KALRAY

Supporting standard CNN inference on manycore processors

Benoît Dupont de Dinechin, CTO

MPSoC 2017

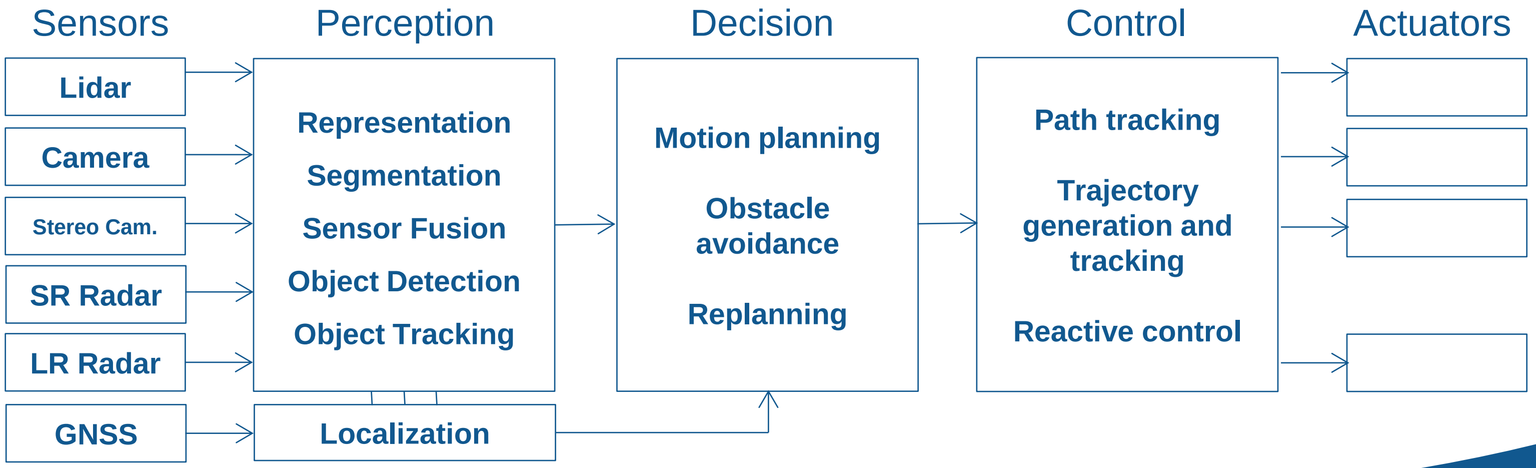
www.kalrayinc.com

PROCESSING IN AUTOMATED VEHICLES

1 **INPUT**

2 **ANALYSIS**

3 **OUTPUT**



Sensors

Perception

Decision

Control

Actuators

Lidar

Camera

Stereo Cam.

SR Radar

LR Radar

GNSS

Representation

Segmentation

Sensor Fusion

Object Detection

Object Tracking

Localization

Motion planning

Obstacle avoidance

Replanning

Path tracking

Trajectory generation and tracking

Reactive control

SENSING AND PERCEPTION REQUIREMENTS

1

INPUT (SENSING/ PERCEPTION)

Demands ***high-performance*** functions

2

ANALYSIS (DECISION)

3

OUTPUT (CONTROL)

Demand ***high-integrity*** functions

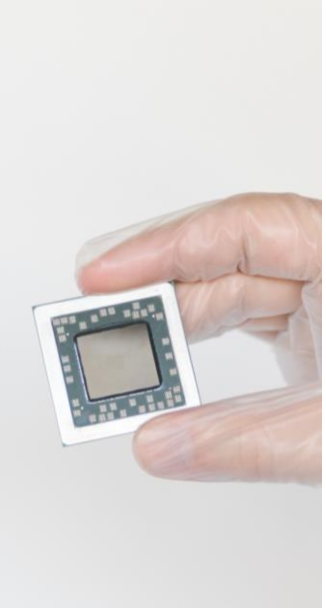
Functions such as segmentation of point cloud lidar data, image processing with deep learning and sensor fusion require increased performance

Micro-controllers for device drivers
CPUs for high-level processing
GPUs for machine learning
DSPs for mid-level processing
FPGAs for pre-processing

IDEAL PLATFORM FOR PERCEPTION

According to the Autoware[1] project leader (Cool Chips 2015, Japan), an ideal platform for perception would be a processor with multiple CPU-type cores, where one single core could be dedicated to each simple task, while compute-intensive tasks would be executed in parallel over several cores.

This platform must be complemented by reconfigurable or dedicated hardware for low-level sensor data processing



[1] Open-source software for urban autonomous driving, https://github.com/CPFL/Autoware

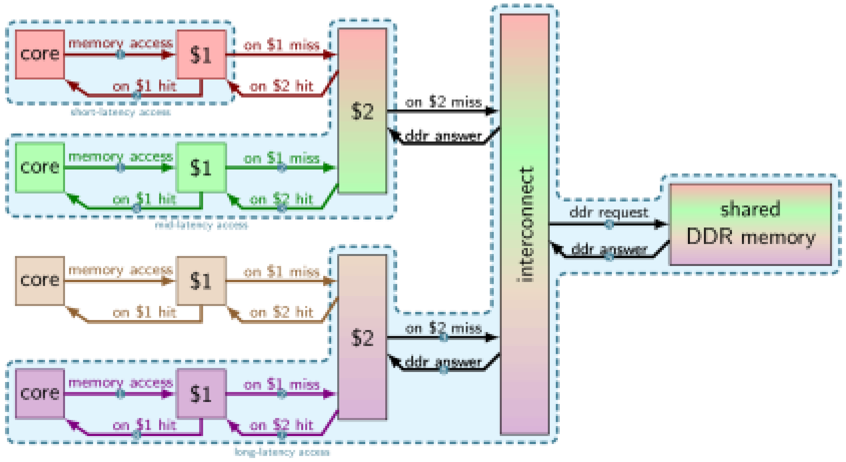
MULTICORE MEMORY HIERARCHY

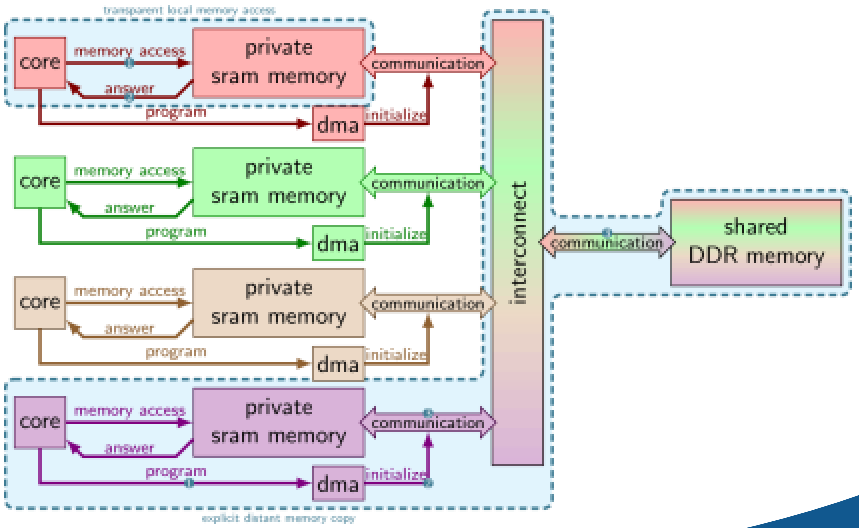
Classic Multicore Challenge

- Managing interference between cores

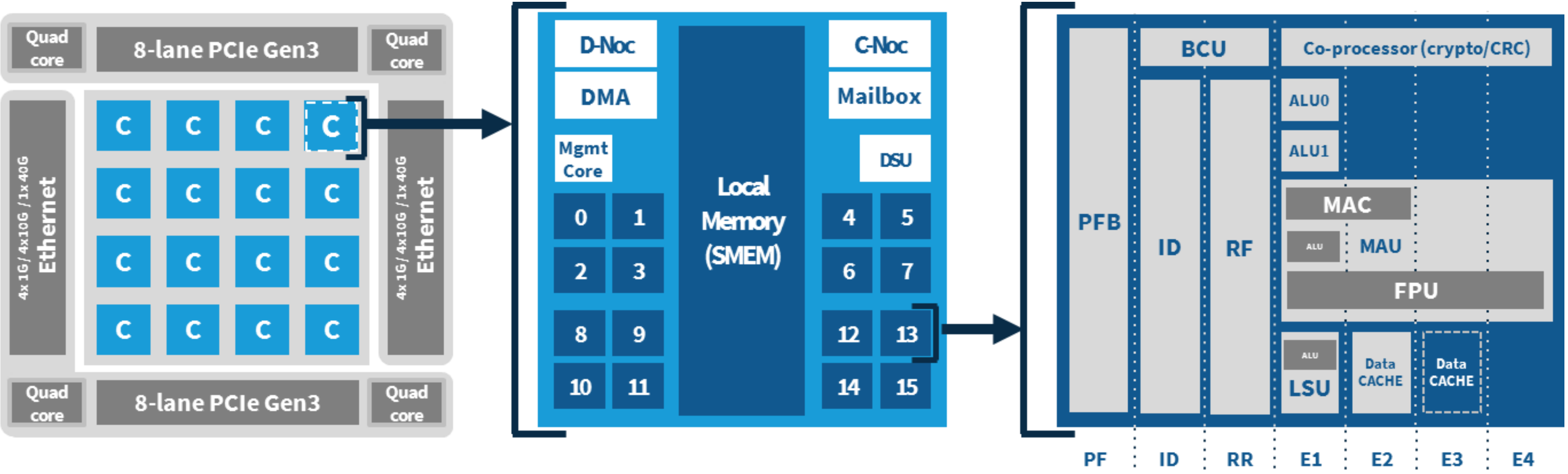
Embedded Multicore Challenge

- Programmability of DMA and private memories





2nd GENERATION MPPA® MANYCORE ARCHITECTURE (2016)



BOSTAN PROCESSOR

256+32 cores at 600 MHz, 28nm

COMPUTE CLUSTER

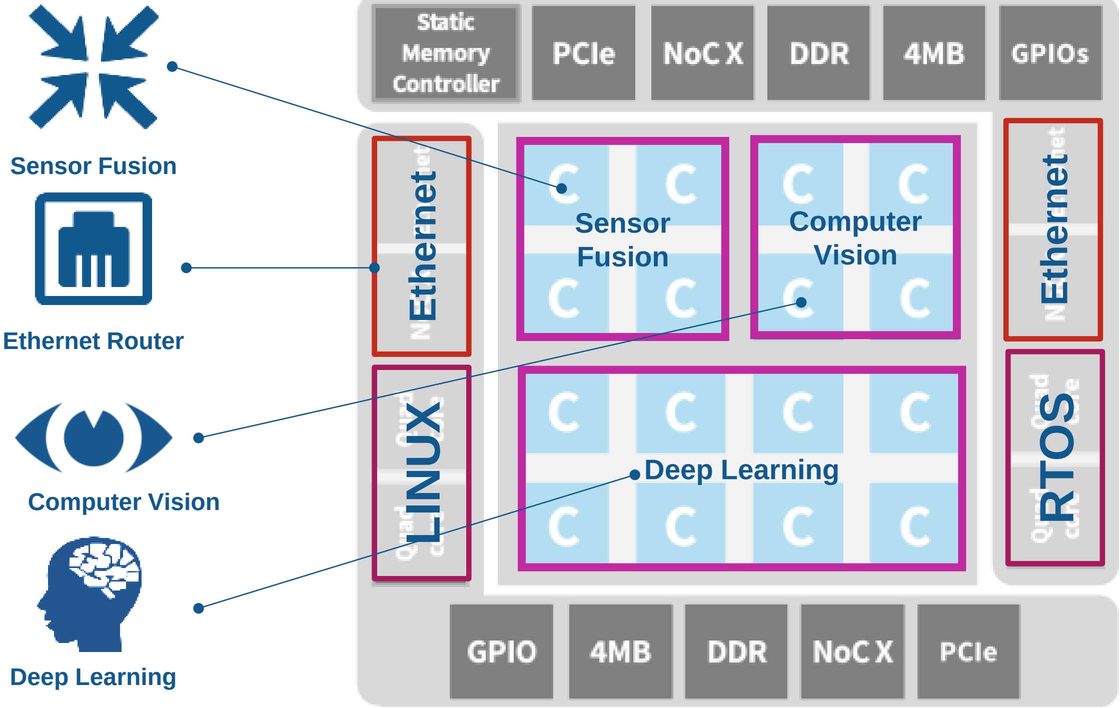
16+1 cores, 2 MB local memory

5-ISSUE VLIW CORE

64x 32-bit register file

KALRAY

MPPA® BOSTAN PROCESSOR SPATIAL PARTITIONING



BENEFITS

- Simpler execution platform with one ISA and compatible programming environments

- Energy-efficient and time-predictable

- Compute-intensive functions run in true concurrency

- Freedom from interference between functions

DEEP LEARNING INFERENCE IN PERCEPTION

Deep Neural Networks (DNN)

- Input data are processed through successive layers, with each layer applying multiple linear and non-linear operations

DNN training versus inference

- Representative data sets are used to train the network based on large number of samples (off-line)
- Trained system are deployed in embedded systems and process input under real-time constraints

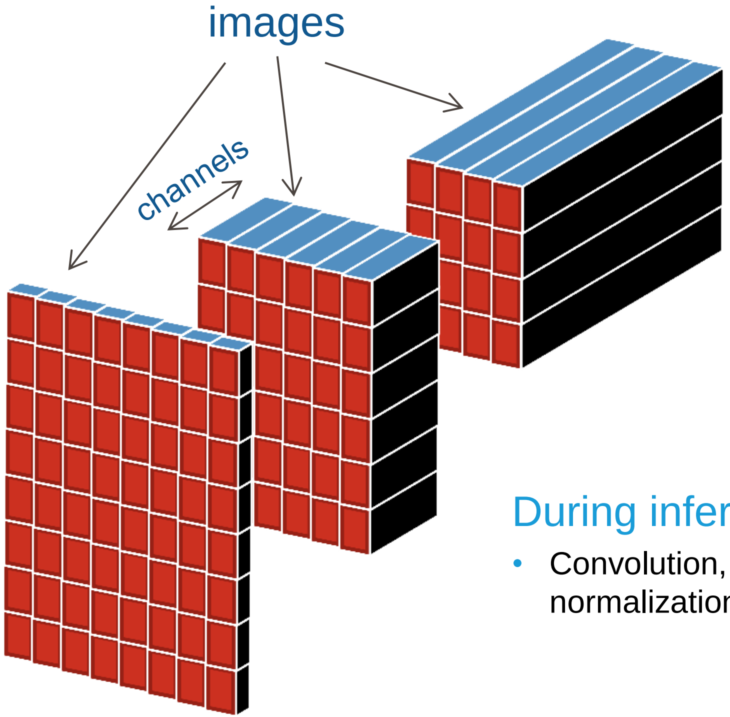
CNN (Convolutional Neural Networks)

- Biologically-inspired DNN with a majority of convolutional layers and a few fully-connected layers
- In a convolutional layer, neuron activation is computed as a convolution between inputs and a set of (height, width) translation-invariant parameters

CNN as a building block for perception

- Regional-CNN (R-CNN), Fast R-CNN, Faster R-CNN for image segmentation and object detection
- CNN features feeding LSTM (Long-Short Term Memory) networks for object detection and tracking

CNN INFERENCE OVERVIEW



State of a CNN layer is a 3D pixel matrix, similar to a 2D image with many channels

- Not just 3 channels as on RGB image but typically between 100 and 1000
- Pixels are in 32-bit or 16-bit IEEE binary float format on mainstream frameworks like Berkeley Caffe
- Frameworks such as Google TensorFlow are moving to 16-bit or 8-bit integer quantization of float formats

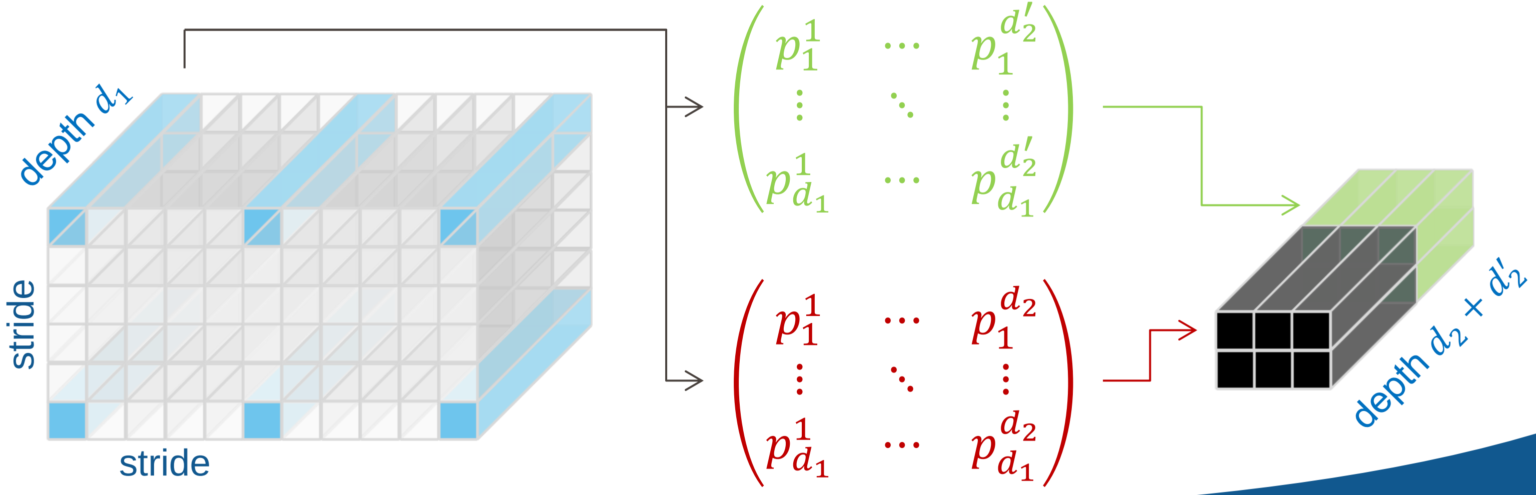
During inference, each layer is used to compute a new image

- Convolution, average, sub-sampling, rectified linear unit (reLU), local response normalization (LRN), softmax, deconvolution, …

CNN INFERENCE ON A MANYCORE PROCESSOR (1)

NxN convolutions decomposed as accumulations of N^2 1x1 convolutions

- 1x1 convolutions can be computed in parallel and accumulated in any order
- Pixels layout is sequential along depth (channels) for dense memory accesses



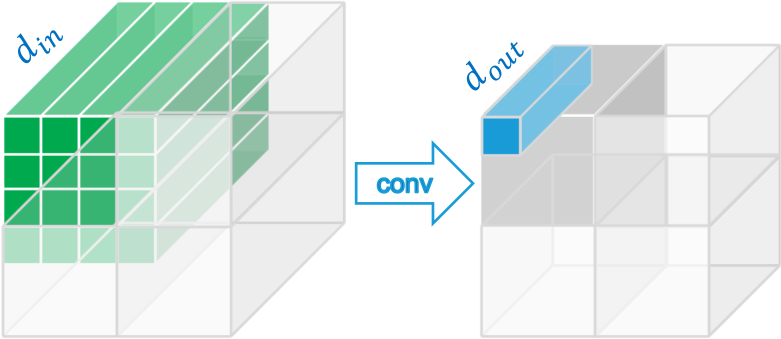
$$\begin{pmatrix} p_1^1 & \cdots & p_1^{d_2'} \\ \vdots & \ddots & \vdots \\ p_{d_1}^1 & \cdots & p_{d_1}^{d_2'} \end{pmatrix}$$

$$\begin{pmatrix} p_1^1 & \cdots & p_1^{d_2} \\ \vdots & \ddots & \vdots \\ p_{d_1}^1 & \cdots & p_{d_1}^{d_2} \end{pmatrix}$$

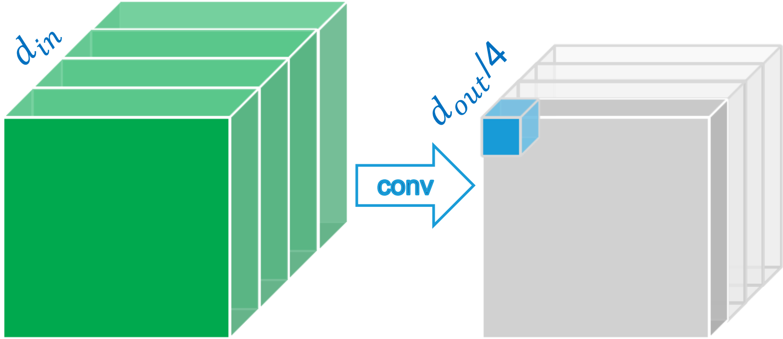
CNN INFERENCE ON A MANYCORE PROCESSOR (2)

Partition images across clusters, splitting along spatial and/or depth dimensions

- Spatial dimension splitting requires that the full set of parameters be loaded from external memory
- Temporal dimension splittig requires access to the whole input image and a subset of the parameters
- NoC multicasting of parameters fosters spatial dimension splitting except for small dimension (e.g. FC)



$[3][3][d_{in}][d_{out}]$



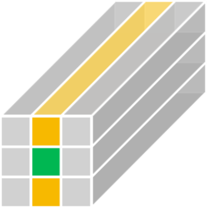
$[3][3][d_{in}][d_{out}/4]$

CNN INFERENCE ON A MANYCORE PROCESSOR (3)

Process layers sequentially, overlapping computations with on-chip communications

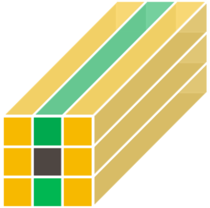
- With spatial dimension splitting, each local memory stores a tile + shadow region of the previous image
- Compute the current image from previous one in 3 steps to overlap execution with shadow region transfers

Step 1



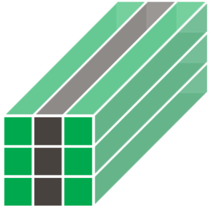
Initial results stored into the output image initialized with biases

Step 2



Intermediate results added to step 1 results and stored into the output image

Step 3



Final results added to step 2 results and stored into the output image

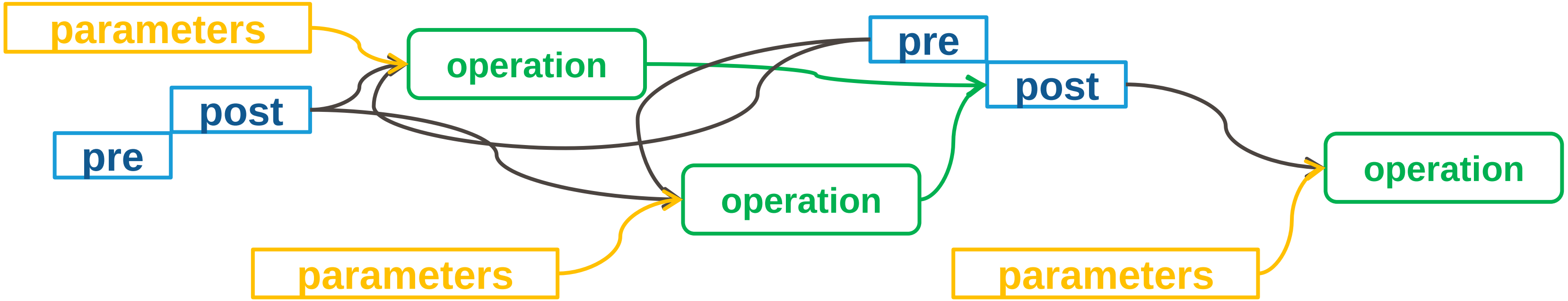
🟨 Pixels being transfered from neighbours compute clusters

🟩 Pixels being processed by the cluster cores

CNN INFERENCE ON A MANYCORE PROCESSOR (4)

Build a local memory buffer allocation and task execution schedule in cluster

- Overlap parameter transfers from external memory with computations on local memory
- Allocation and scheduling are performed on the CNN network, considering an image corresponds to pre and post tasks, and layer compute operations corresponds to a malleable task
- All clusters operate in SPMD (Single Program Multiple Data) to benefit from parameter multicasting by the NoC

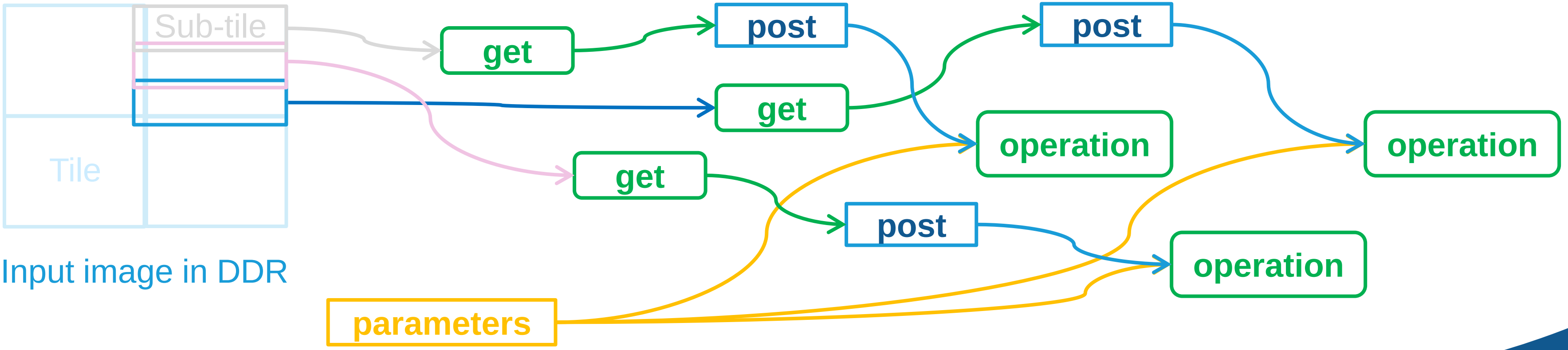


- Pre tasks load biases from external memory into the layer local memory buffer

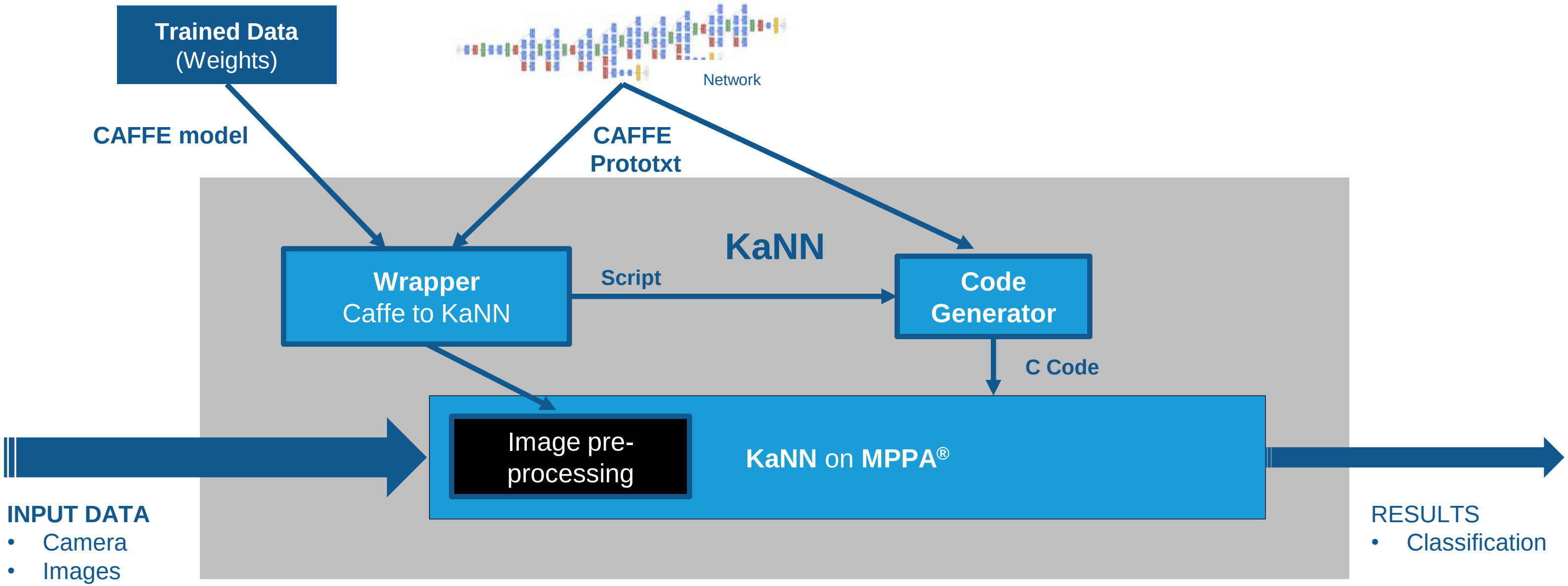
CNN INFERENCE ON A MANYCORE PROCESSOR (5)

For layers where images do not fit on-chip, stream sub-tiles from external DDR memory

- All clusters put (remote write) their tile of output image to DDR memory, then enter a synchronization barrier
- After clusters leave the barrier, they pipeline the get (remote read) from DDR / operate / put to DDR of sub-tiles
- Larger sub-tiles factor more control overhead but reduce the amount of pipelining



KaNN: CNN INFERENCE CODE GENERATOR FOR MPPA®

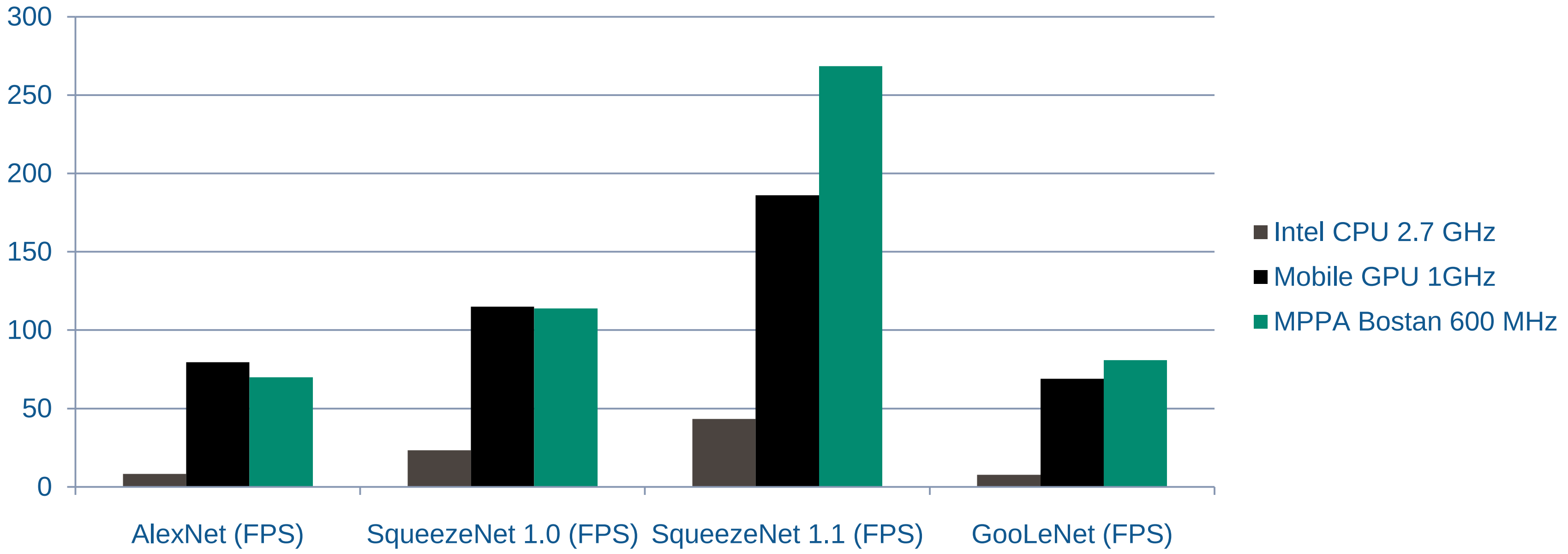


INPUT DATA
- Camera
- Images

RESULTS
- Classification

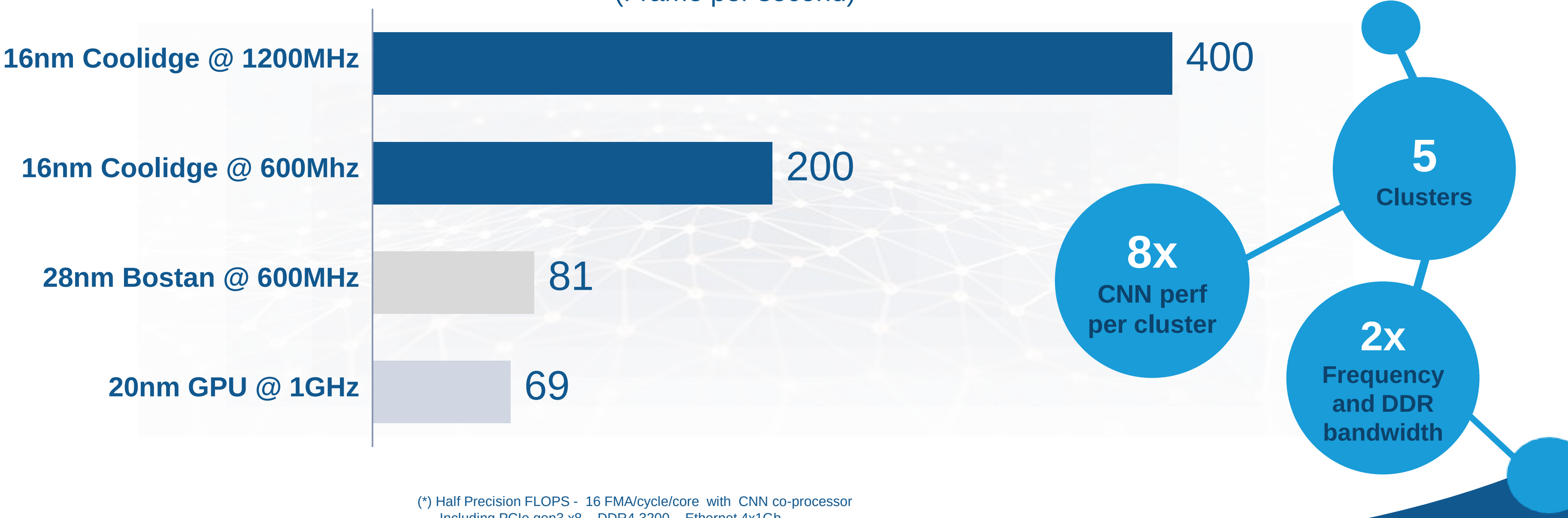
Prototxt: Description of the network (type, layers, kernels,…)

MPPA® BOSTAN VS CPU & GPU ON CNN INFERENCE



MPPA® BOSTAN VS MPPA® COOLIDGE ON CNN INFERENCE

GoogleNet
(Frame per second)



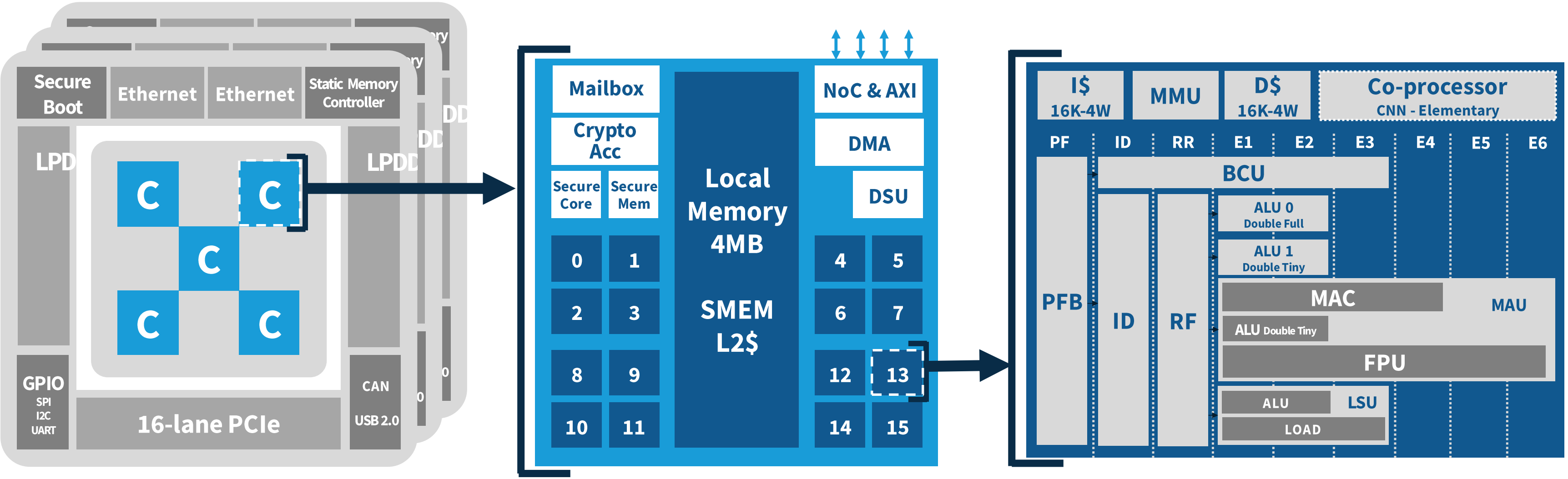
5 Clusters

8x CNN perf per cluster

2x Frequency and DDR bandwidth

(*) Half Precision FLOPS - 16 FMA/cycle/core with CNN co-processor
Including PCIe gen3 x8 – DDR4 3200 – Ethernet 4x1Gb

3rd GENERATION MPPA® MANYCORE ARCHITECTURE (2018)



COOLIDGE PROCESSOR

80+5 cores at 1200 MHz, 16nm

COMPUTE CLUSTER

16+1 cores, 4 MB local memory

5-ISSUE VLIW CORE

64x 64-bit register file

HIGH PERFORMANCES ON MPPA®: BEYOND OPENCL

OpenCL abstract architecture

- OpenCL Device has one global memory shared by Compute Units
- OpenCL Compute Unit has one local memory shared by Processing Elements
- OpenCL Processing Element has a private memory, local and global caches

OpenCL on a manycore processor with local memories

- Work Item executions may be aggregated on each core to expose more ILP
- Using OpenCL **async_work_group_copy** between local & global memories is highly effective

Cannot be expressed in standard OpenCL

- Data transfers between Compute Unit local memories
- Multicasting data transfers between global and local memories
- Direct data transfers across OpenCL devices (global memories)

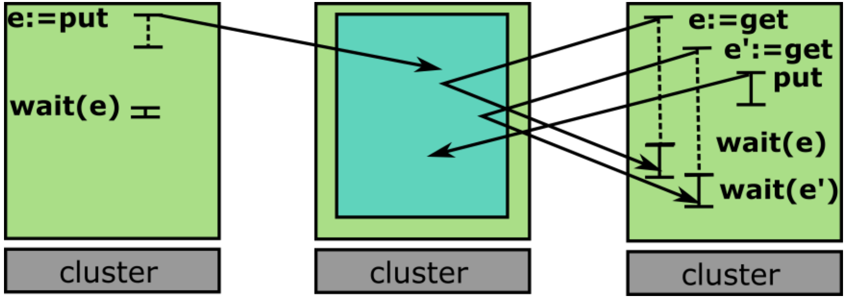
MPPA® ASYNCHRONOUS ONE-SIDED OPERATIONS (1)

Inspired by HPC clusters one-sided communication & synchronization

- Cray SHMEM, ORNL ARMCI, Berkeley GasNet, MPI-3 one-sided communications subset
- Cannot directly reuse these HPC communication libraries because of the MPPA® memory architecture

Asynchronous remote data transfers

- Put (remote DMA write) and Get (remote DMA read) one-sided operations between memory segments
- Operations return immediadely to caller, an event structure can be used to wait/test for local completion
- Collective Put/Get operations leverage NoC multicasting
- Generalization of OpenCL **async_work_group_copy**



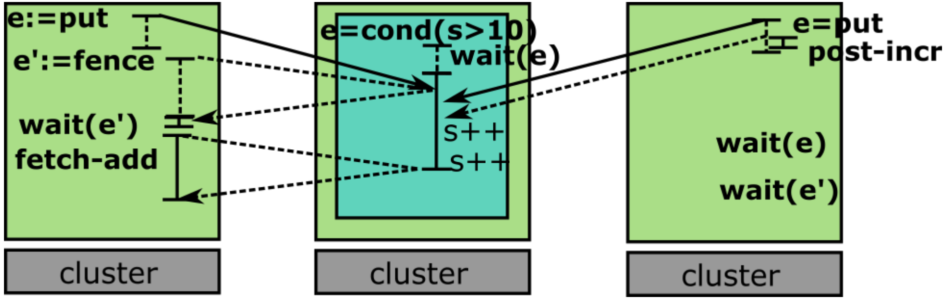
MPPA® ASYNCHRONOUS ONE-SIDED OPERATIONS (2)

Point-to-point synchronization operations

- Fence (global completion), peek, poke, post-add, fetch-clear, fetch-add
- Local wait on the comparison between a local variable and a value
- No busy waiting, lock-free data structures
- Building blocks for efficient barriers and semaphores

Remote queues N to 1

- Push on a remote queue-like memory segment, with atomicity if possible
- Classic distributed synchronization primitive, foundation of active messages



MPPA® EXTENSIONS OF OPENCL TASK PARALLEL MODE

Parts of standard OpenCL that are useful on a manycore processor

- Host program allocates global buffers, creates executables kernels, and dispatches work in queues
- Kernel invocation with a user-defined argument list, which distinguishes between local and global objects

Parts of OpenCL that must be extended for better performances & functionality

- Kernel code written in standard C/C++ and/or assembly language
- Kernel code that exploits CPU-style multi-threading [TI's "OpenMP Dispatch With OpenCL"]
- Kernel code that accesses the local memory of other Compute Units

OpenCL 1.2 extensions for the MPPA® processor

- Use the OpenCL Task Parallel mode to dispatch one Work Group of one Work Item on each cluster
- Kernel code linked with ELF executable uses Pthreads or GCC OpenMP to activate cluster cores
- Kernel code accesses the full asynchronous one-sided communications & synchronizations API

CONCLUSIONS AND PERSPECTIVES

The MPPA® manycore architecture excels on standard CNN inference

- Not only on performance, but also on energy efficiency and time-predictability
- The key is to exploit the high-bandwidth local memory shared by cores in a cluster
- This is achieved by the KaNN code generation tool working from standard Caffe descriptions

Standard OpenCL environment must be extended to enable such performance

- Standard OpenCL Task Parallel mode has been extended to support C/C++, pthreads & OpenMP, and asynchronous one-sided operations between Compute Units (MPPA® compute clusters)
- Resulting programming model is excellent for direct coding and as a code generation target, unlike KPN models

Techniques successfully applied by the KaNN code generator will be reused

- KaNN extensions to 8-bit and 16-bit fixed-point computations as enabled by standard frameworks (TensorFlow)
- KaNN extensions to Binarized Neural Networks (BNN), where convolution becomes POPCOUNT(XNOR)
- OpenVX framework for MPPA® processor under development (to be released end of 2017)

THANK YOU

KALRAY S.A. - GRENOBLE - FRANCE
445 rue Lavoisier,
38 330 Montbonnot - France
Tel: +33 (0)4 76 18 09 18
email: info@kalray.eu

KALRAY INC. - LOS ALTOS - USA
4962 El Camino Real
Los Altos, CA - USA
Tel: +1 (650) 469 3729
email: info@kalrayinc.com

Pictures credits: Kalray, ©Fotolia

KALRAY