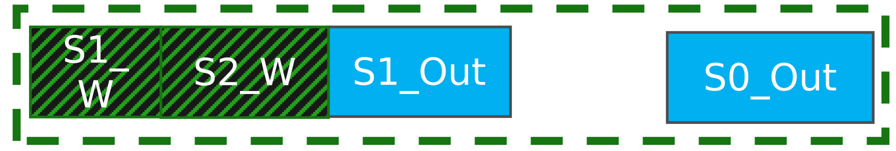
Company Proprietary

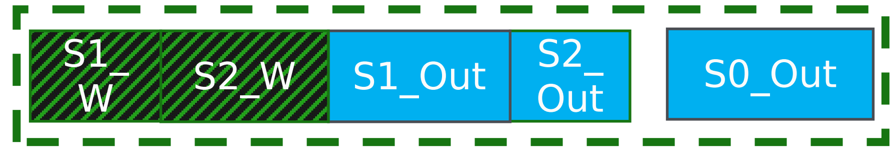
GREENWAVES TECHNOLOGIES

Autotiler Graph: Static allocation (tensor promotion)

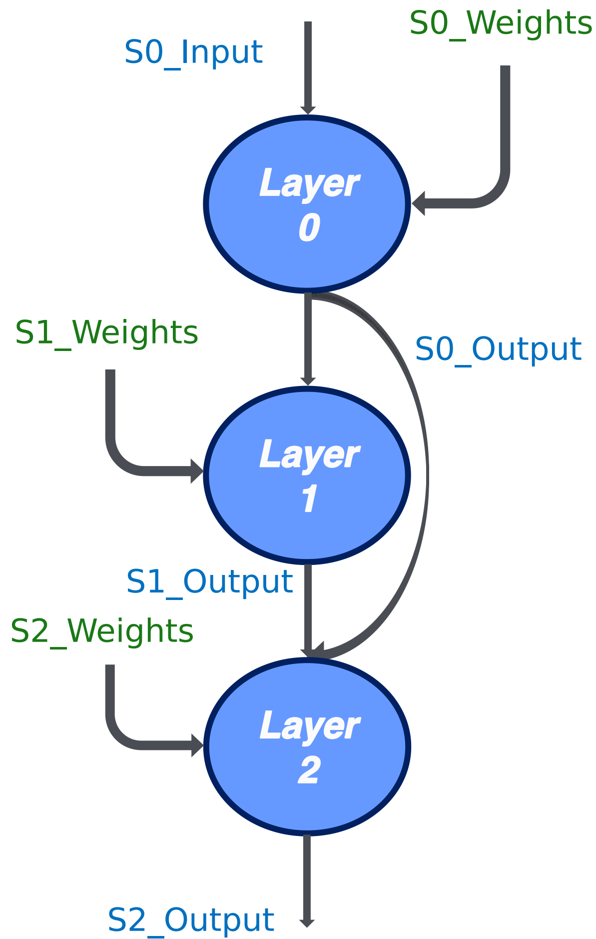






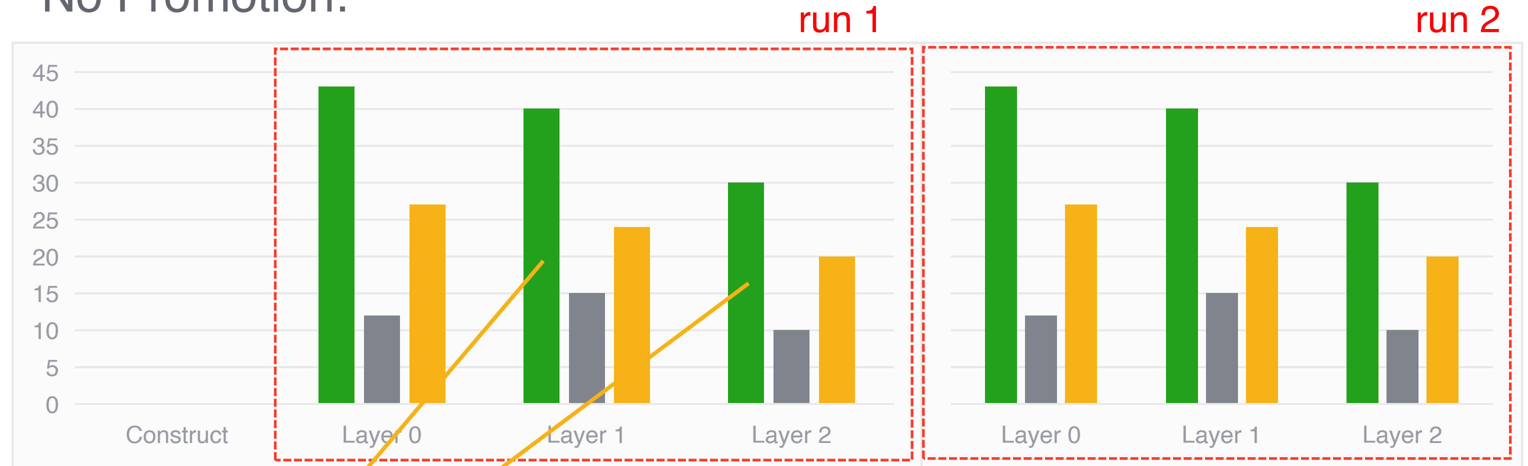


2. Graph mode: Mem allocation inter-layer

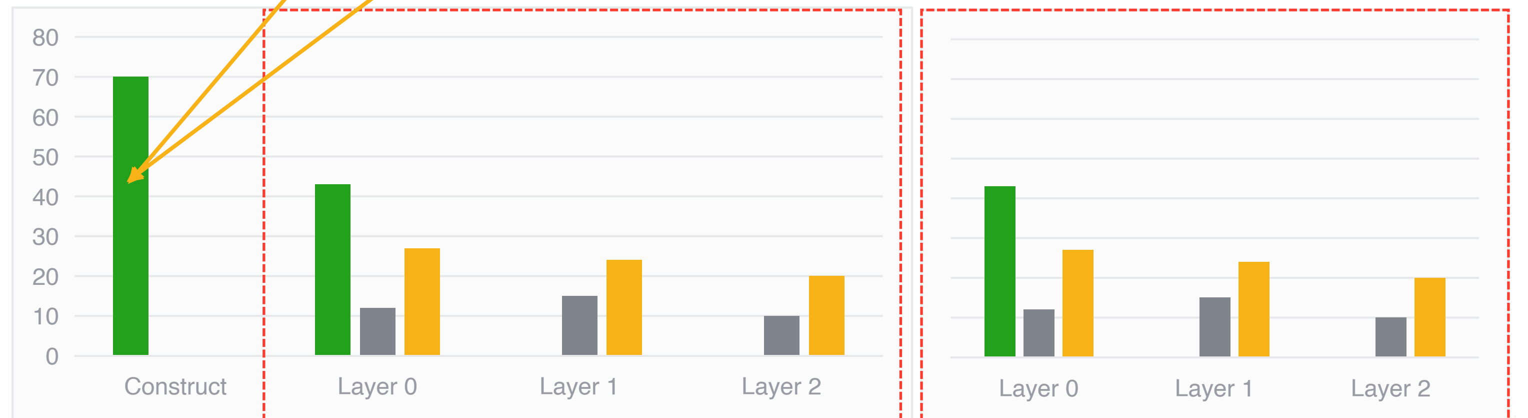


Energy Weights L3->L2
Energy L2->L1
Energy Execution

No Promotion:

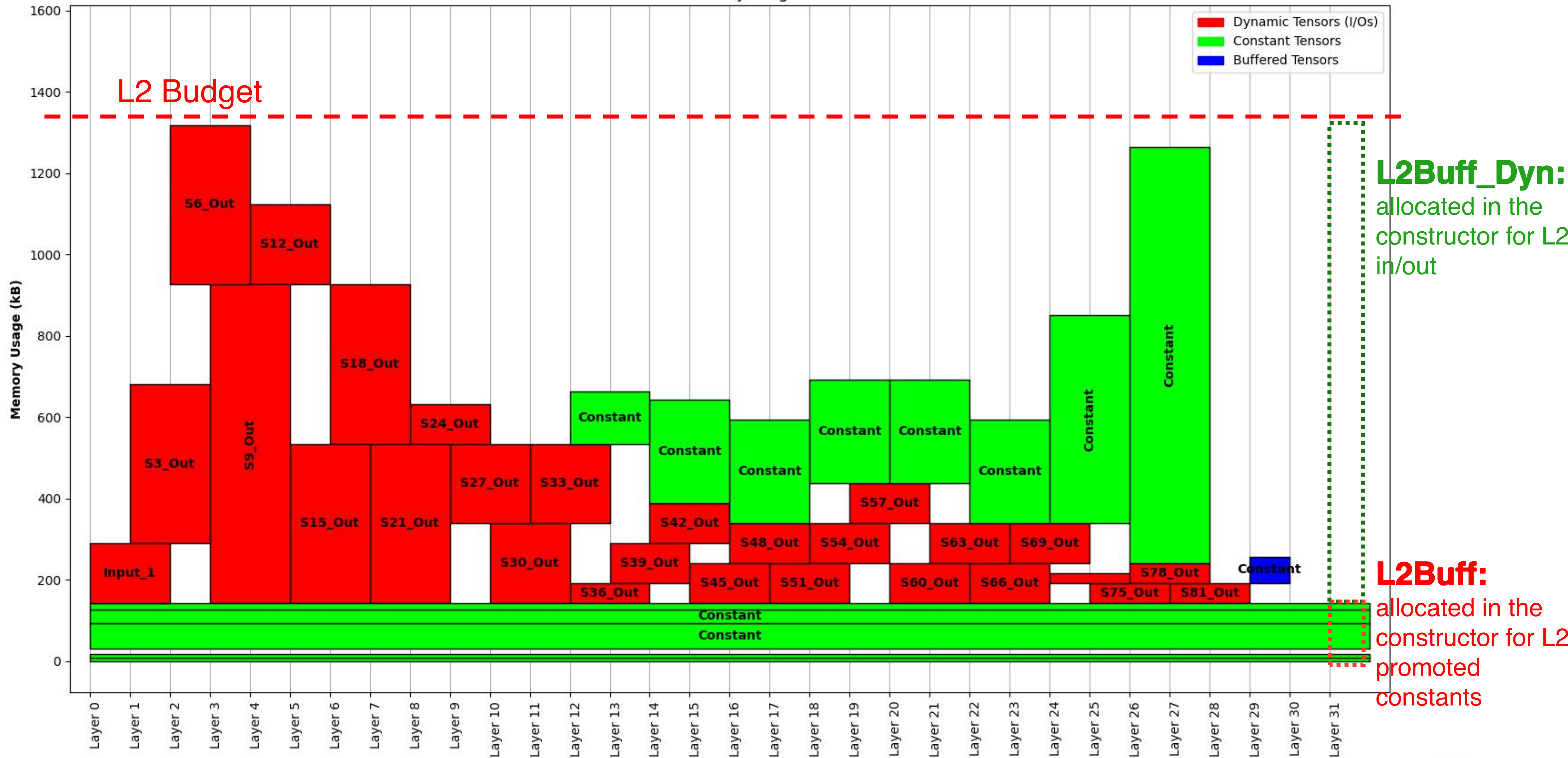


Static Promotion of constants



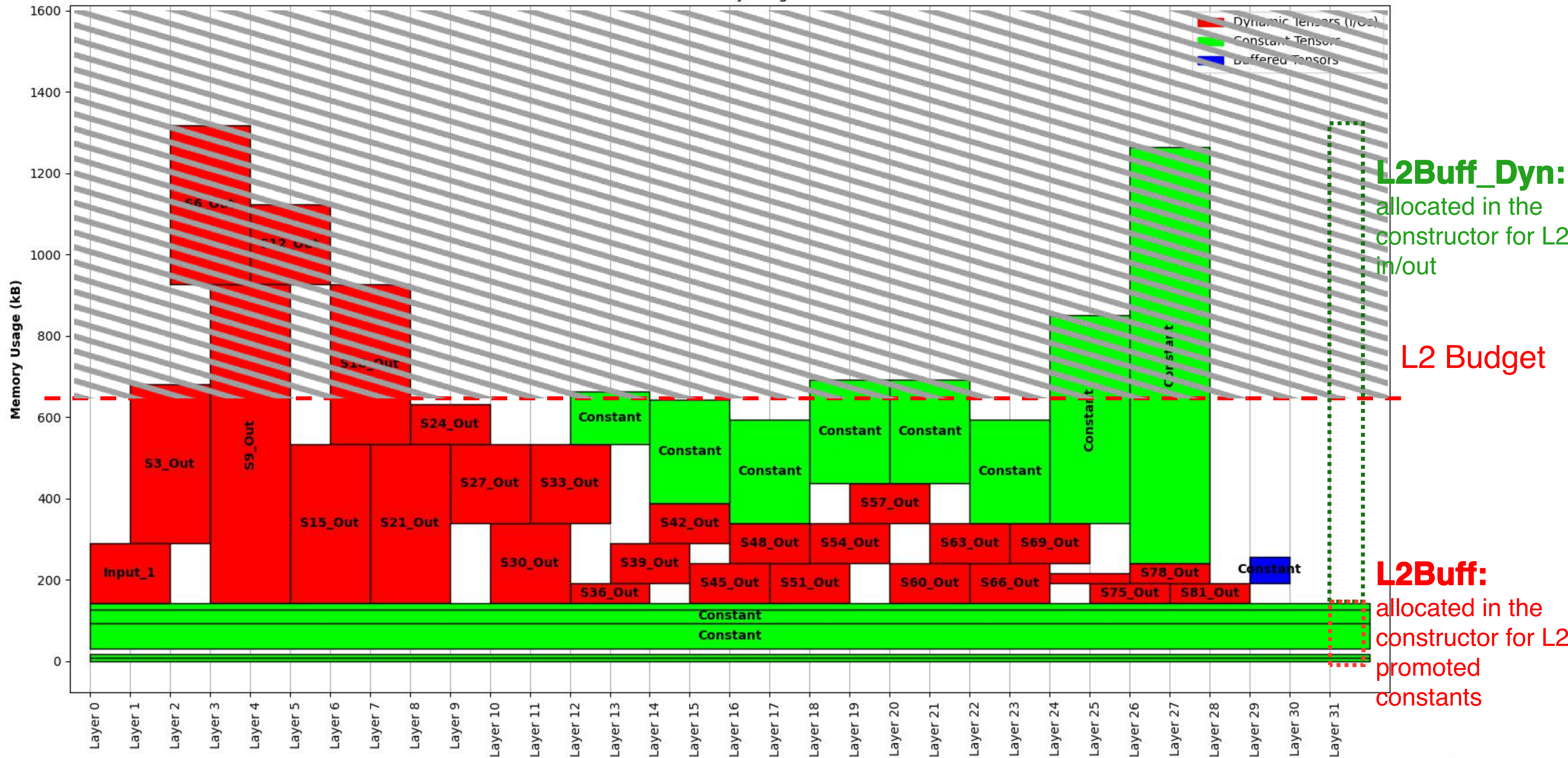
Company Proprietary

MobilenetV1 L2 RAM Graph Memory allocation Strategy



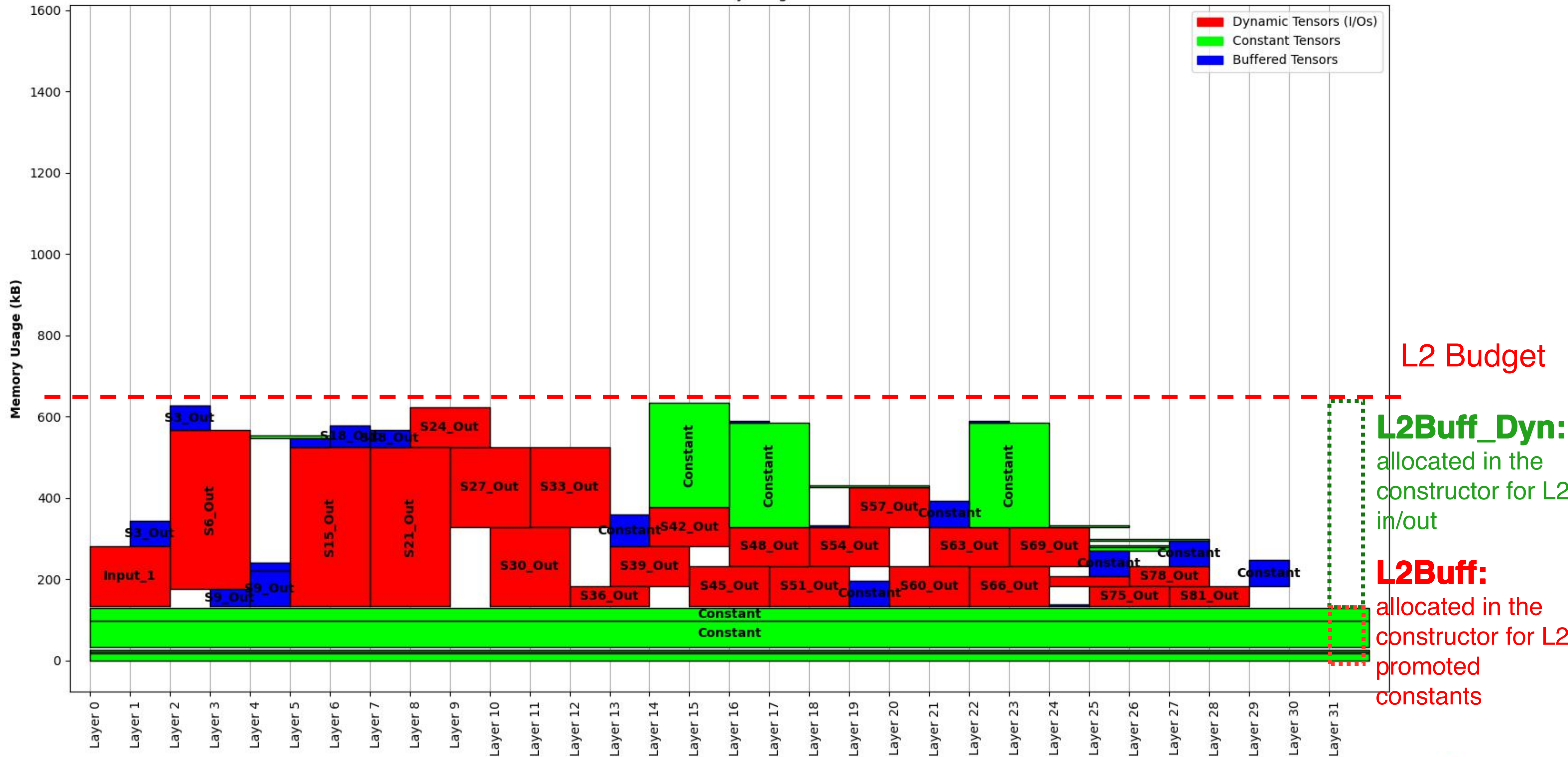
GREENWAVES TECHNOLOGIES

MobilenetV1 L2 RAM Graph Memory allocation Strategy



GREENWAVES TECHNOLOGIES

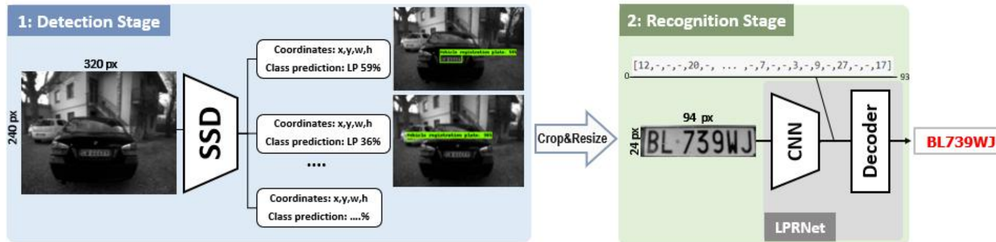
MobilenetV1 L2 RAM Graph Memory allocation Strategy



GREENWAVES TECHNOLOGIES

Enabling complex NN on energy constrained devices

Multi NN – Licence Plate

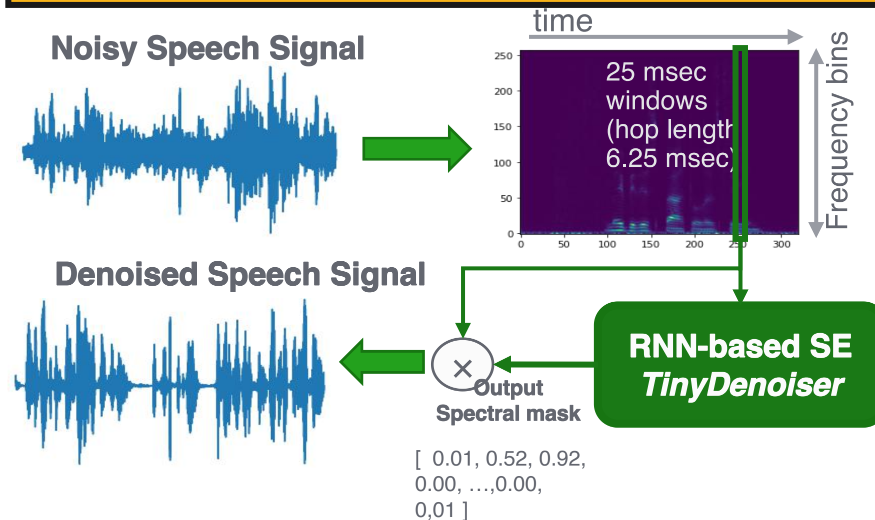


https://ieeexplore.ieee.org/abstract/document/9401730

Smart glasses - object detection

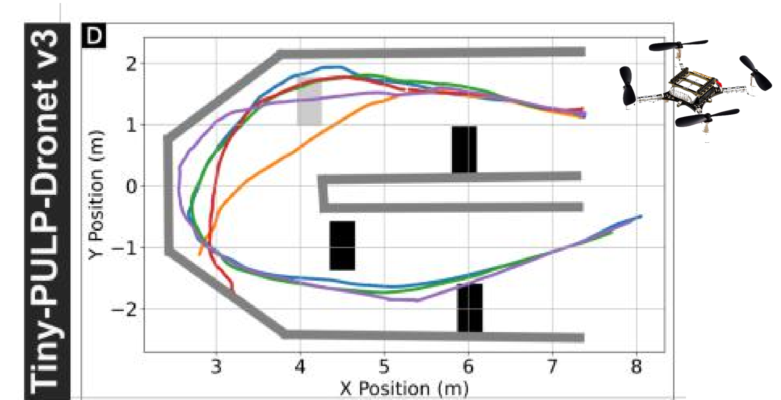
https://arxiv.org/ftp/arxiv/papers/2311/2311.01057.pdf

DSP/NN Mixed-Precision – Denoiser



https://arxiv.org/pdf/2210.07692

Nano drones autonomous navigation



https://arxiv.org/pdf/2407.12675

GREENWAVES TECHNOLOGIES

TinyML: best performance HW and SW

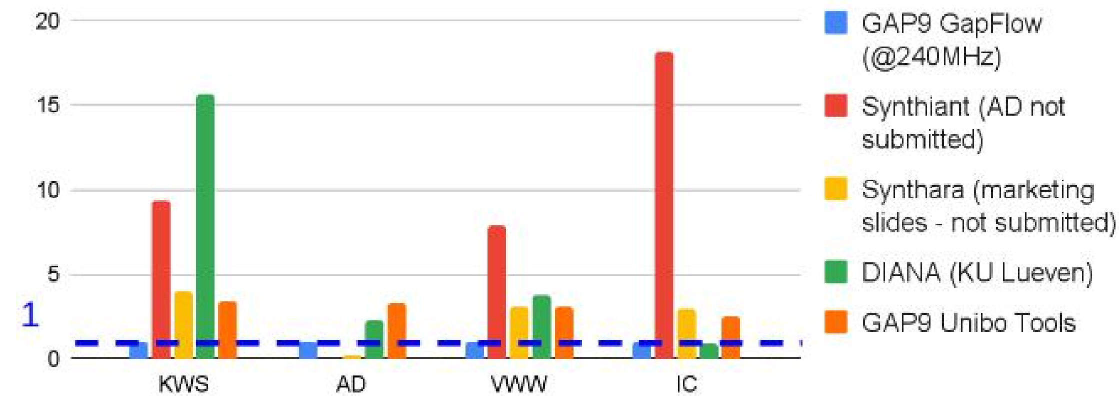
Latency:

- 7-18x faster than second best submitted result (Synthiant)
- Up to 3x faster than industry leader yet to be available (Synthara)
- Up to 15x faster than latest research chips (DIANA)
- 2-4x faster than other publicly available SW libraries/tools with same HW (GAP9)

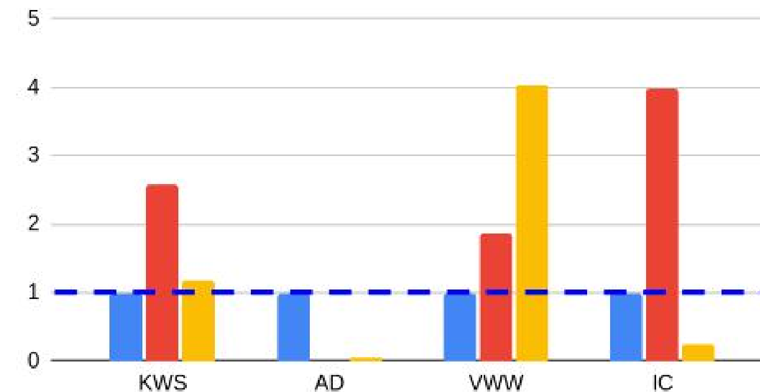
Energy:

- 2-5x lower energy than second best submitted result
- Up to 4x better than not yet available HW
- *DIANA and GAP9 Unibo data not available for energy

Latency normalized to GAP9 GAPflow



Energy normalized to GAP9 GAPflow



Company Confidential – Shared under NDA

GREENWAVES TECHNOLOGIES

Workshop

Exercise:

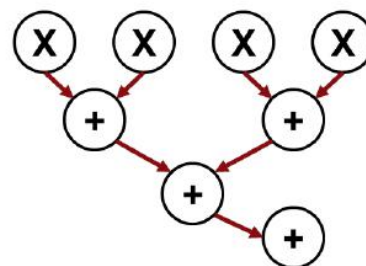
1. Create graph in nntool to compute a matrix multiplication (C = A*B) using int8 quantization and profile it in different scenarios:
   a) A (32x64) and B (64x128) variable
   b) A (32x64) variable and B (64x128) constant - is it better than a)? Why? Can you improve it more?
2. Add a resize node in front of the mobilenet v2 we deployed (DO NOT use the PATCH trick)
   a) Force the input to be stored in L3 RAM
3. Deploy a yolox model like we did for the mobilenet v2. Can all the optimizations we did be applied straight-forward? (maybe you have to change something :))

Company Proprietary

Convolution HW acceleration: NE16

NN Digital Accelerators

- Two main families:
  - Convolvers:
    - Filter specific (difficult to generalize)
    - Maximal data reusage
    - Work well in Depth-wise convolutions
    - Examples: NE16



Implement the spatial filter as an adder tree

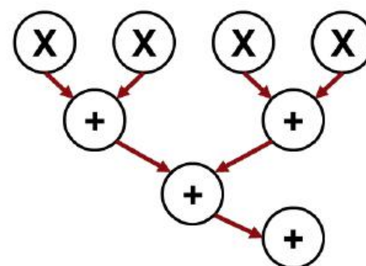
  - MatMul Accelerators:
    - Adapts well to any type of filter sizes
    - Does not work in Depth-wise
    - Requires Im2Col
    - Most commonly used approach
    - Examples: Google TPU, NVDLA

- More Options available:
  - **Combining** ideas of the 2 families
  - Exploit **serialization**: e.g. bit-serial multipliers
  - Exploit **sparsity**

Company Proprietary
GREENWAVES TECHNOLOGIES

NN Digital Accelerators

- Two main families:
  - Convolvers:
    - Filter specific (difficult to generalize)
    - Maximal data reusage
    - Work well in Depth-wise convolutions
    - Examples: NE16
  - MatMul Accelerators:
    - Adapts well to any type of filter sizes
    - Does not work in Depth-wise
    - Requires Im2Col
    - Most commonly used approach
    - Examples: Google TPU, NVDLA



Implement the spatial filter as an adder tree

- More Options available:
  - **Combining** ideas of the 2 families
  - Exploit **serialization**: e.g. bit-serial multipliers
  - Exploit **sparsity**

NE16

Company Proprietary

Why bit serialization?

0	1	0	1	1	1	0	1
1	1	0	0	1	0	0	0

✖

1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0

One-step solution**:**
- more **power efficient**
- more area (**more complex**)
- not scalable (less bit, same power)
  e.g. 4x8bits = 8x8bits

Bit-Serial solution**:**
- Each cycle compute 1bit product
- less area (**simpler**)
- less bits, less cycles (energy)
  e.g. 4x8bits ~ 1/2 time of 8x8bits

0 1 0 1 1 1 0 1

1 1 0 0 1 0 0 0

0 0 0 0 0 0 0 0

shift & add 0 1 0 1 1 1 0 1

0 1 0 1 1 1 0 1 0

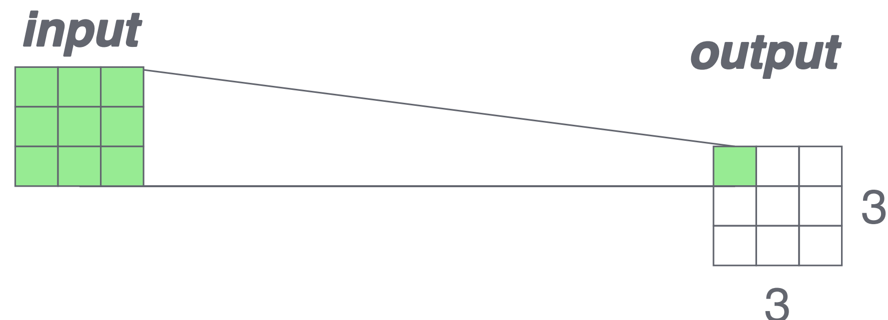
shift & add 0 0 0 0 0 0 0 0

⋮ x8

Company Proprietary

NE16 - 3x3 Convolver

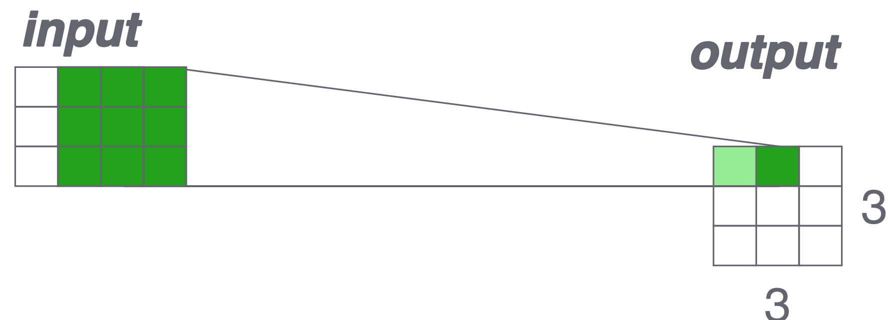
- Why 3x3? Widely adopted in many vision NN (SqueezeNet, VGG, ...)
- **Design choices:**
  - **3x3 filter** computed in parallel --> each adder tree has 9 filter elements
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels



Company Proprietary

NE16 - 3x3 Convolver

- Why 3x3? Widely adopted in many vision NN (SqueezeNet, VGG, ...)
- **Design choices:**
  - **3x3 filter** computed in parallel --> each adder tree has 9 filter elements
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels

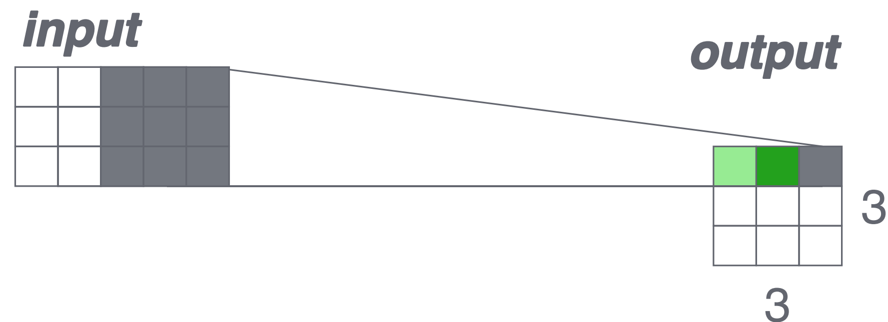


Company Proprietary

GREENWAVES TECHNOLOGIES

NE16 - 3x3 Convolver

- Why 3x3? Widely adopted in many vision NN (SqueezeNet, VGG, ...)
- **Design choices:**
  - **3x3 filter** computed in parallel --> each adder tree has 9 filter elements
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels

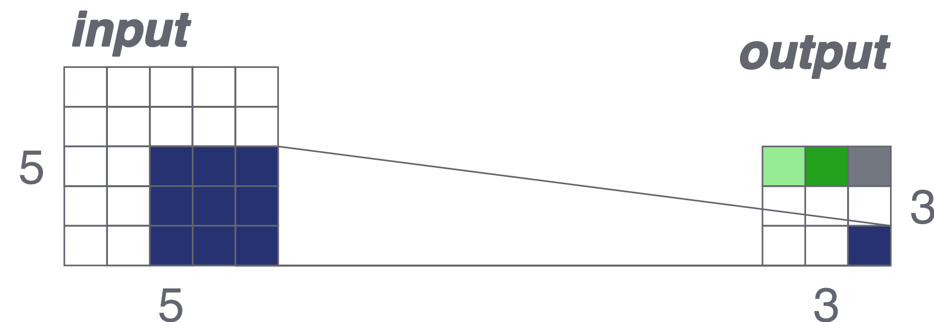


Company Proprietary

GREENWAVES TECHNOLOGIES

NE16 - 3x3 Convolver

- Why 3x3? Widely adopted in many vision NN (SqueezeNet, VGG, ...)
- **Design choices:**
  - **3x3 filter** computed in parallel --> each adder tree has 9 filter elements
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels

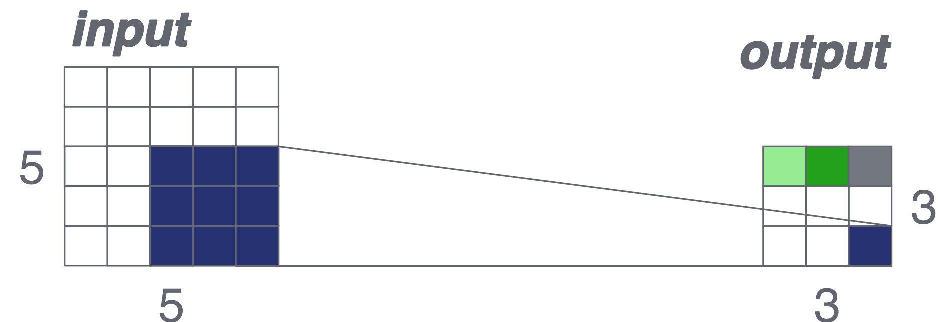


Company Proprietary

GREENWAVES TECHNOLOGIES

NE16 - 3x3 Convolver

- Why 3x3? Widely adopted in many vision NN (SqueezeNet, VGG, ...)
- **Design choices:**
  - **3x3 filter** computed in parallel --> each adder tree has 9 filter elements
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels

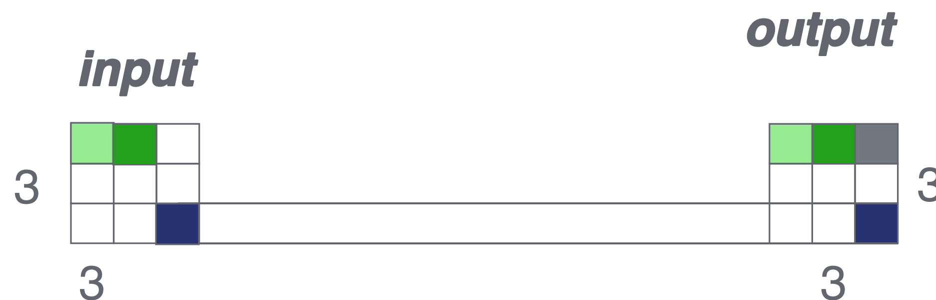


  - **5x5 input buffer**
  - **32 output channels before reiterating**

Company Proprietary

NE16 - 3x3 Convolver (ONLY?)

- How does it do 1x1 (aka MatMul)?
  - **1x1 filter** computed in parallel --> each adder tree has 1 filter elements
  - **Parallelize on the weight bits (no more bit serial here!!!)** --> 2-8 bits in parallel
  - **3x3 output pixels** computed in **parallel** --> 9 adder trees
  - **16 input channels** computed in **parallel** --> each adder tree has 16 channels



- **5x5 input buffer**
- **32 output channels before reiterating**

Company Proprietary
GREENWAVES TECHNOLOGIES

Theory is cool, but reality?

	3x3	**1x1**
Theory	162 MAC/Cyc	144 MAC/Cyc
Reality	~130 MAC/Cyc (~80%)	~60 MAC/Cyc (<50%)

Why is 1x1 so bad in reality?

- In both modes before starting the accumulation we need to load the input buffer !!!!

	Input buffer size	*Number of MACs for the input buffer*	*MAC/Load ratio*
3x3	5x5x16	32x3x3x16x3x3=41472	103.7
1x1	3x3x16	32x3x3x16=4608	32.0

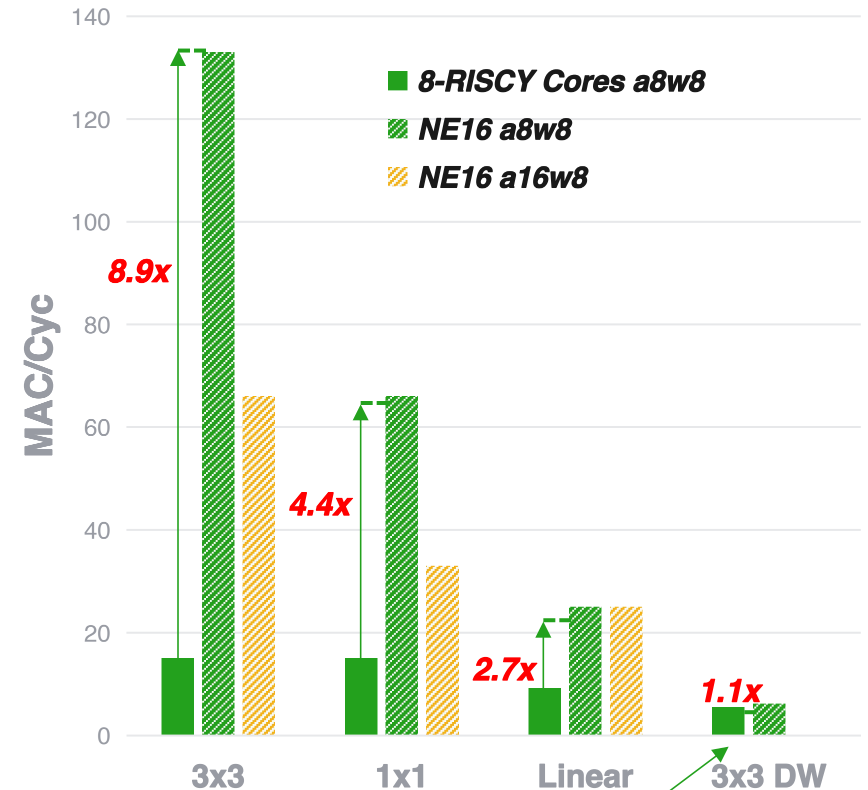
1x1 mode has lower MAC/Load ratio --> it could go faster but it is bounded by the input buffer load

Company Proprietary
GREENWAVES TECHNOLOGIES

NE16HW accelerator – heterogeneous performance

- Operating modes:
  - ***3x3*** w/ pad (w/ stride 1x1 or 2x2)
  - ***1x1*** (MatMul)
  - ***Linear*** (MatVector multiply)
  - ***DepthWise 3x3*** w/ pad (w/ stride 1x1 or 2x2)

- NE SW Library:
  - We can combine parallel SW (8 cores) and NE16 to achieve the best performance in many scenarios, e.g. Im2Col + NE16-1x1

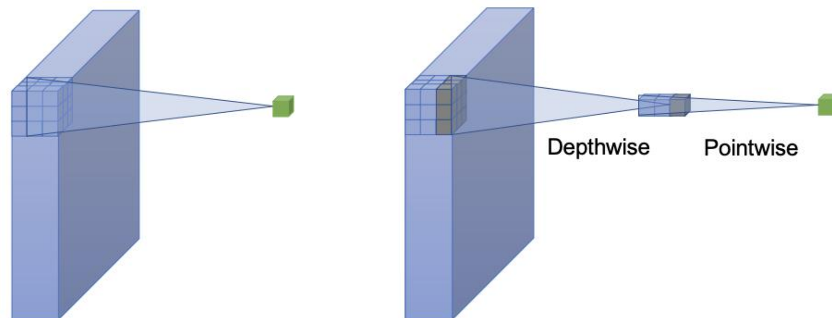


GREENWAVES TECHNOLOGIES

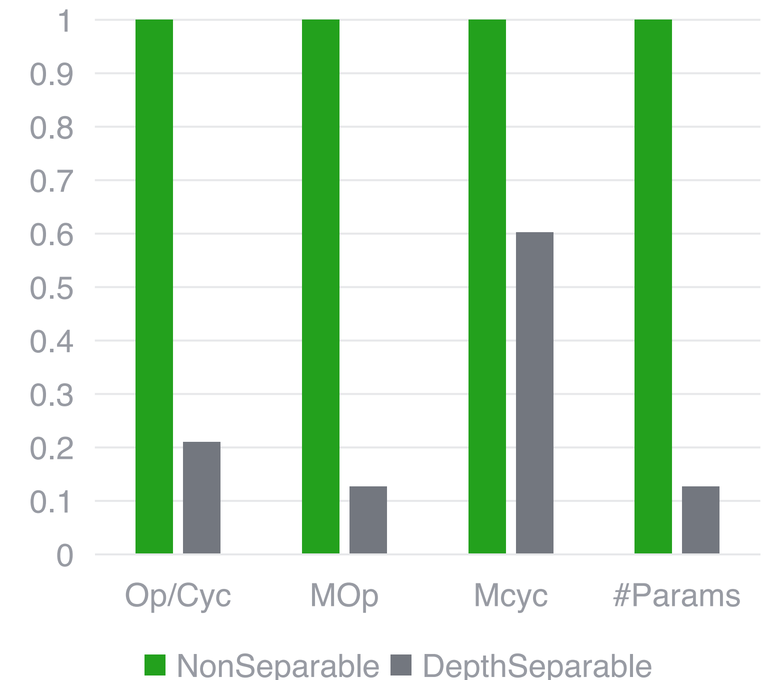
Architecture design still matters

Even with dedicated HW, NN architecture can help AI on the edge, e.g.:

- **3x3 Non-Separable** Convolution:
  - NE16-friendly --> **120MAC/Cyc**
- **3x3 DepthSeparable** Convolution: 3x3 DW + 1x1 PW
  - non NE16-friendly --> 6MAC/Cyc + 60MAC/Cyc ~ **20MAC/Cyc**



[Depthwise Convolution is All You Need for Learning Multiple Visual Domains]



GREENWAVES TECHNOLOGIES

Hands-on