tinyML® EMEA

Enabling Ultra-low Power Machine Learning at the Edge

tinyML EMEA Technical Forum 2021 Proceedings

June 7 – 10, 2021

Virtual Event

TINY ML

www.tinyML.org

AT THE VERY EDGE

Image-based Target Identification on a tiny RISC-V multi-core application processor

Enabling AI

Presented by:

Manuele Rusci

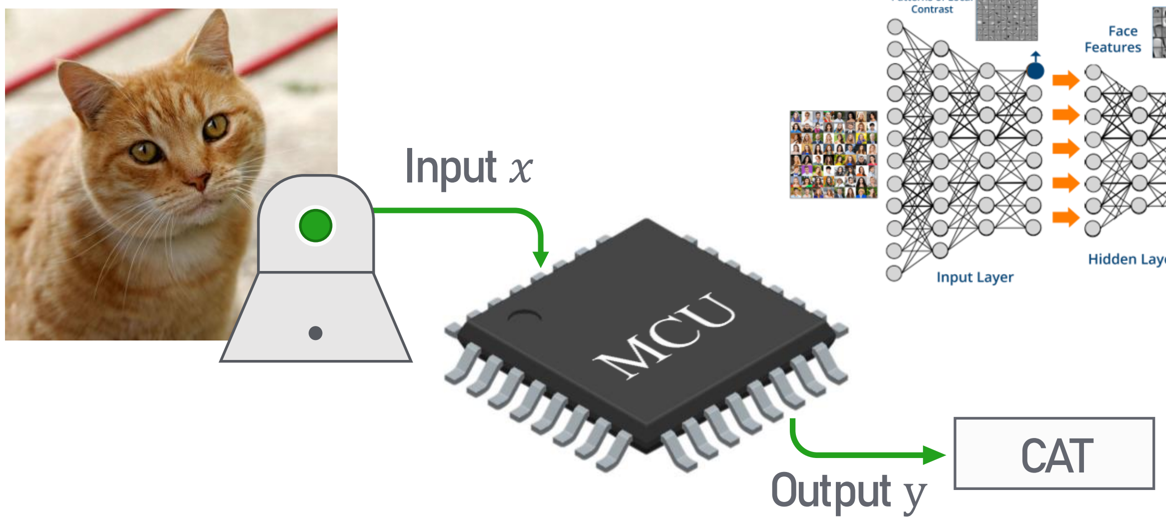
GreenWaves Technologies

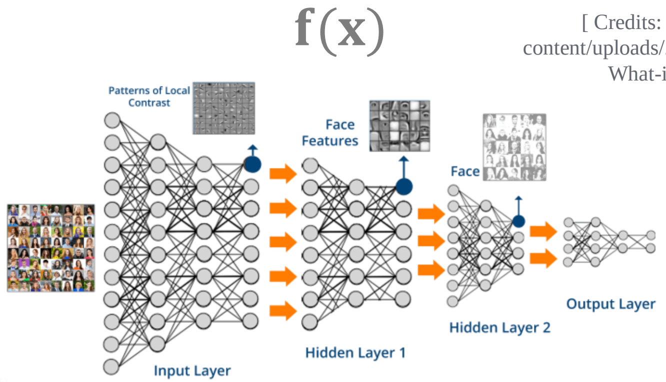
GREENWAVES TECHNOLOGIES

Design of Image-based Smart Sensors

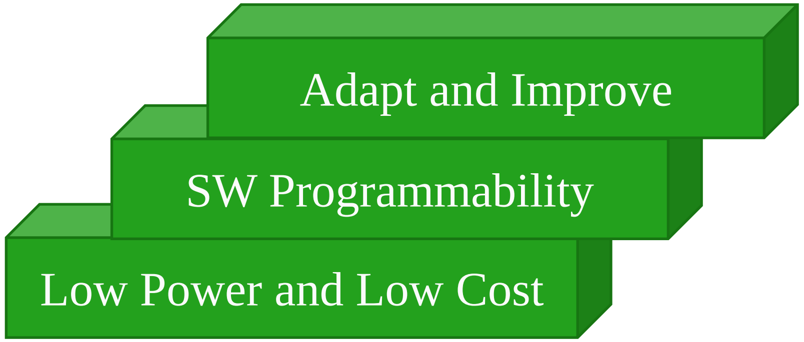
- Low Power Consumption
- Flexible
- Easy to "program"
- Efficient and Effective
  - Going beyond "TinyML" benchmarks

MCU-centric Smart Camera Systems





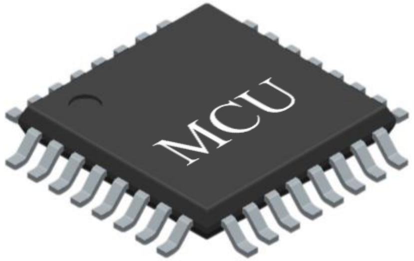
[Credits: https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png]

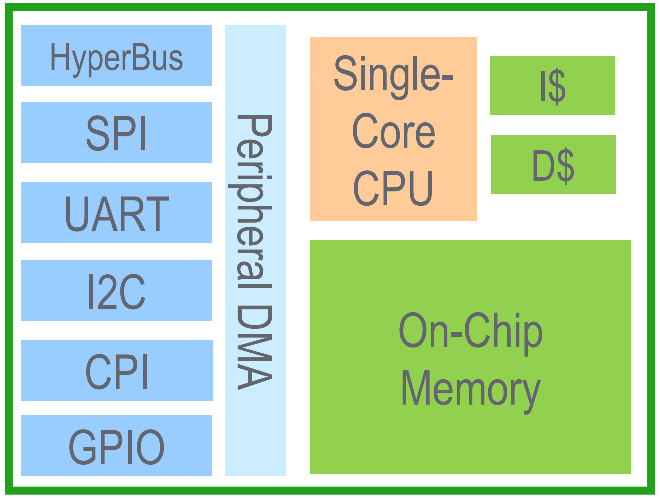


MicroControllers (MCUs) for Deep Neural Networks (DNNs) based Image Processing and Identification on battery powered devices

GREENWAVES TECHNOLOGIES

Typical Design flows for DNN Deployment



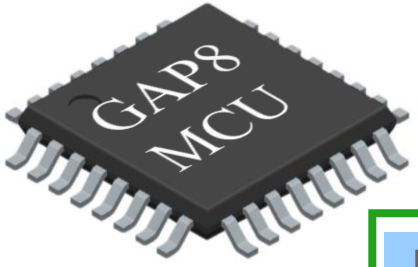



```
Inference_tak(
    In = input_data;
    Filter = coeff_0;
    Out = int_buffer_0;
    Conv_Layer(char * In, char * Filter,
    char * Out);
    In = int_buffer_0;
    Filter = coeff_1;
    Out = int_buffer_1;
    Conv_Layer( char * In, char * Filter,
    char * Out);
    In = int_buffer_1;
    Filter = coeff_2;
    Out = int_buffer_0;
    Conv_Layer(char * In, char * Filter,
    char * Out);
)
```

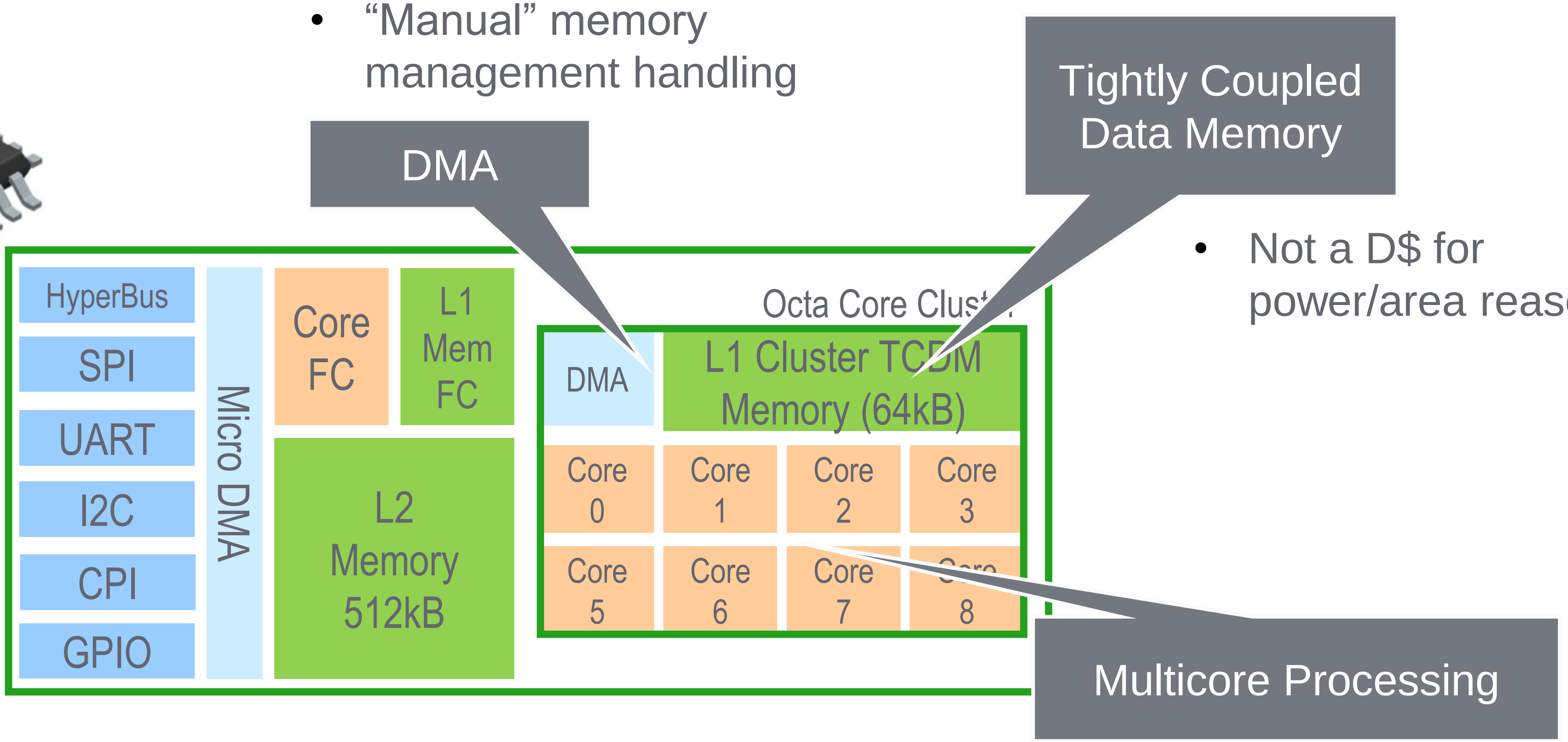
Optimized Code Generation targeting single-core and flat memory

- Bare Metal Programming (e.g. CMSIS-NN)
- Software runtime w/ optimized library (e.g. TF micro, STMCubeAI)
- Binary Code Generation (e.g. uTVM)

Our Design mantras for energy-efficient MCU design!



- “Manual” memory management handling



- Not a D$ for power/area reason

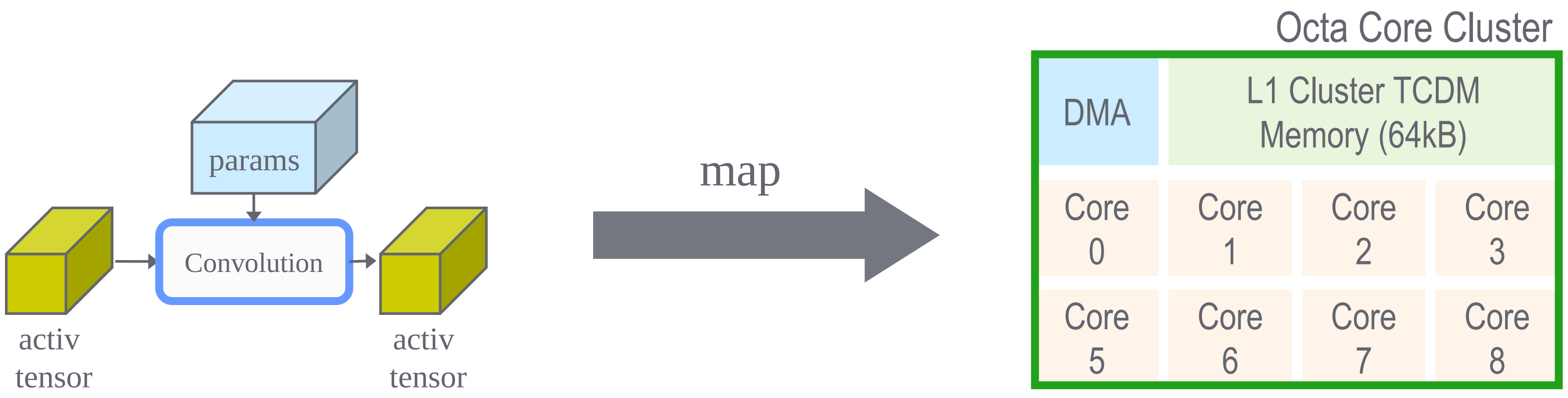
Going beyond typical MCU architectures!

- Cannot leverage on existing frameworks for efficient DNN deployment on MCU because of memory hierarchy and parallel computation.

- General purpose RISC-V CPUs but optionally CNN accelerators
- DSP-oriented ISA

GREENWAVES TECHNOLOGIES

Parallel Computation

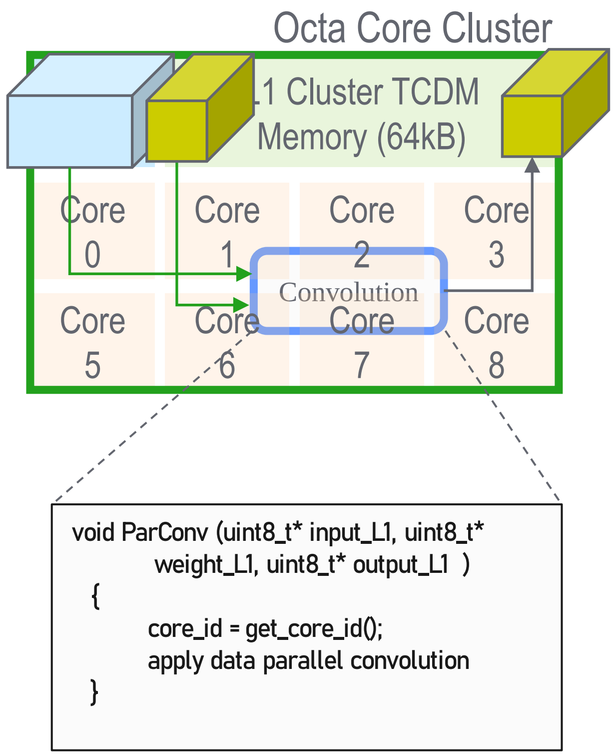


DNN Operator

Parallel Computation

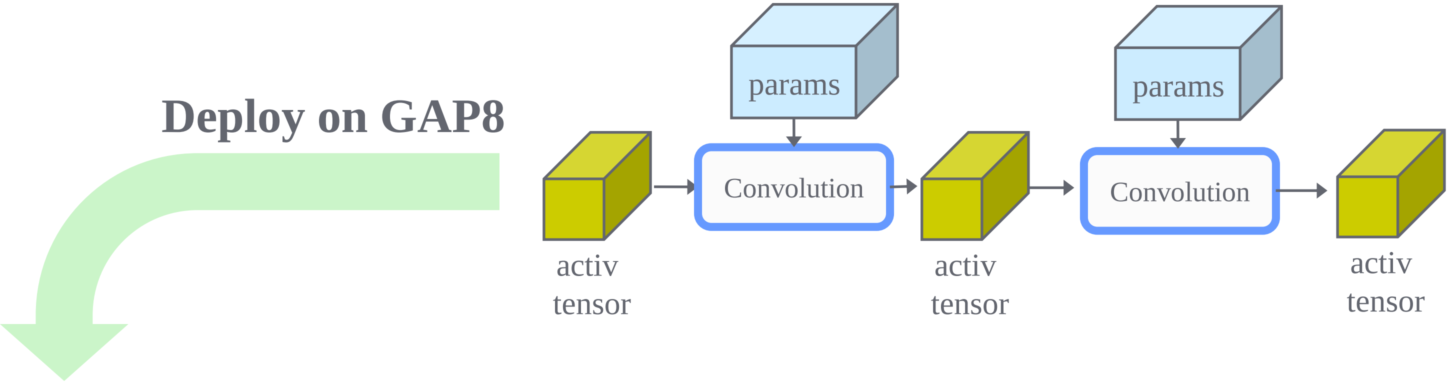
Parallel DNN Basic Kernels Library

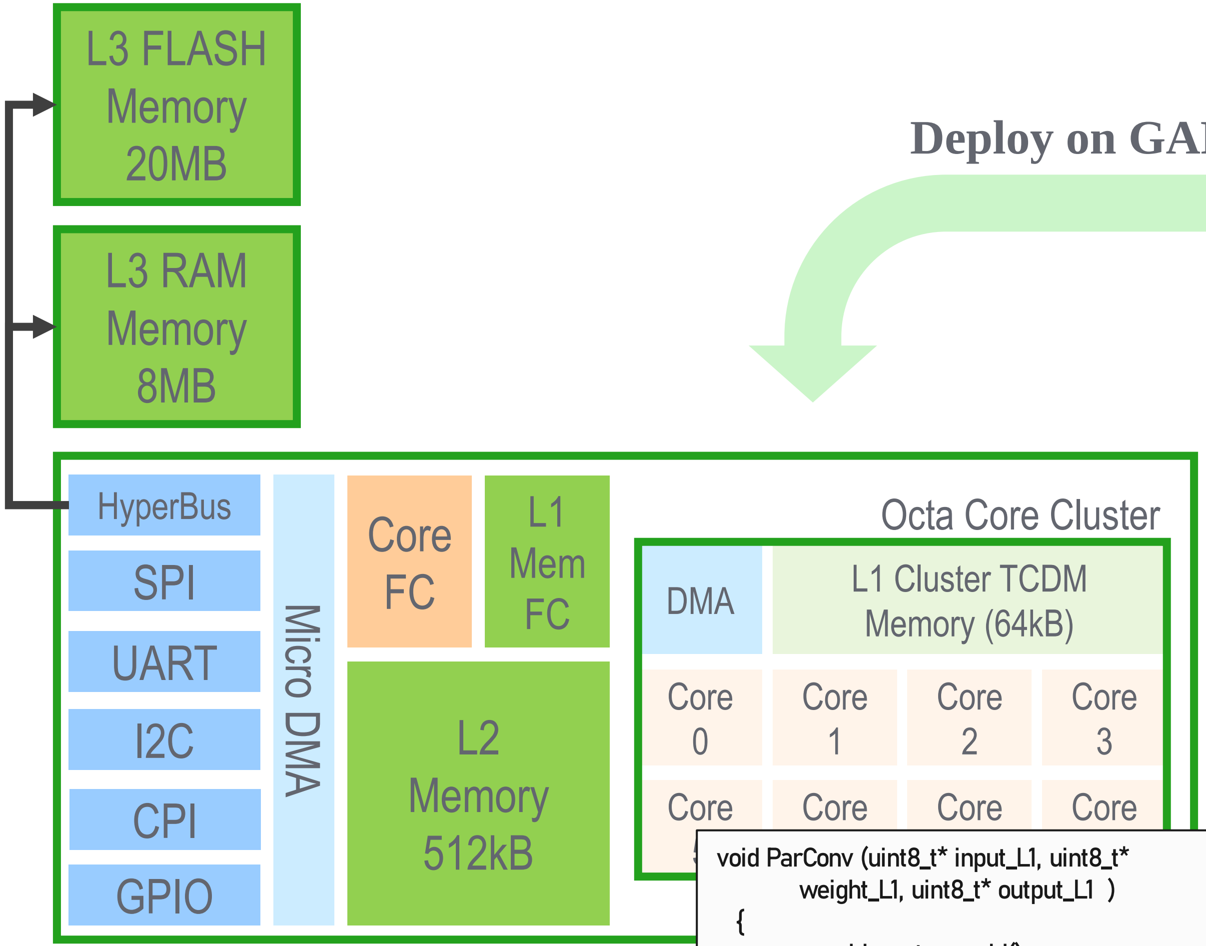
- Optimized to run efficiently on the 8-core cluster
- Leverages GAP8 ISA-extended instructions & vectorization
- Operate on Cluster L1 Data



GAP8 Optimized SW Basic Kernel

Mapping a NN Graph to the GAP8 HW/SW architecture






```
void ParConv (uint8_t* input_L1, uint8_t*
        weight_L1, uint8_t* output_L1  )
{
        core_id = get_core_id();
        apply data parallel convolution
}
```

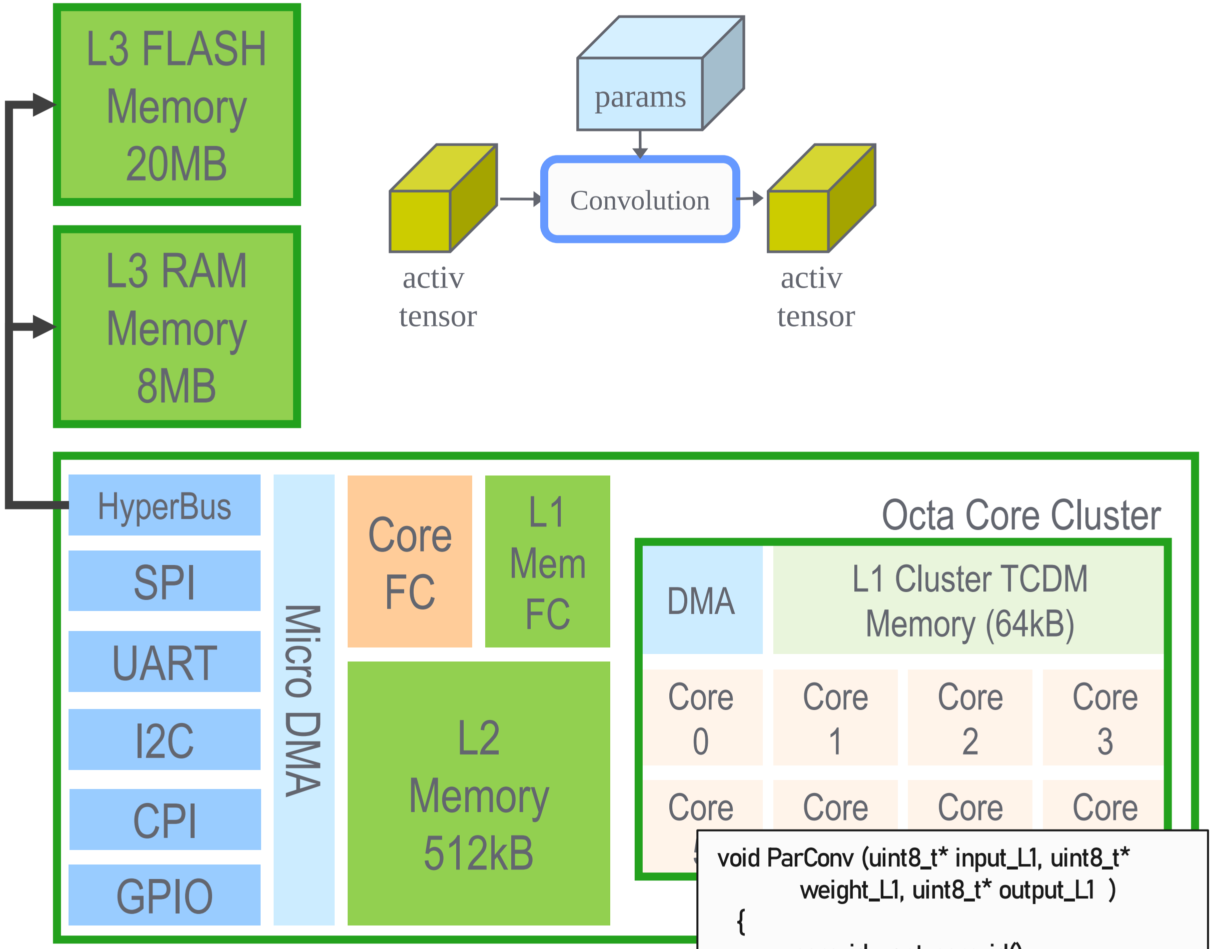
GAP8 Optimized SW Basic Kernel

Main Challenges

- DNN Graph memory requirements do not fit the L1 cluster's memory
- **Optimize data transfer** from/to the cluster parallel engine (no Dcache!)

GREENWAVES TECHNOLOGIES

Mapping a NN Graph to the GAP8 HW/SW architecture



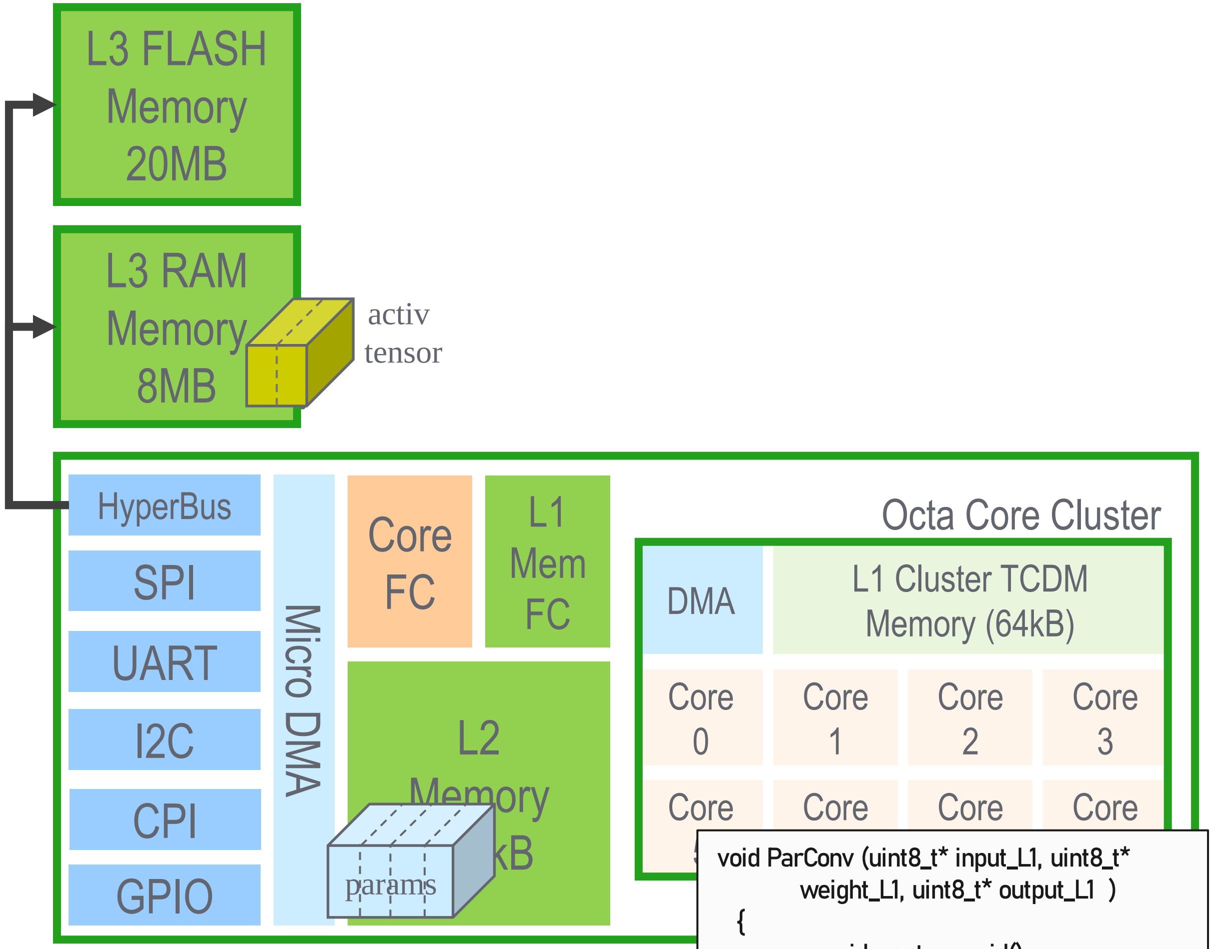

```
void ParConv (uint8_t* input_L1, uint8_t*
        weight_L1, uint8_t* output_L1  )
 {
        core_id = get_core_id();
        apply data parallel convolution
 }
```

GAP8 Optimized SW Basic Kernel

Computation dataflow

GREENWAVES TECHNOLOGIES

Mapping a NN Graph to the GAP8 HW/SW architecture




```
void ParConv (uint8_t* input_L1, uint8_t*
         weight_L1, uint8_t* output_L1  )
  {
        core_id = get_core_id();
        apply data parallel convolution
  }
```

GAP8 Optimized SW Basic Kernel

Computation dataflow

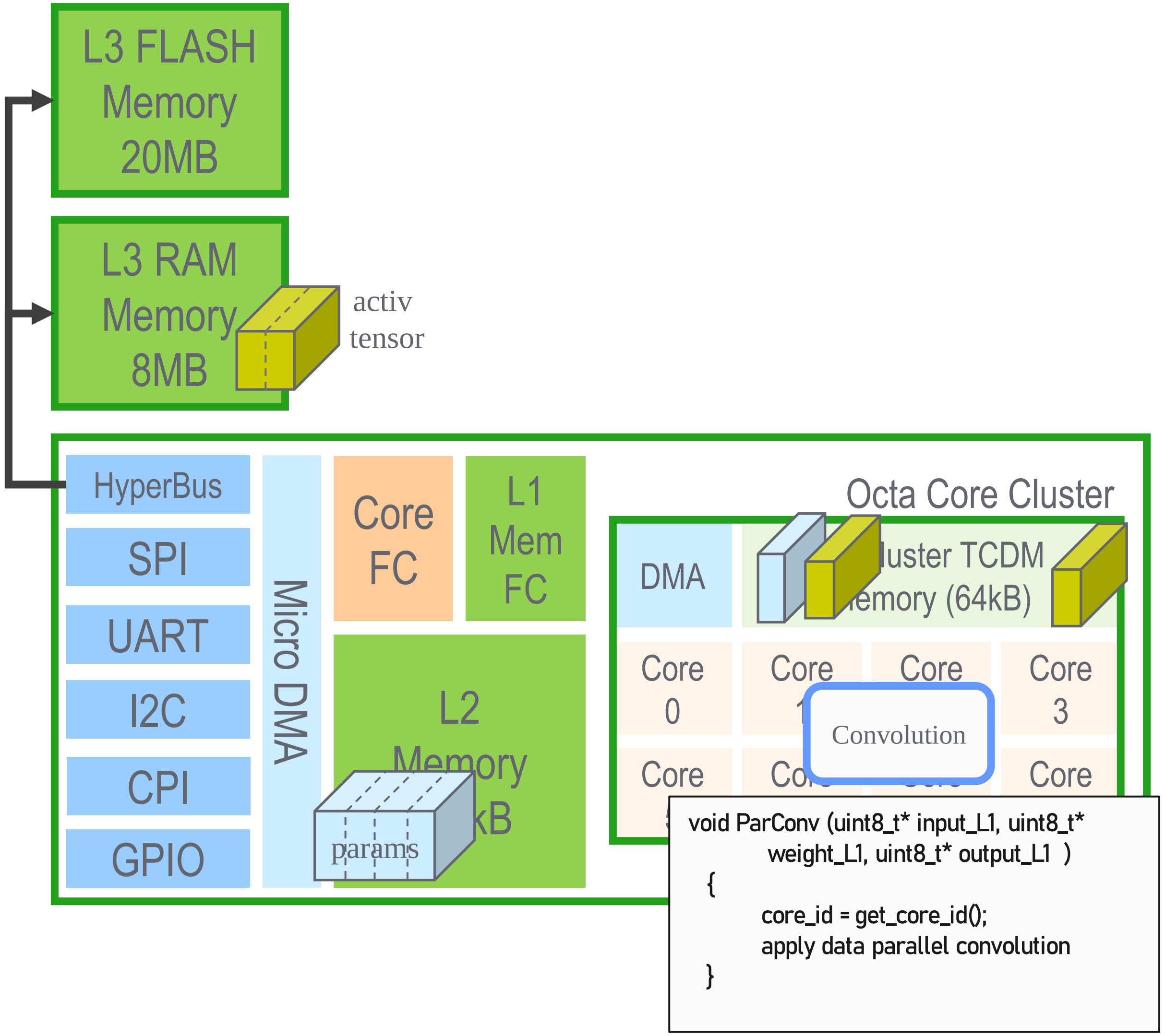
Ahead of time

- 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

Mapping a NN Graph to the GAP HW/SW architecture



GAP8 Optimized SW Basic Kernel

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

GREENWAVES TECHNOLOGIES

Challenges

Given L1, L2, L3 memory constraints

- **Where to store data (L3/L2) ?**
  - Dealing with many static (e.g. parameters) or dynamic (e.g. IOs) tensors

- **How to tile the data to transfer to L1?**
  - Optimal sizing of the tiles to reduce memory latency overhead
    - ML/Signal Processing data traffic is predictable at compile time…

- **How to produce an optimized code?**
  - Double-buffering mechanism

Our Solution: the Autotiler Tool for TinyML deployment on GAP8

The **AT Model function** calls the **AT Generators APIs** corresponding to the graph's layers

User Kernels: generated function code that *interleaves* calls to basic kernels and memory transfers

Host (x86)

AT Model

GWT Autotiler Tool

User Kernels

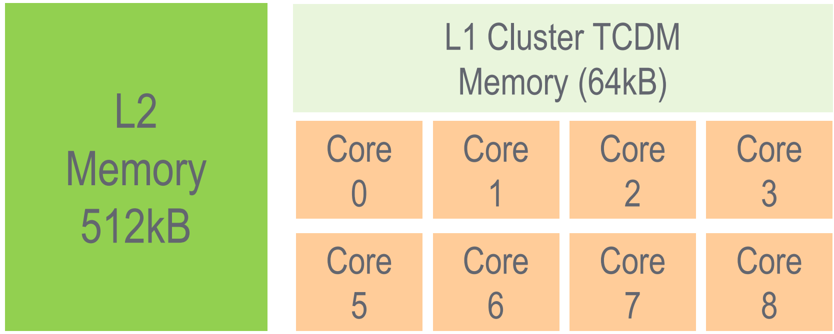
```
…
CNN_ConvolutionPoolAct_SQ8(
   "Conv_Layer0",
   4, 1, 32, 32, 112, 112,
   KOP_CONV_DW, 3, 3, 1, 1, 1, 1, 1,
   KOP_NONE, 0, 0, 0, 0, 0, 0, 0,
   KOP_RELU
);
CNN_ConvolutionPoolAct_SQ8(
   "Conv_Layer1",
   4, 1, 32, 64, 56, 56,
   KOP_CONV, 1, 1, 1, 1, 0, 0, 1,
   KOP_NONE, 0, 0, 0, 0, 0, 0, 0,
   KOP_RELU
);

…
```

- **Select** the **best** basic kernels
- **Compute** the **tile** size of any tensor
- Handle **memory allocation** (static and dynamic)

```
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
}
```

Calls to basic kernels

Autotiler Code Generation example



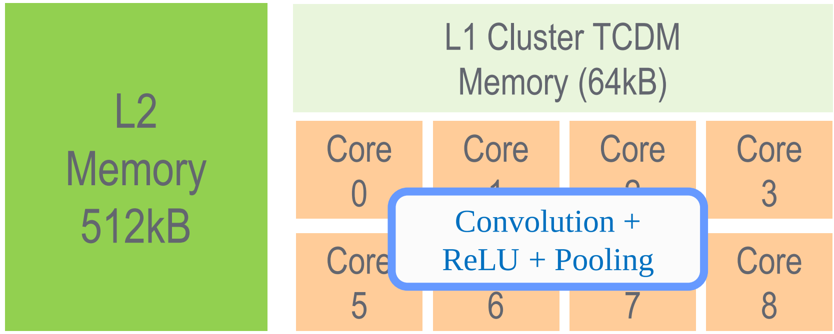
Convolution +
ReLU + Pooling

fused convolutional layer

- Operand arguments fits on-chip L2 memory (512 kB) but not the L1 memory (64kB)

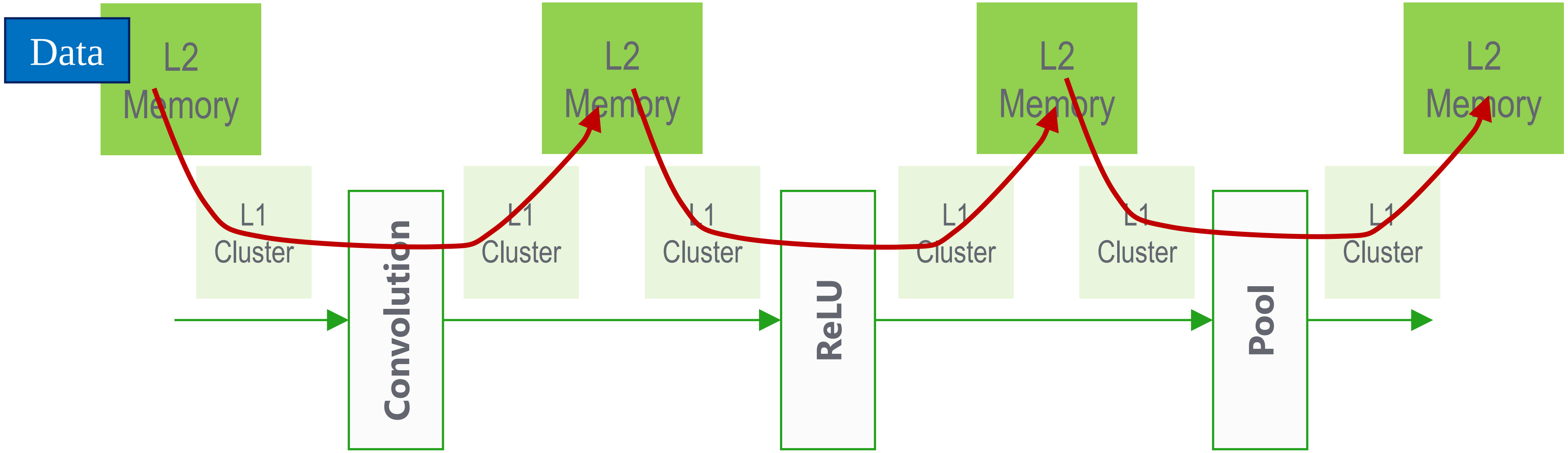
Assuming L2 as working memory

Autotiler Code Generation example



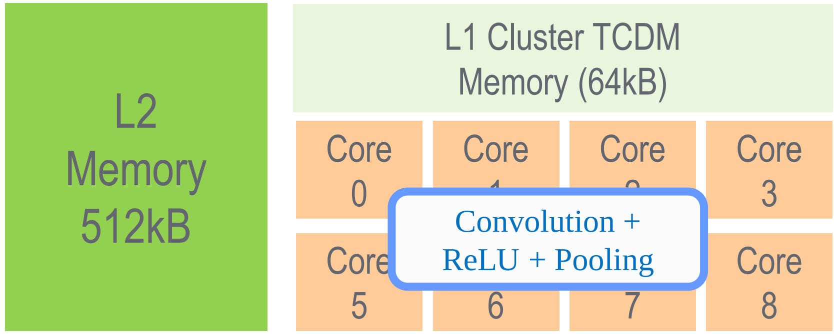
Not optimal!

- Increase L2 BW means higher energy (and latency)

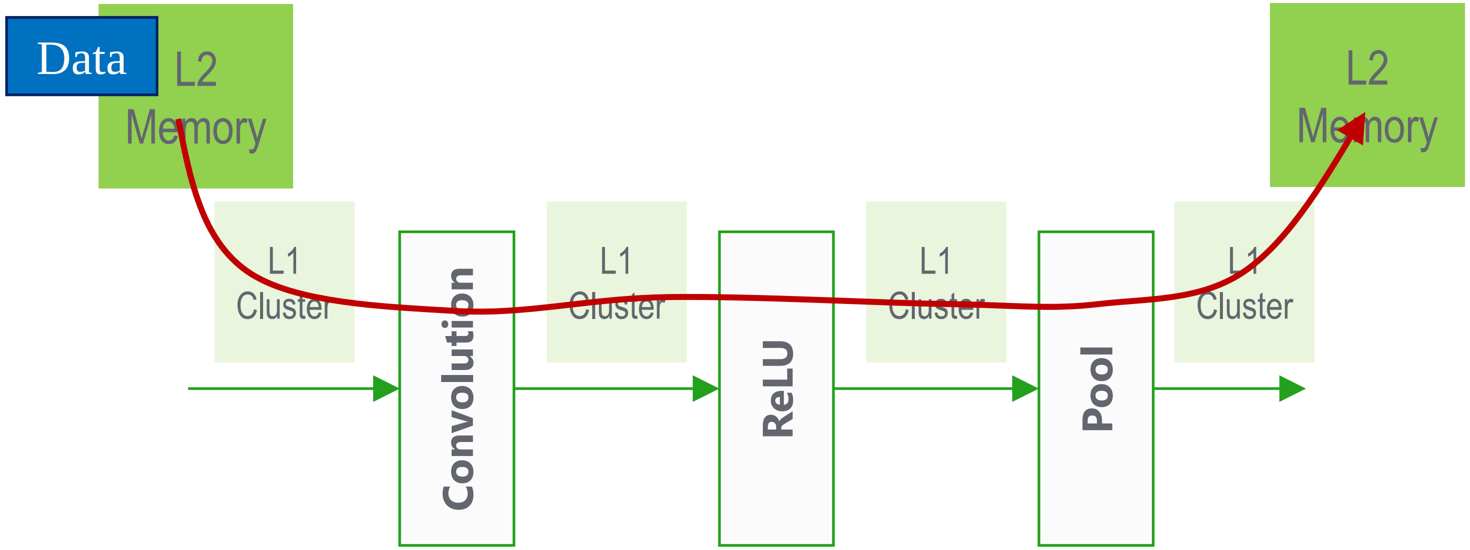


Assuming L2 as working memory

Autotiler Code Generation example



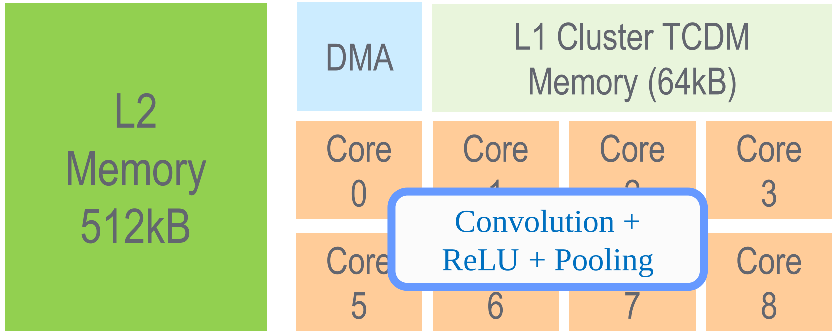
Saving Memory BW!

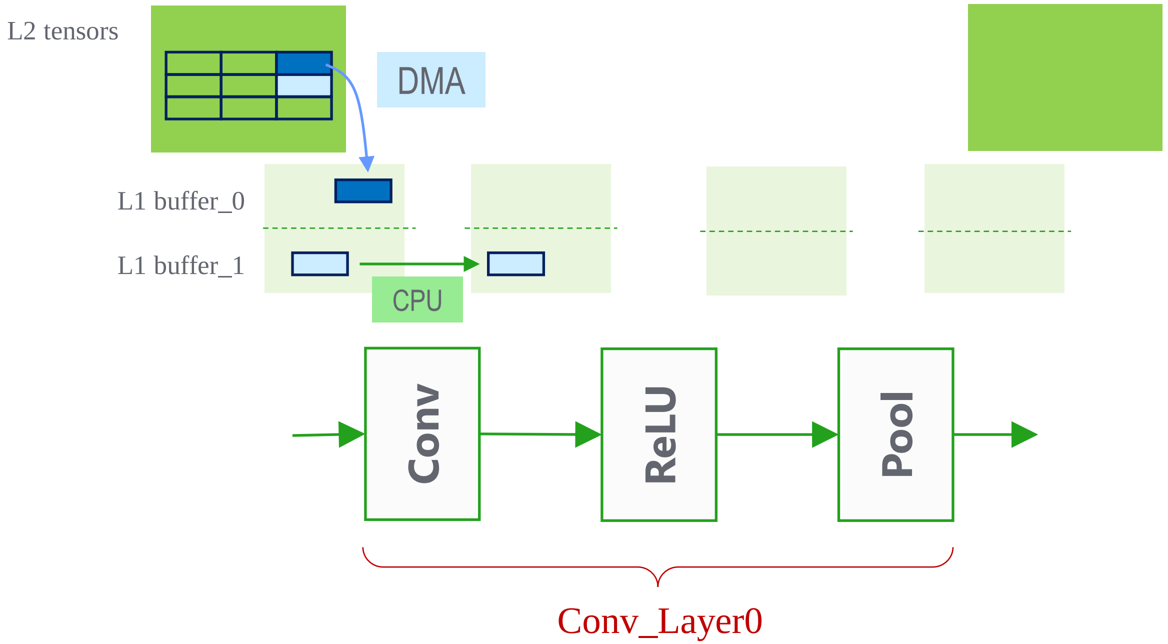


Assuming L2 as working memory

GREENWAVES TECHNOLOGIES

Autotiler Code Generation example



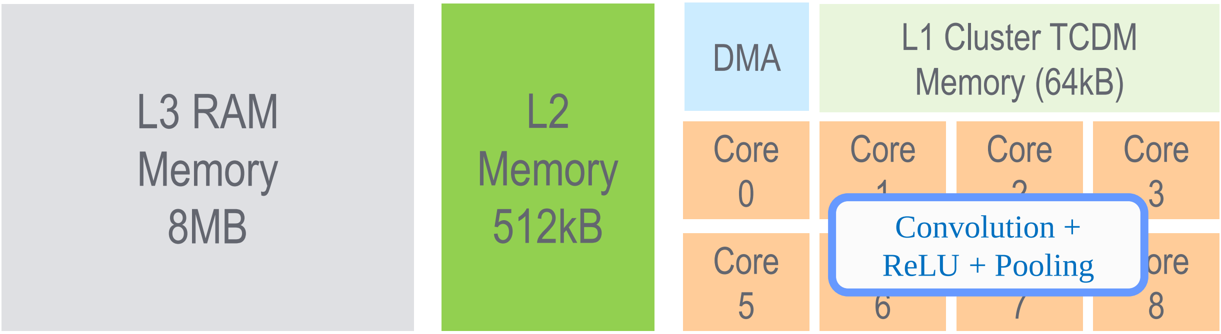


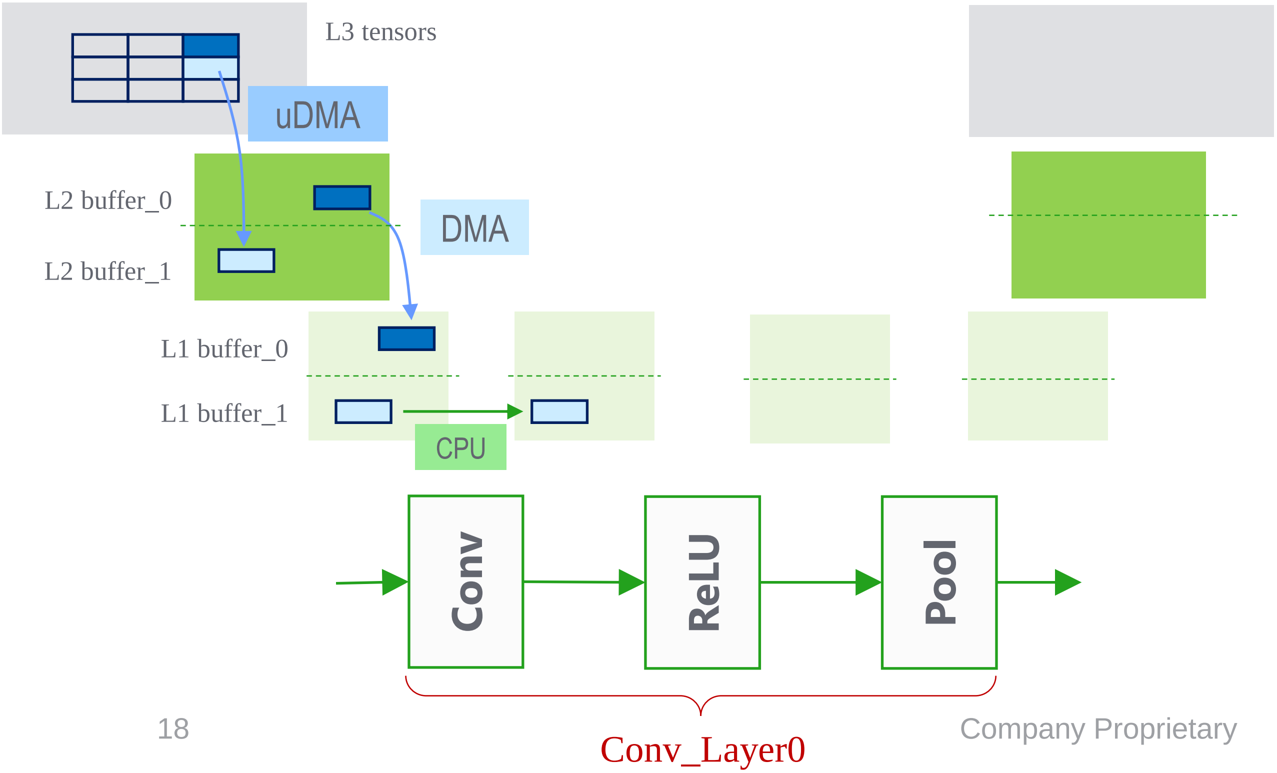
Generated User Kernels

```
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

}
```


GREENWAVES TECHNOLOGIES

Autotiler Code Generation example





Generated User Kernels

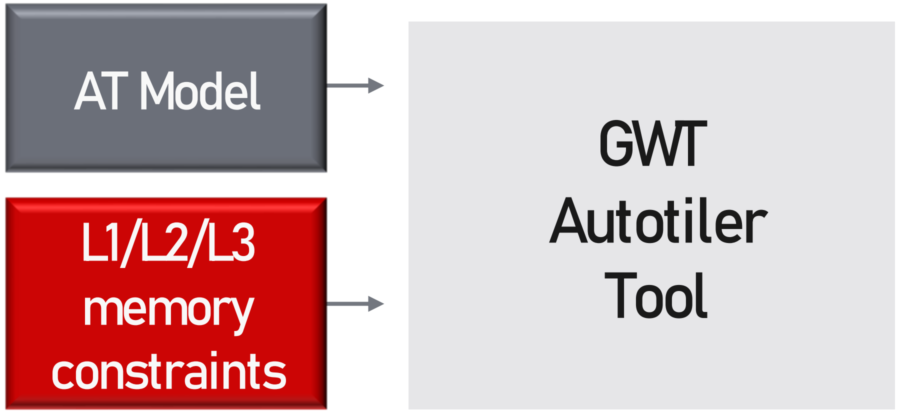
```
static void Conv_Layer0
(
    signed char * In,       // input L3 vector
    signed char * Weights,  // input L3 vector
    signed char * Bias,     // input L3 vector
    signed char * Out,      // output L3 vector
){

    //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
}
```

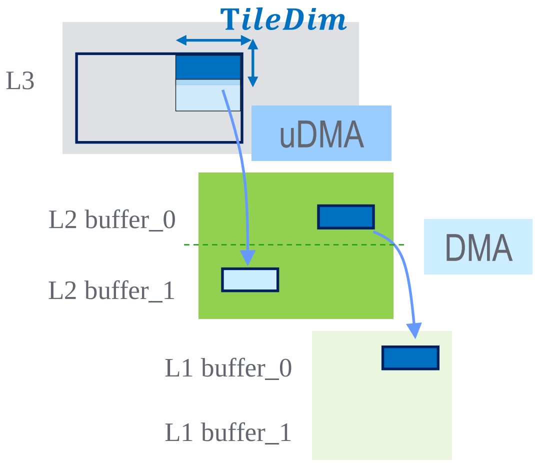
Enables larger kernels at low- overhead transfer cost

Solution of the Tiling problem

The *AT engine* computes the optimal **tiling scheme** based on:

- Computation dataflow (defined by the AT model)
- Memory Constraints (input user defined)





$$TileDim = \arg\min TilingOverhead = \frac{Transferred\ Data}{Total\ Data}$$

$$\text{s.t. } Used\ Memory < Available$$

Lower is better
Optimal = **1**

GREENWAVES TECHNOLOGIES

Experimental Setup and Results

The **GWT Autotiler** is part of the **GAP*flow*** toolset, which is included in the GAP SDK (https://github.com/GreenWaves-Technologies/gap_sdk)

- *NNtool* front-end to produce the AT model from TFLITE or ONNX
- Autotiler generate source code, including graph glue code
- Automatic allocation of dynamic and static graph's tensors

The **GWT Deployment framework** including the GWT Autotiler is tested over several Image-based benchmarks that runs on *GAP8*

- Imagenet Classification (Mobilenets)
- Person Detections
- Object Classification (License Plate)

Deep Learning based Image Processing on GAP8

Credits: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

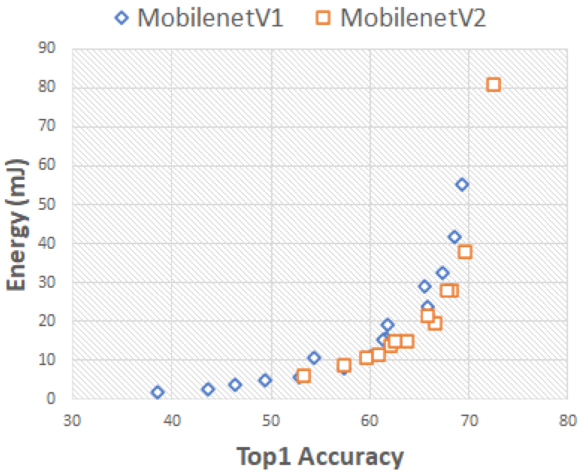
GreenWaves-Technologies / **image_classification_networks**

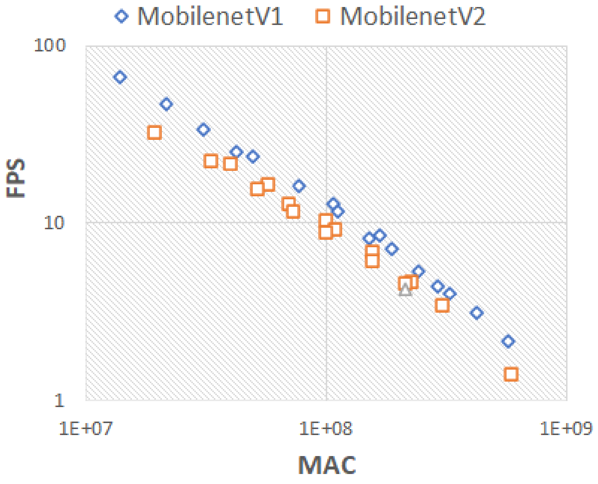
```
$ git clone git@github.com:GreenWaves-Technologies/image_classification_networks.git
$ make clean all run platform=gvsoc
```

Supported Models

Below the models tested with the GAP *flow*. More will come shortly...

MODEL ID	Quantized TFLite Graph	MACs (M)	Parameters (M)	Top 1 Accuracy	FPS	Energy (mJ)
0	MobileNet V1 224 1.0	569	4.24	70.1	2.1	49.7
1	MobileNet V1 192 1.0	418	4.24	69.2	3.1	35.2
2	MobileNet V1 160 1.0	291	4.24	67.2	4.4	25.1
3	MobileNet V1 128 1.0	186	4.24	63.4	7.2	15.2
4	MobileNet V1 224 0.75	317	2.59	66.8	4.0	27.6
5	MobileNet V1 192 0.75	233	2.59	66.1	5.3	20.9
6	MobileNet V1 160 0.75	162	2.59	62.3	8.5	13.3
7	MobileNet V1 128 0.75	104	2.59	55.8	12.9	8.5
8	MobileNet V1 224 0.5	150	1.34	60.7	8.2	13.6





GAP8 1.2V@175MHz

- **GAP8** @ 1.2V, 175MHz Cluster, 250MHz FC, **up to 110mW**
- from **1.5mJ @66fps** to **55mJ@2fps** per inference (incl. ext memories)

Person Detection

People Spotting a.k.a. Visual Wake Words

(a) 'Person' (b) 'Not-person'

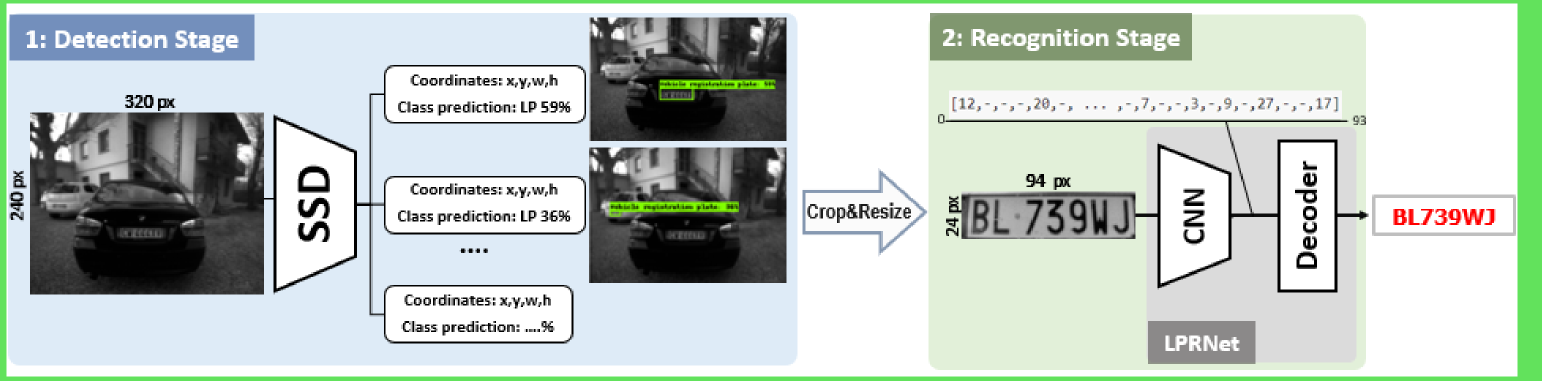
[Credits: Chowdhery, Aakanksha, et al. "Visual wake words dataset." *arXiv preprint arXiv:1906.05721* (2019)]

Model	System	Acc.	MMAC	Params	FPS	Energy (mJ)
ProxylessNAS	GAP8+ GAPflow	94.6	48.15	199k	7.55	7.75
ProxylessNAS [1]	TFMicro + STM32F7	94.6	48.15	199k	0.13	3284*
MobilnetV2 [2]	STCubeAI + STM32H7	92	20.8	391k	6.8	63.12*

*estimate

[1] Banbury, Colby, et al. "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers." Proceedings of Machine Learning and Systems 3 (2021).
[2] Table 14. STMicroelectronics "UM2611: Artificial Intelligence (AI) and computer vision function pack for STM32H7 microcontrollers"

Automatic License Plate Recognition



GreenWaves-Technologies / **licence_plate_recognition**

- ❑ **1.1 FPS** inference @ 175MHz, performing **687M MAC**.
- ❑ **4.1 MB** memory footprint (after 8-bit quantization).
- ❑ **Accuracy:** 39% mAP for LP det. & > 99.13% for LP rec.
- ❑ **Max recognition distance:** 4m for detection and 2m for recognition
- ❑ **117mW** power envelope**, 108 mJ** per inference.
- ❑ **SoA: 73x** less energy w.r.t. previous ALPR system.

Conclusion

We presented our framework to deploy DNN-based Image Target Identification on the GAP8 processor

- Optimized Parallel Kernels
- Automated Memory Management Scheme
- Code Generation

- Enable NN computation on MCU beyond tinyML benchmarks
- Our deployment framework can adapt to heterogeneous multi-core platform (e.g. featuring convolutional accelerators)

Thank you!

Manuele Rusci

manuele.rusci@greenwaves-technologies.com

https://greenwaves-technologies.com/

GREENWAVES TECHNOLOGIES

TINY ML EMEA
Virtual
JUNE 7-10, 2021

Premier Sponsor

Intelligent Agent
Neuton

Automated TinyML

Zero-code SaaS solution

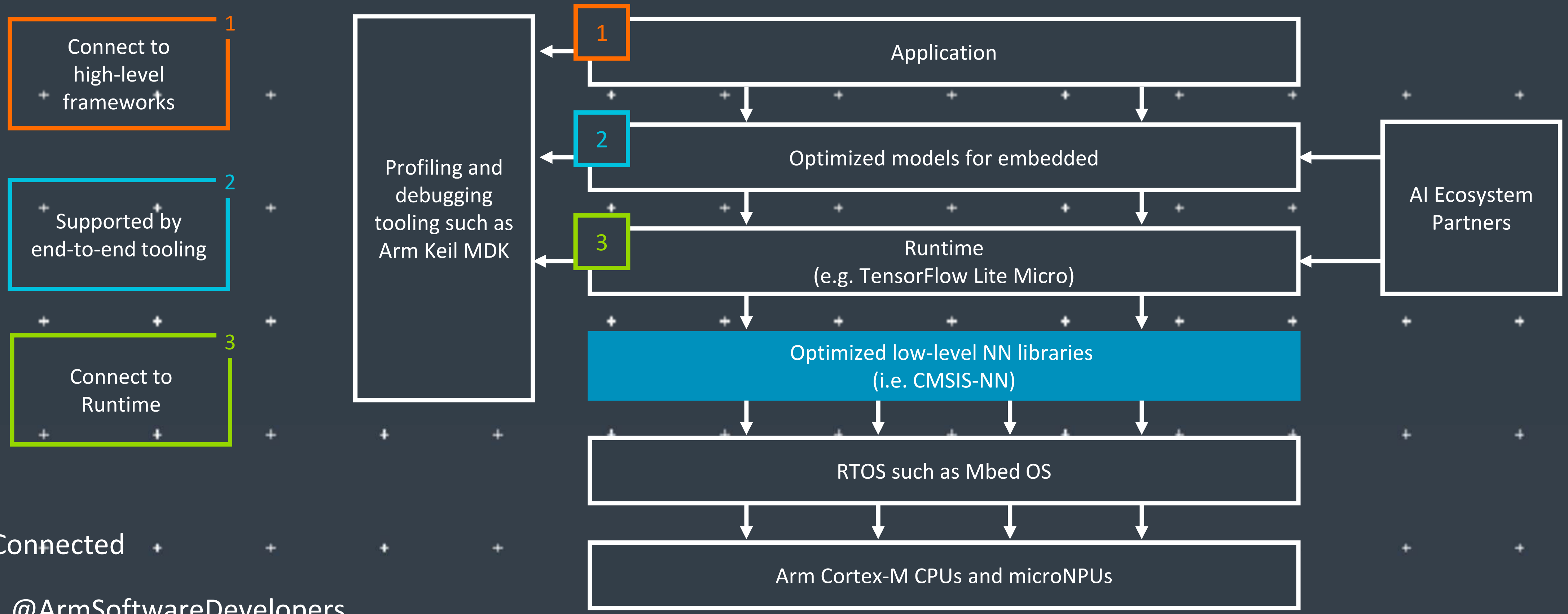
Create tiny models, ready for embedding,
in just a few clicks!

Compare the benchmarks of our compact models to those of TensorFlow and other leading neural network frameworks.

Build Fast. Build Once. Never Compromise.

Executive Sponsors

Arm: The Software and Hardware Foundation for tinyML



Stay Connected

@ArmSoftwareDevelopers

@ArmSoftwareDev

Resources: developer.arm.com/solutions/machine-learning-on-arm

TinyML for all developers

C++ library

Arduino library

WebAssembly

Dataset

Acquire valuable training data securely

Enrich data and train ML algorithms

Edge Device

Real sensors in real time

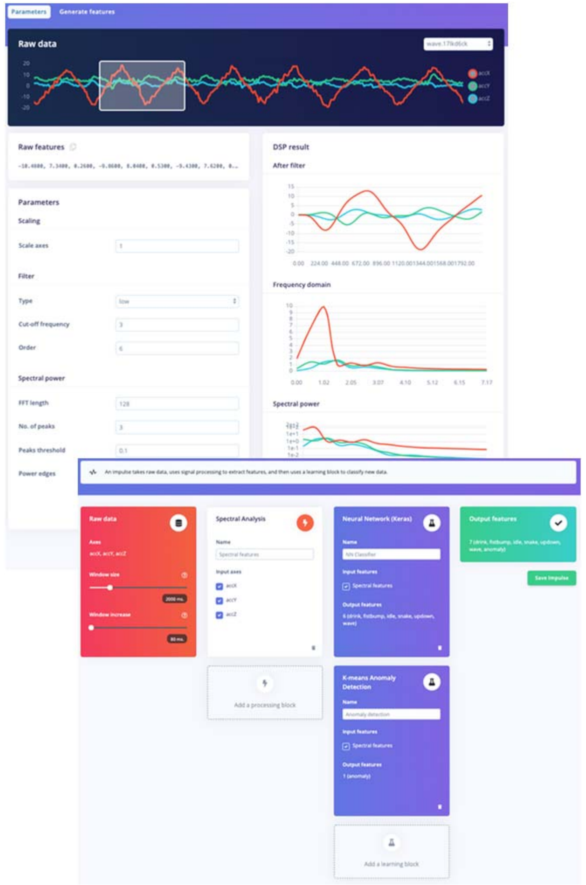
Open source SDK

Impulse

Embedded and edge compute deployment options

Test impulse with real-time device data flows

Test



www.edgeimpulse.com

Qualcomm AI research

Advancing AI research to make efficient AI ubiquitous

Power efficiency

Model design, compression, quantization, algorithms, efficient hardware, software tool

Personalization

Continuous learning, contextual, always-on, privacy-preserved, distributed learning

Efficient learning

Robust learning through minimal data, unsupervised learning, on-device learning

A platform to scale AI across the industry

Perception

Object detection, speech recognition, contextual fusion

Reasoning

Scene understanding, language understanding, behavior prediction

Action

Reinforcement learning for decision making

Edge cloud

Cloud

IoT/IIoT

Automotive

Mobile

SYNTIANT

Syntiant Corp. is moving artificial intelligence and machine learning from the cloud to edge devices. Syntiant's chip solutions merge deep learning with semiconductor design to produce ultra-low-power, high performance, deep neural network processors. These network processors enable always-on applications in battery-powered devices, such as smartphones, smart speakers, earbuds, hearing aids, and laptops. Syntiant's Neural Decision Processors™ offer wake word, command word, and event detection in a chip for always-on voice and sensor applications.

Founded in 2017 and headquartered in Irvine, California, the company is backed by Amazon, Applied Materials, Atlantic Bridge Capital, Bosch, Intel Capital, Microsoft, Motorola, and others. Syntiant was recently named a CES® 2021 Best of Innovation Awards Honoree, shipped over 10M units worldwide, and unveiled the NDP120 part of the NDP10x family of inference engines for low-power applications.

www.syntiant.com

@Syntiantcorp

TINY ML EMEA
Virtual
JUNE 7-10, 2021

Platinum Sponsors

Part of your life. Part of tomorrow.

www.infineon.com

Reality AI®

Add Advanced Sensing to your Product with Edge AI / TinyML

https://reality.ai info@reality.ai @SensorAI Reality AI

Pre-built Edge AI sensing modules, plus tools to build your own

Reality AI solutions

- Prebuilt sound recognition models for indoor and outdoor use cases
- Solution for industrial anomaly detection
- Pre-built automotive solution that lets cars "see with sound"

Reality AI Tools® software

- Build prototypes, then turn them into real products
- Explain ML models and relate the function to the physics
- Optimize the hardware, including sensor selection and placement

Gold Sponsors

LatentAI

Adaptive AI for the Intelligent Edge

Latentai.com

SensiML™

Build Smart IoT Sensor Devices From Data

SensiML pioneered TinyML software tools that auto generate AI code for the intelligent edge.

- End-to-end AI workflow
- Multi-user auto-labeling of time-series data
- Code transparency and customization at each step in the pipeline

We enable the creation of production-grade smart sensor devices.

sensiml.com

TINY ML EMEA
Virtual
JUNE 7-10,2021

Silver Sponsors

emza
visual sense

GREENWAVES
TECHNOLOGIES

HOTG

imagimob

Qeexo

seeed studio
The IoT Hardware Enabler

ST
life.augmented

Copyright Notice

www.tinyML.org