Co-Design of the Kalray Manycore Accelerator for Edge Computing

- HiPEAC CSW Autumn 2021

26 October 2021

Benoît Dupont de Dinechin, CTO

www.kalrayinc.com

KALRAY IN A NUTSHELL

Kalray offers a new type of processor and solutions targeting the booming market of intelligent systems.

A Global Presence

- France (Grenoble, Sophia-Antipolis)
- USA (Los Altos, CA)
- Japan (Yokohama)
- Canada (Partner)
- China (Partner)
- South Korea (Partner)

Leader in Manycore Technology

3rd generation of MPPA® processor

~€85m R&D investment

30 Patent families

Industrial investors

NXP

RENAULT NISSAN MITSUBISHI

SAFRAN

MBDA

EURONEXT

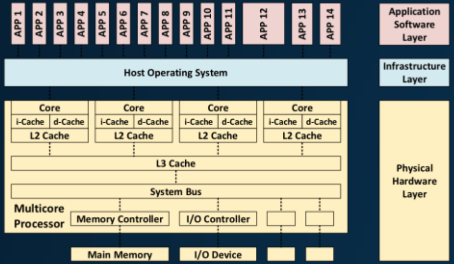
- Public Company (ALKAL)
- Support from European Govts
- Working with 500 fortune companies

KALRAY

Outline

Multicore Processors and Manycore Accelerators

Homogeneous Multicore Processor



Multiple CPU cores sharing a cache-coherent memory hierarchy

- Scalability by replicating CPU cores
- Standard programming models

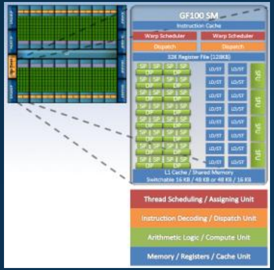
Energy efficiency issues

- Global cache coherence scaling

Time-predictability issues

- No scratch-pad or local memories

GPGPU Manycore Accelerator


GF100 SM
Instruction Cache
Warp Scheduler
Dispatch
Register File
L1 Cache / Shared Memory
Thread Scheduling / Assigning Unit
Instruction Decoding / Dispatch Unit
Arithmetic Logic / Compute Unit
Memory / Registers / Cache Unit

Multiple Streaming Multiprocessors (SMs)

- Restricted programming models

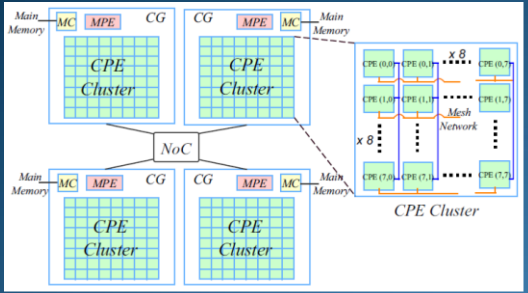
Performance issues of 'thread divergence'

- Branch divergence of PEs inside a 'warp'
- Memory divergence: non-coalesced accesses

Time-predictability issues

- Dynamic allocation of thread blocks to SMs
- Dynamic scheduling of warps inside a SM

CPU-Based Manycore Accelerator



Multiple "Compute Units" connected by a network-on-chip (NoC)

- Scalability by replicating Compute Units
- Standard multicore programming inside a Compute Unit

Compute Unit

- Group of cores + DMA engines
- Scratch-pad memory (SPM)
- Local cache coherency

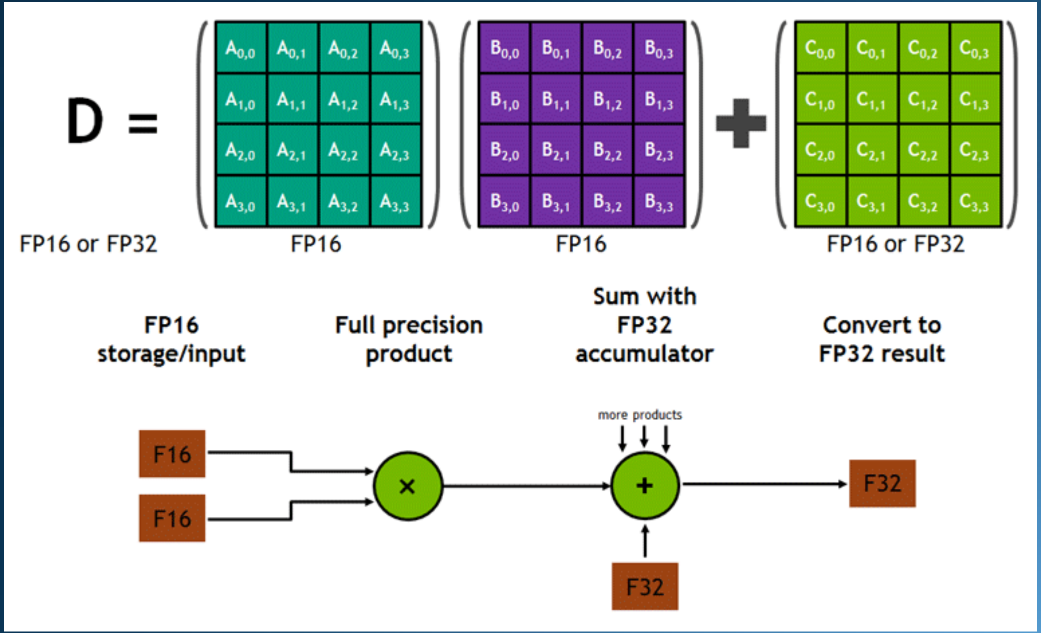
GPGPU Tensor Cores for Deep Learning

NVidia Volta architecture (2017)

- 64x FP32 cores per SM
- 32x FP64 cores per SM
- 8x Tensor cores per SM

Tensor core operations

- Tensor Core perform D = A x B + C, where A, B, C and D are matrices
- A and B are FP16 4x4 matrices
- D and C can be either FP16 or FP32 4x4 matrices
- Higher performance is achieved when A and B dimensions are multiples of 8
- Maximum of 64 floating-point mixed-precision FMA operations per clock



$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$
FP16 or FP32
FP16
FP16
FP16 or FP32
FP16 storage/input
Full precision product
Sum with FP32 accumulator
Convert to FP32 result
F16
F16
×
more products
+
F32
F32

KALRAY

Manycore Accelerator Compute Unit Co-Design

	Design Choice	Advantages	Issues
Processing Engines	Single-core	Simple memory hierarchy	Limited performances
	Multi-core	**OpenMP3 / Pthread multi-threading inside Compute Units**	**Multi-banked local memory**
	Core multi-threading	Overlap compute & transfers	Requires more registers and local memory capacity
Local Memory	**Sratch-pad memory**	**Energy-efficient, deterministic**	**Exposed only by OpenCL and OpenVX Motivates a RDMA engine**
	Local cache coherence	**Required by OpenMP and PThread programming**	**Non time-predictable, must be disabled for hard real-time**
	Global cache coherence	Multi-core programming model across compute units	Not energy-efficient & not scalable [Proxy Architectures for Exascale]
	Global memory addressing	Scalable, applied by GPGPUs	Atomic operations are more difficult to implement
Global Memory	**Semi coherent with host cores**	**Enough for OpenCL support, OpenMP offloading possible**	**Support by system interconnect and cache coherency**
	Fully coherent with host cores	Simpler OpenMP offloading	Emerging standards CCIX and CXL; CXL requires PCIe Gen5
	Hardware prefetch engine	May improve performances	Less energy-efficiency than RDMA engines on prescribed addresses

KALRAY

MPPA®3 Manycore Processor

5 Compute Units, 80 Accelerated VLIW Cores with Tensor Coprocessor

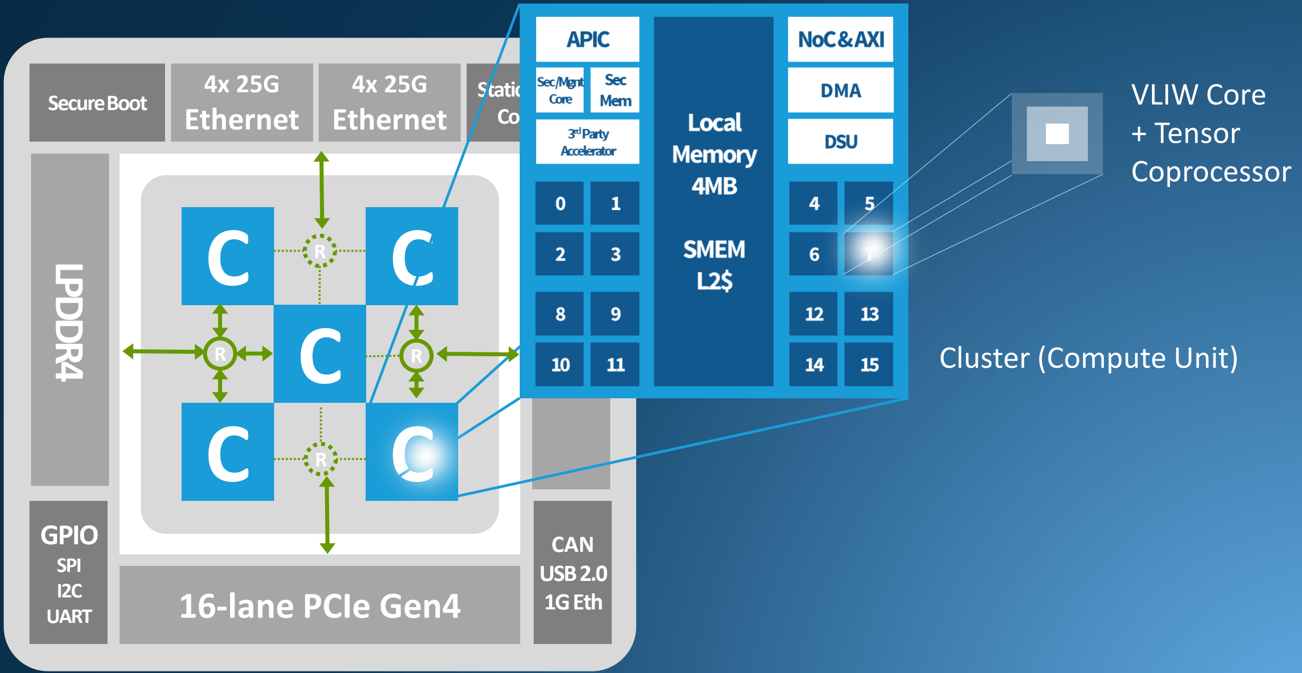
Peak Performances
200KMIPs, 25 DL TOPS at 1.2GHz

Power efficiency
40W Typical

High Speed I/F
200Gbs Ethernet, 16x PCIe Gen4

Functional Safety & Cyber-Security
Secure Islands, Secure Boot

Programming
Control Plane – Linux – 16 cores
Data Plane - 64 cores



Mapping Functions to Compute Units

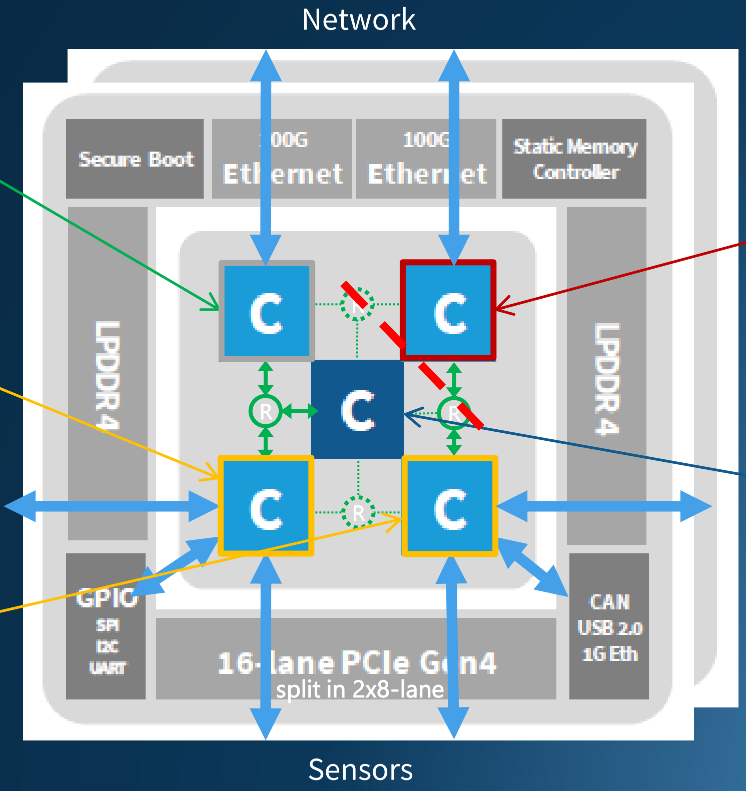
Secured communications

Machine Learning

Embedded HPC

Hard real-time application

Rich OS environment



KALRAY

Outline

1. Manycore Accelerators
2. Edge Computing
3. MPPA3 Processor and IP
4. Accelerator Offloading
5. Outlook & Conclusions

Edge Computing Definitions

Intel (https://www.intel.com/content/www/us/en/edge-computing)

What Is an Edge Device?

Edge computing solutions place Internet of Things (IoT) devices, gateways, and computing infrastructure as close as possible to the source of data

Types of Edge Devices

- Intelligent edge devices offer capabilities like onboard analytics or AI.
- Intelligent edge devices used in manufacturing may include vision-guided robots or industrial PCs
- Digital cockpit systems built into commercial vehicles can help support driver assistance

NVIDIA (https://blogs.nvidia.com/blog/2019/10/22/what-is-edge-computing/)

What Is Edge Computing?

Edge computing is the concept of capturing and processing data as close to the source of the data as possible via processors equipped with AI software

What Are the Benefits of Edge Computing?

- Reduced latency: bringing AI computing to where data is generated
- Improved security: the need to send sensitive data to the public cloud is decreased
- Greater range: edge computing processes data without internet access

Intelligent Systems for Edge Computing

Cyber-physical systems

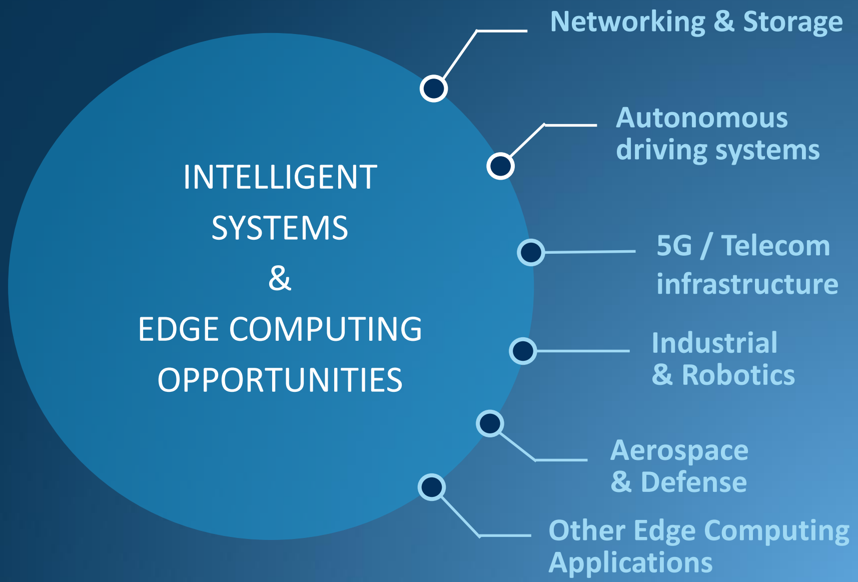
- Information processing and physical processes are tightly integrated
- Time constraints associated with information manipulation
- Functional safety and cyber-security
- Distributed systems (over Ethernet)

Artificial intelligence

- The science and engineering of creating intelligent machines (J. McCarthy, 1956)
- Mostly Machine Learning, in particular Deep Learning [Multiple processing layers to learn representations of data with multiple levels of abstraction -- Yann Le Cun et al., 2015]

Intensive computing

- Image, signal, numerical, crypto/Galois, graphs


INTELLIGENT SYSTEMS & EDGE COMPUTING OPPORTUNITIES
Networking & Storage
Autonomous driving systems
5G / Telecom infrastructure
Industrial & Robotics
Aerospace & Defense
Other Edge Computing Applications

Kalray K200™-LP Networking & Storage Acceleration Card

HIGH AVAILABILITY

DATA PROTECTION

SECURITY

DATA MANAGEMENT

FLASHBOX

STORAGE EFFICIENCY

FULLY PROGRAMMABLE
ACS Storage Framework

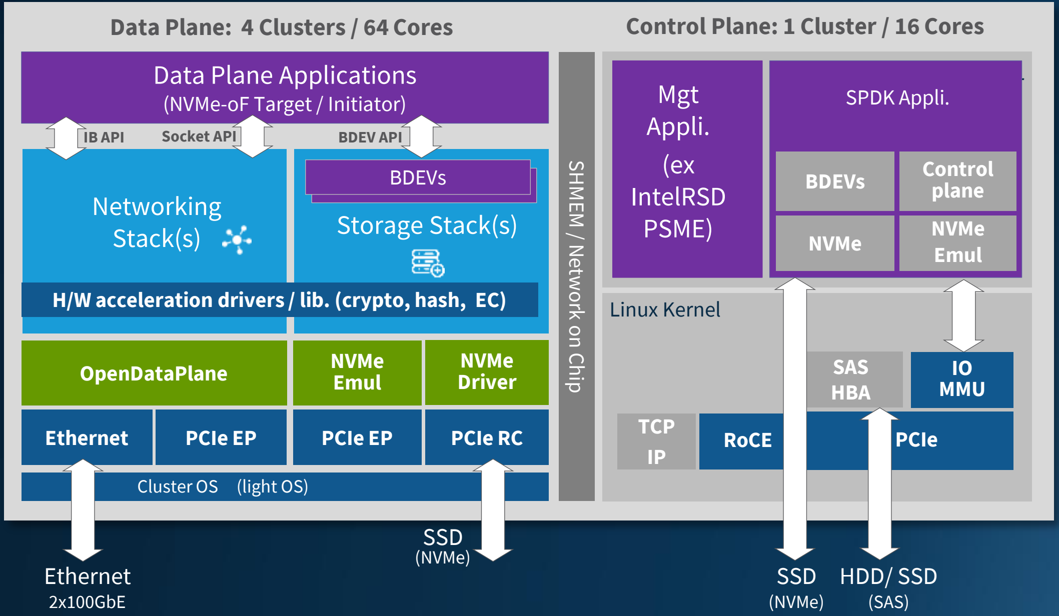
HIGH-SPEED I/O INTERFACES
NVMe-oF TCP & RoCE
(2x100GbE)

ZERO HOST CODE IMPACT
X86 transparent integration via NVMe emulation
Or no X86

HIGH-PERF SSDs I/F
(RC or P2P PCIe Gen4)

LOW CONSUMPTION
30W Typ.

AccessCore Storage Software Based on SPDK Optimized for Kalray Manycore



5 SPDK instances, one per Cluster

- 1x Linux for control plane, and also for non accelerated protocols (ex iSCSI)
- 4x instances on lightweight POSIX OS for data plane (I/O queues only)
- Support standard SPDK APIs for easy added value service addition
- Single SPDK instance from external management point of view

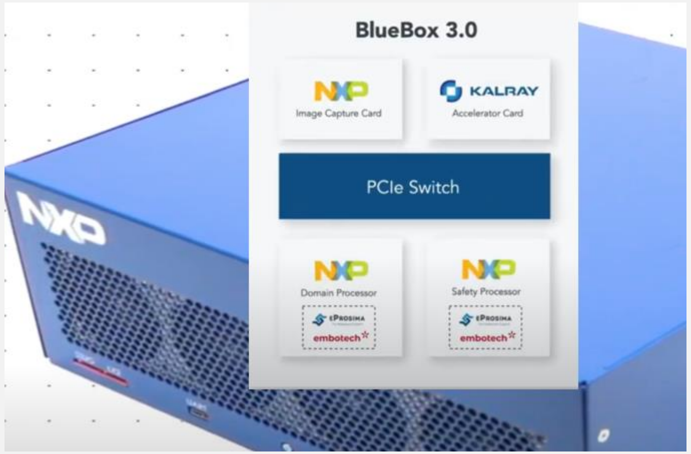
Advanced storage services

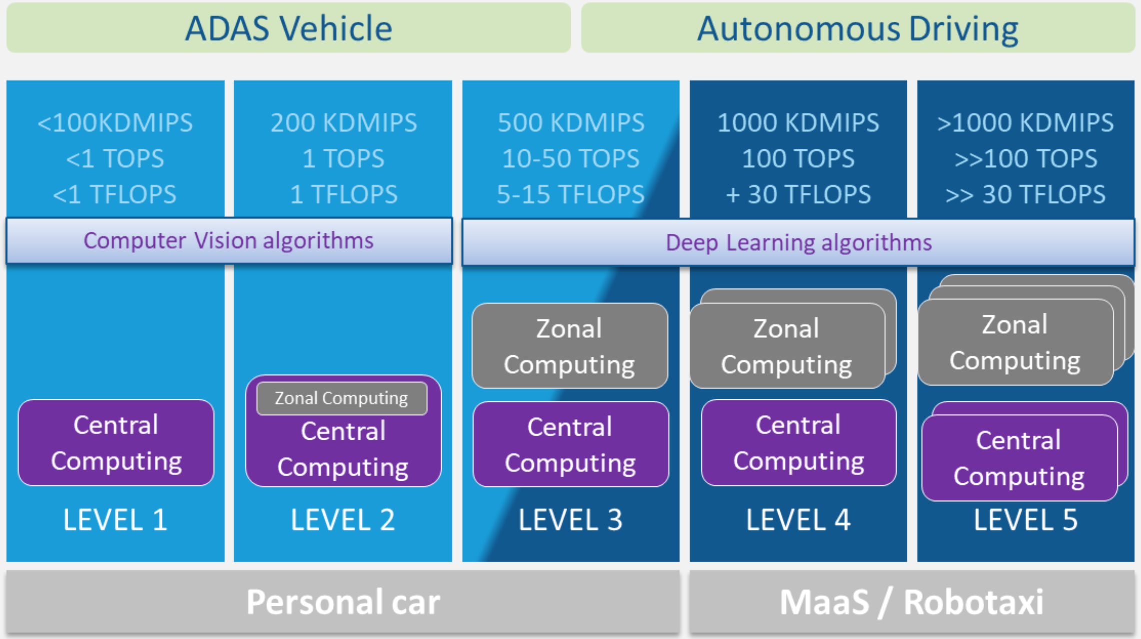
- Logical volume management
- Snapshots and clones
- Erasure coding, RAID 0, 1 and 6
- High-availability with 2 cards in Active/Passive configuration
- Encryption, data compression

Edge Computing in Automated Cars

NXP BlueBox 3.0 (L2 to L4)

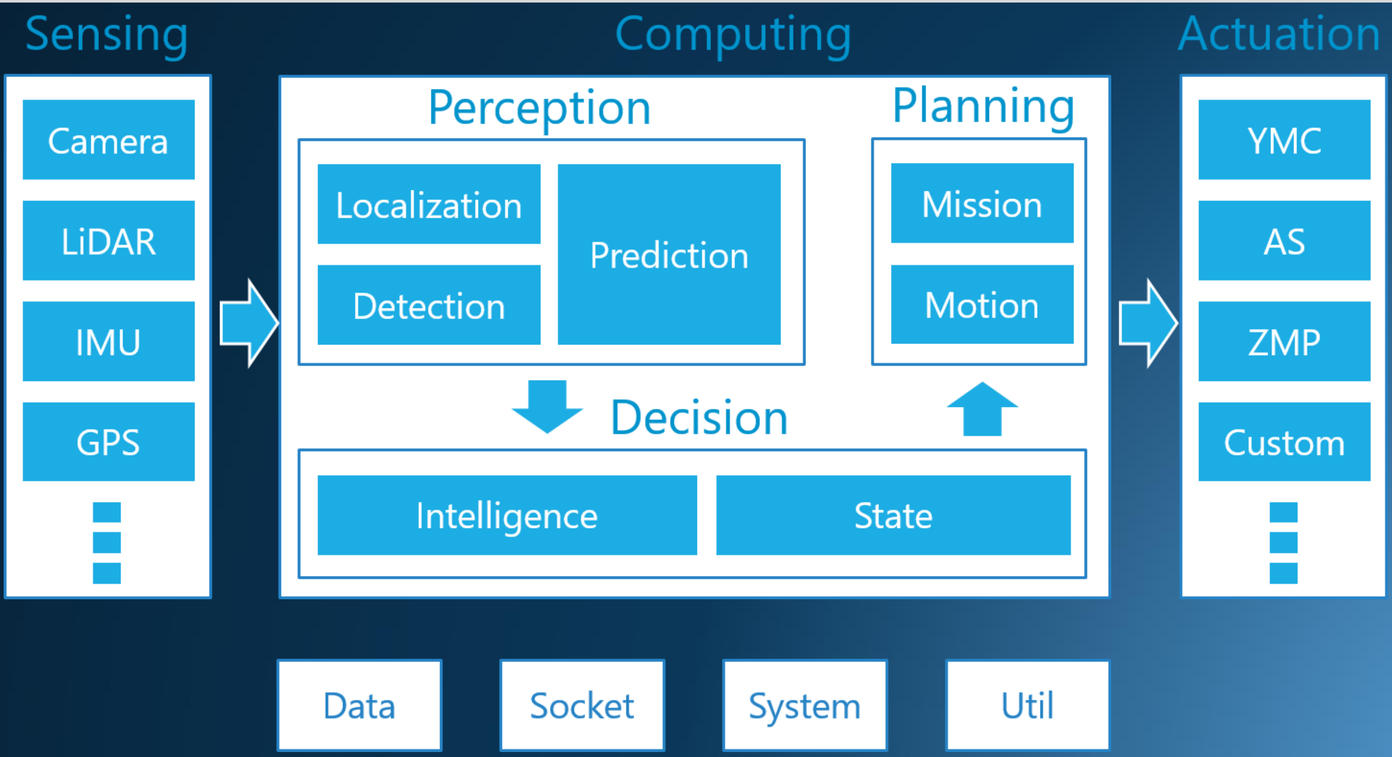
NXP Semiconductors proposes a compute platform for ADAS and Autonomous Driving Systems accelerated by MPPA®3 Coolidge™ processor.





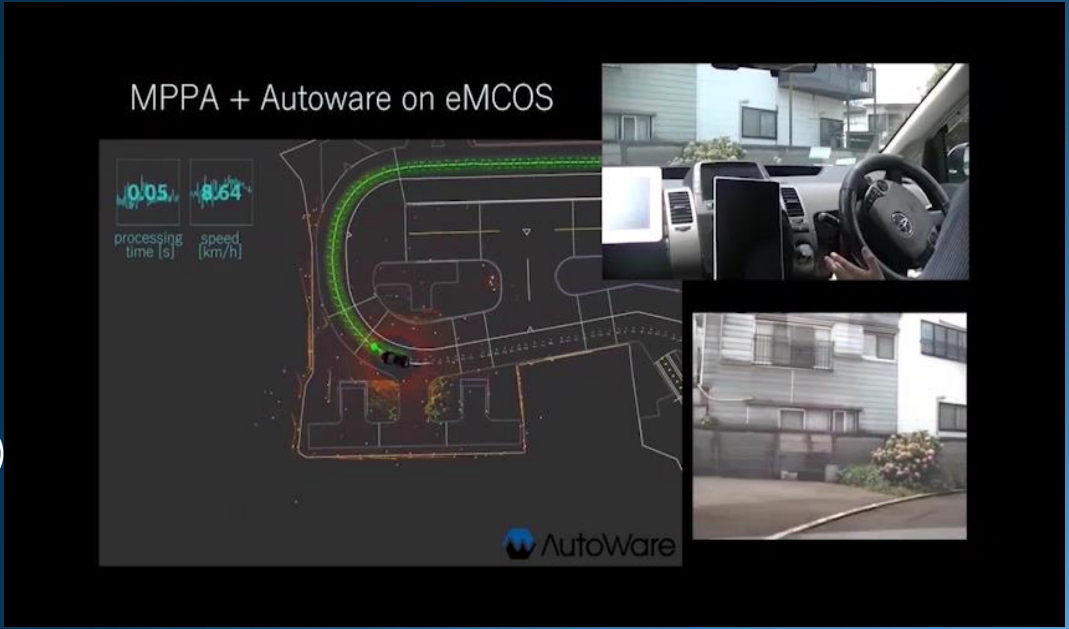
Autoware.Auto

[An open-source software stack based on ROS 2 for self-driving]



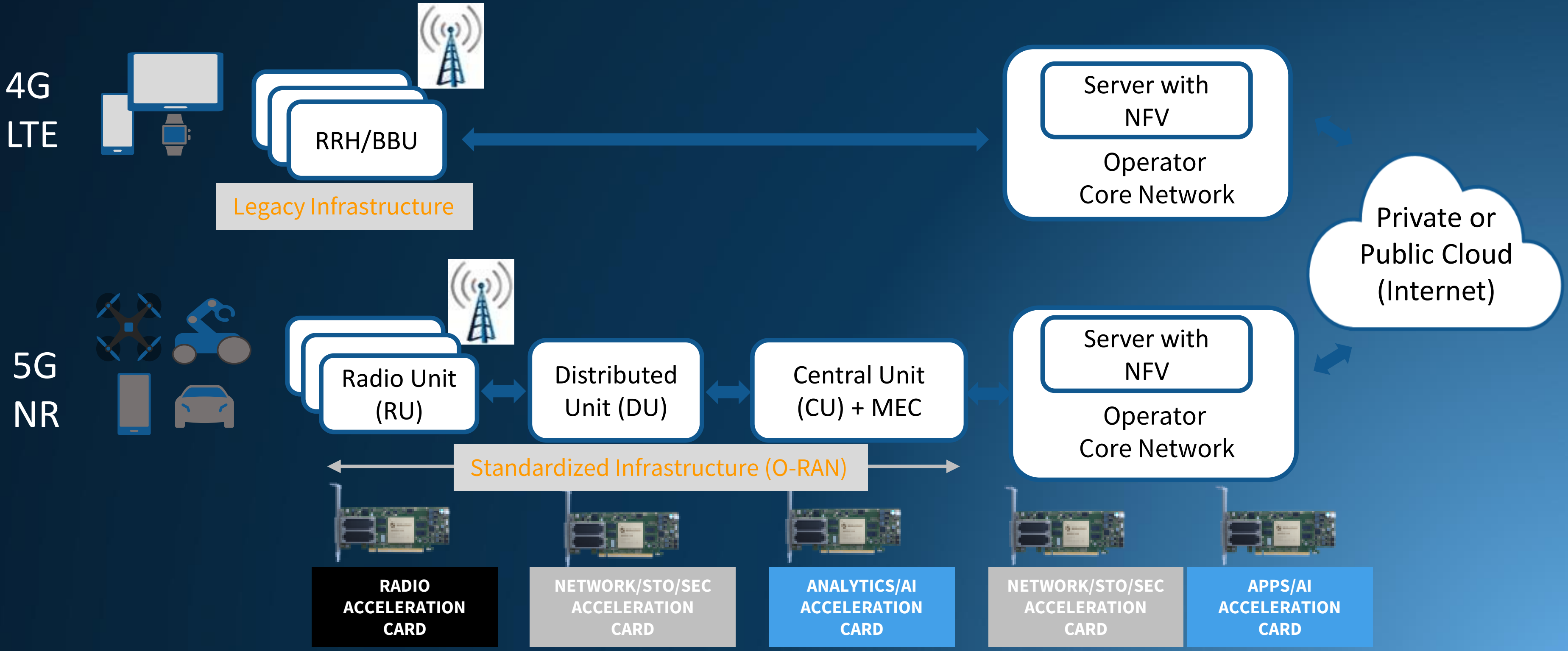
Perception and Path Planning Acceleration

- **Camera & Lidar Perception**
  - Computer vision (OpenCV, OpenVX)
  - CNN Object detection (YOLOv3)
  - Normal Distribution Transform
- **Path Planning Acceleration**
  - Occupancy Grid Mapping (OpenMP)
  - Polynomial Trajectory Generation
  - Extended Kalman Filters (C++ Eigen)
- **System Environment**
  - ARM multicore host processor
  - eSOL eMCOS (POSIX) on MPPA
  - Autoware/ROS/DDS ROS



KALRAY

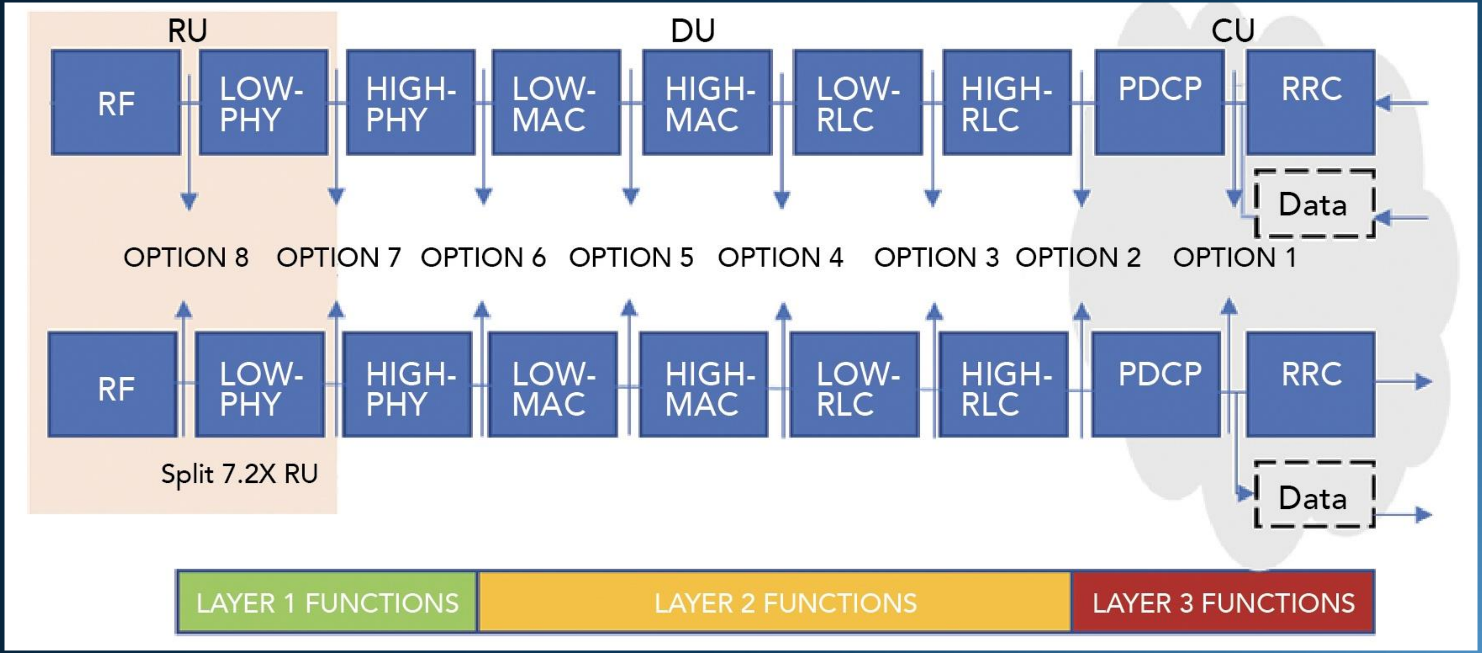
Open Radio Access Network (O-RAN) for 5G



KALRAY

O-RAN Functional Split Options for 5G

[5G Technology World]



Radio Resource Control (RRC)	Service Data Adaptation Protocol (SDAP)	Packet Data Convergence Protocol (PDCP)	Radio Link Control (PDCP)	Medium Access Control (MAC)	Physical Layer (PHY)	Radio Frequency (RF)

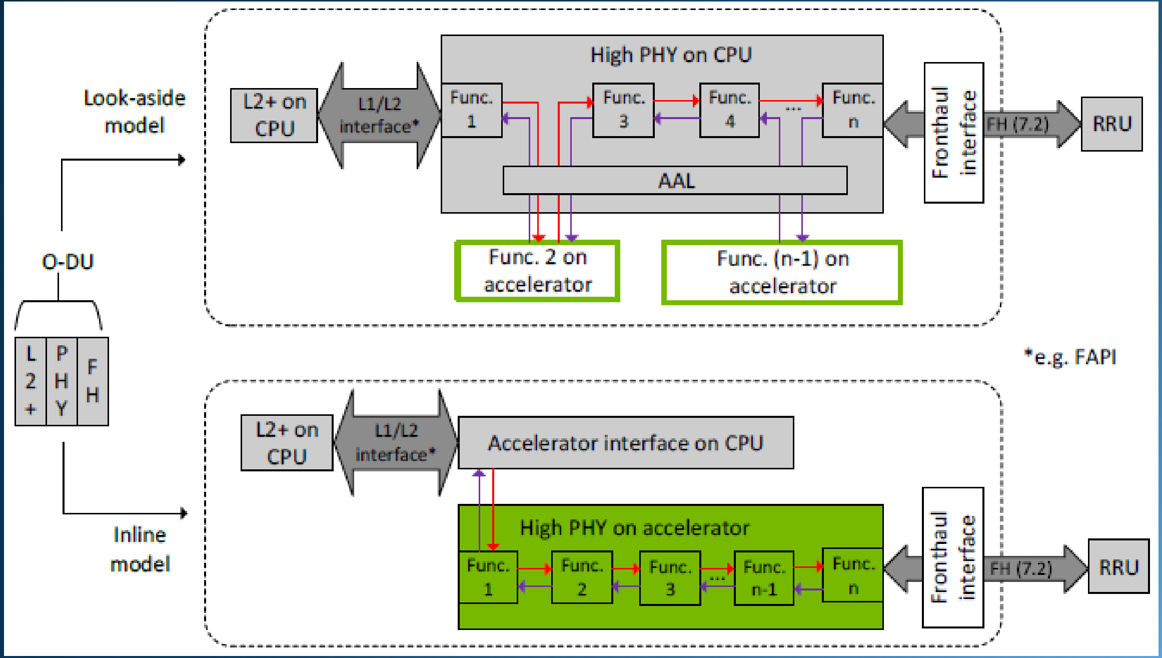
KALRAY

O-RAN Distributed Unit (DU) Acceleration
[O-RAN Cloud Platform Reference Designs]

DU system environment
- Receives eCPRI frames from RU over Ethernet/PTP
- F1 interface between DU and CU over GTP-U/UDP/IP and SCPT/IP

Look-Aside acceleration
- Typically FPGA or ASIC
- Acceleration Abstraction Layer

Inline acceleration
- Software-programmable accelerator processes from Upper-PHY to Lower-MAC (HARQ)

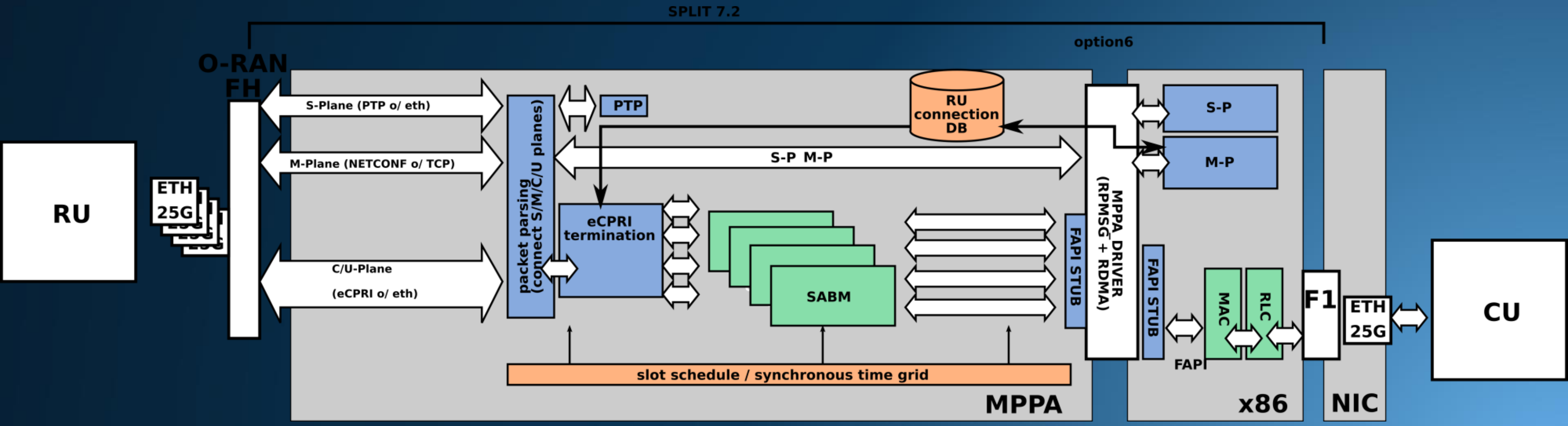


KALRAY

MPPA®3 Inline Accelerator for Time-Critical DU Functions

Leverage the time-predictability of manycore architecture for real-time L1 functions

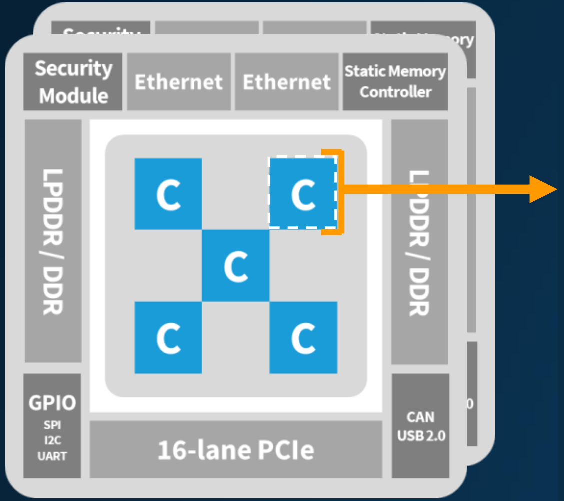
- MPPA3 accelerator terminates the eCPRI fronthaul network and implements L1/L2 interface (FAPI) through PCIe
- General-purpose processor on the DU (e.g. x86) execute L2 functions and connects to CU through F1 interface
- Heaviest processing on the MPPA accelerator is QC-LDPC Decoding and Channel Estimation



KALRAY

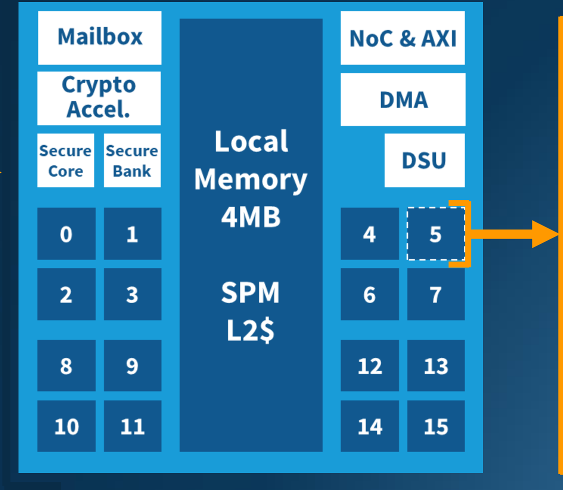
Outline

Kalray MPPA®3 Manycore Processor (80 PEs @ 1GHz)



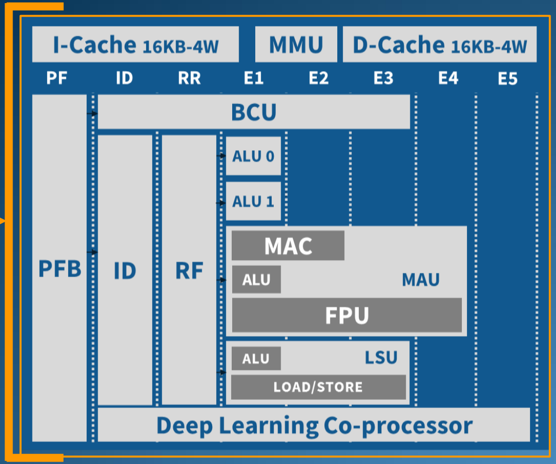
COOLIDGE PROCESSOR

5 compute clusters at 1000 MHz
2x 100Gbps Ethernet, 16x PCIe Gen4



COMPUTE CLUSTER

16+1 cores, 4 MB local memory
NoC and AXI global interconnects



6-ISSUE VLIW CORE

64x 64-bit register file
128MAC/c tensor coprocessor

C Compute Cluster (Compute Unit)

Very Long Instruction Word (VLIW) Architectures

Energy-efficient, time-predictable instruction-level parallel execution

Classic VLIW architecture (J. A. Fisher)

- SELECT operation on Boolean value
- Conditional load/store/FPU operations
- Dismissible loads (non-trapping)
- [Multi-way conditional branches]

Key compiler techniques

- Trace scheduling (global instruction scheduling)
- Partial predication (S. Freudenberger algorithm)

Main examples

- Multiflow TRACE processors
- HP Labs Lx « Embedded Computing: a VLIW Approach »
- STMicroelectronics ST200 (media processor based on Lx)

EPIC VLIW architecture (B. R. Rau)

- Fully predicated ISA
- Speculative loads (control speculation)
- Advanced loads (data speculation)
- Rotating registers

Key compiler techniques

- Modulo scheduling (software pipelining)
- Full predication (R-K algorithm, J. Fang algorithm)

Main examples

- Cydrome Cydra-5
- HP-intel IA64
- TI C6x DSPs

KALRAY

MPPA®3 64-Bit VLIW Core

VLIW architecture co-designed for compilers to appear as an in-order superscalar core

Vector-scalar ISA

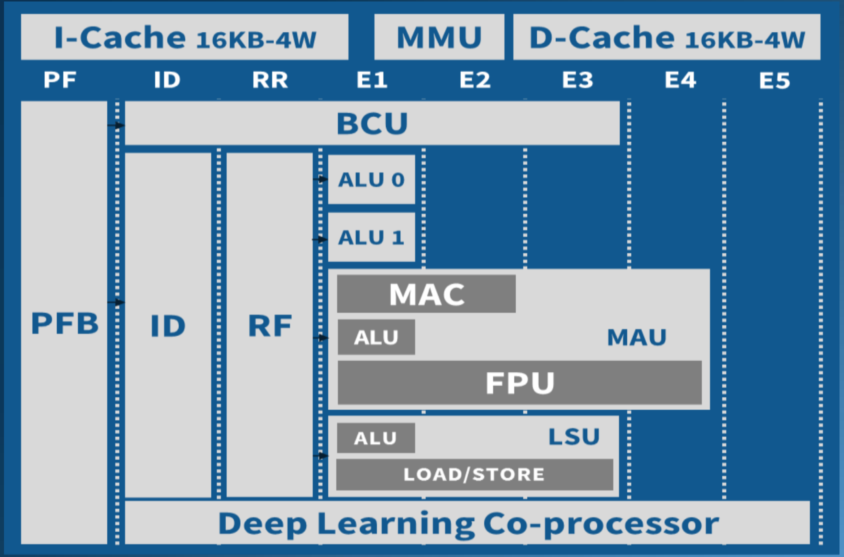
- 64x 64-bit general-purpose registers
- Operands can be single registers, register pairs (128-bit) or register quadruples (256-bit)
- 128-bit SIMD instructions by dual-issuing 64-bit on the two ALUS or by using the FPU datapath

DSP capabilities

- Counted or while hardware loops with early exits
- Non-temporal loads (L1 cache bypass)
- Non-trapping memory loads

CPU capabilities

- 4 privilege levels (rings), MMU (runs Linux kernel)
- Recursive virtualization (Popek & Goldberg)



VLIW CORE PIPELINE

MPPA®3 Tensor Coprocessor

Matrix-oriented arithmetic operations

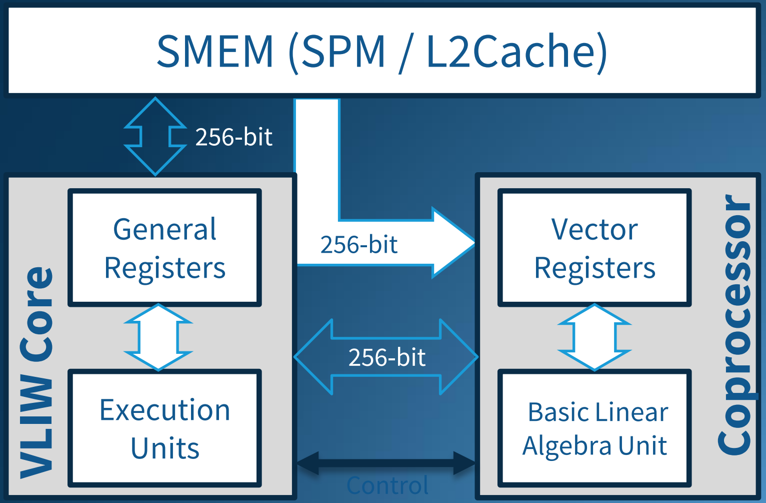
- Separate 48x 256-bit wide vector register file
- Any coprocessor operand (1, 2 or 4 registers) is interpreted as a submatrix with four rows and a variable number of columns

Full integration into core instruction pipeline

- Extended VLIW core ISA with extra issue lanes
- Move instructions supporting matrix-transpose
- Register dependency / cancel management
- Memory directly accessible from coprocessor

Load-scatter memory operations

- Avoids the complexities of Morton memory indexing (Z-patterns for memory data layout)



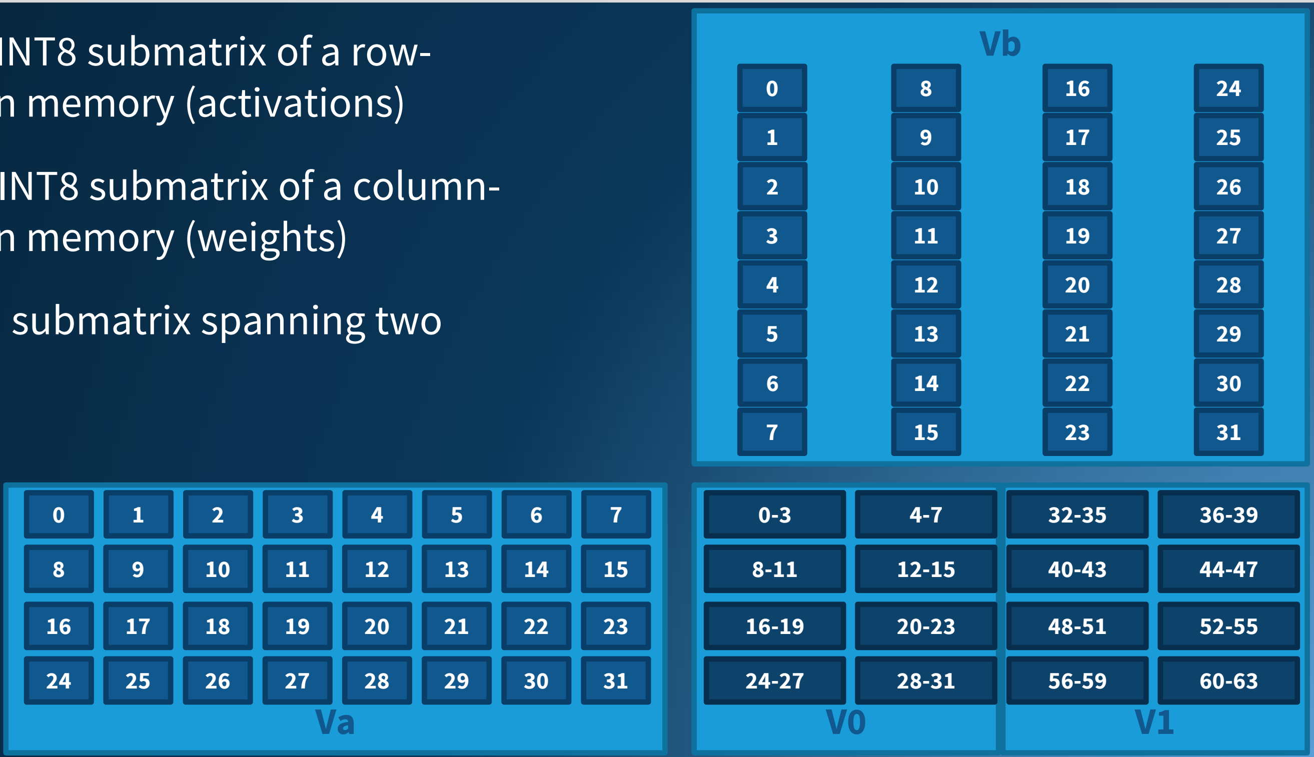
Coprocessor INT8.32 Matrix Multiply-Accumulate

Operand Va is a 4x8 INT8 submatrix of a row-major order matrix in memory (activations)

Operand Vb is a 8x4 INT8 submatrix of a column-major order matrix in memory (weights)

Result is a 4x4 INT32 submatrix spanning two registers V0 and V1

(numbers indicate byte index in 32-byte coprocessor registers)

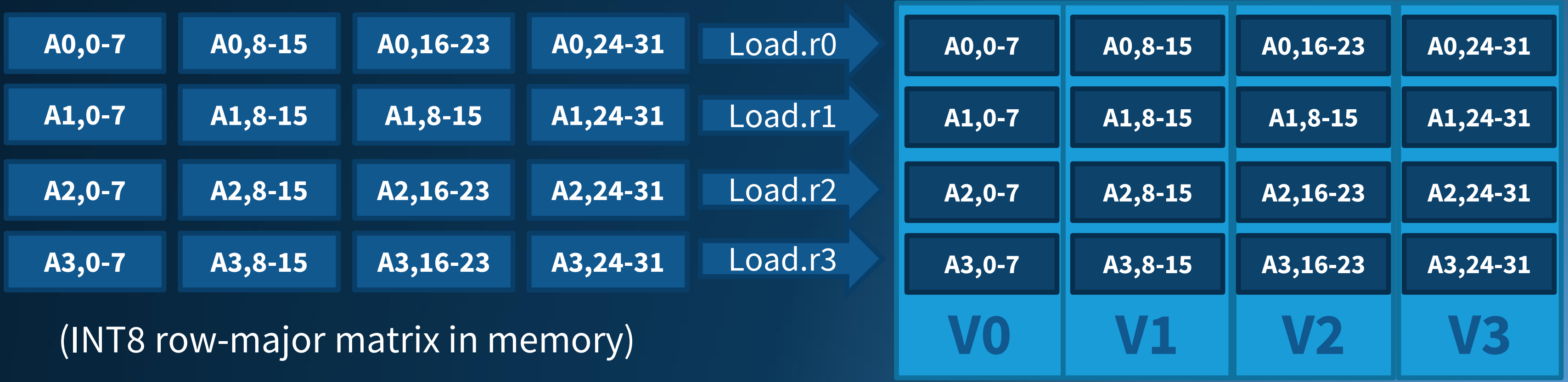


KALRAY

Load-Scatter Coprocessor Memory Operations

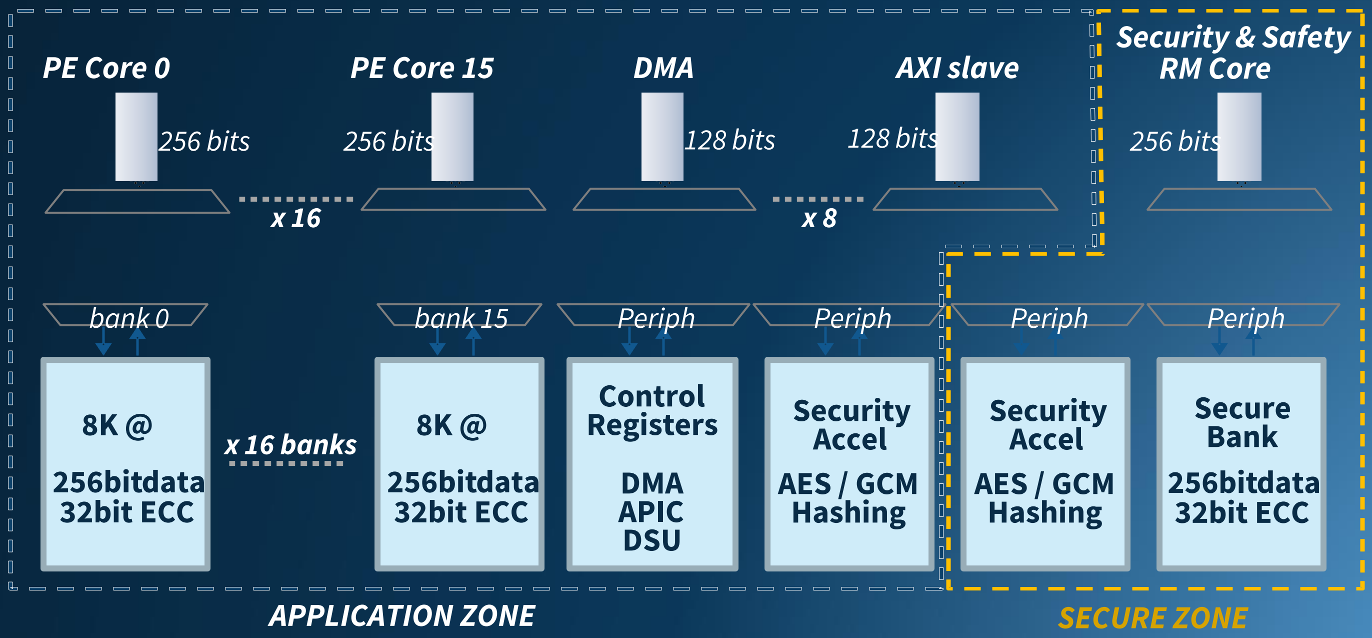
Support the invariant that any coprocessor operand (1, 2 or 4 registers) is interpreted as a submatrix with four rows and a variable number of columns

Avoids the complexities of Morton memory indexing (Z-patterns for memory data layout)



(INT8 row-major matrix in memory)

MPPA®3 Cluster (Compute Unit)



Security & Safety RM Core
PE Core 0
PE Core 15
DMA
AXI slave
256 bits
256 bits
128 bits
128 bits
256 bits
x 16
x 8
bank 0
bank 15
Periph
Periph
Periph
Periph
8K @ 256bitdata 32bit ECC
x 16 banks
8K @ 256bitdata 32bit ECC
Control Registers DMA APIC DSU
Security Accel AES / GCM Hashing
Security Accel AES / GCM Hashing
Secure Bank 256bitdata 32bit ECC
APPLICATION ZONE
SECURE ZONE

MPPA®3 Memory Hierarchy

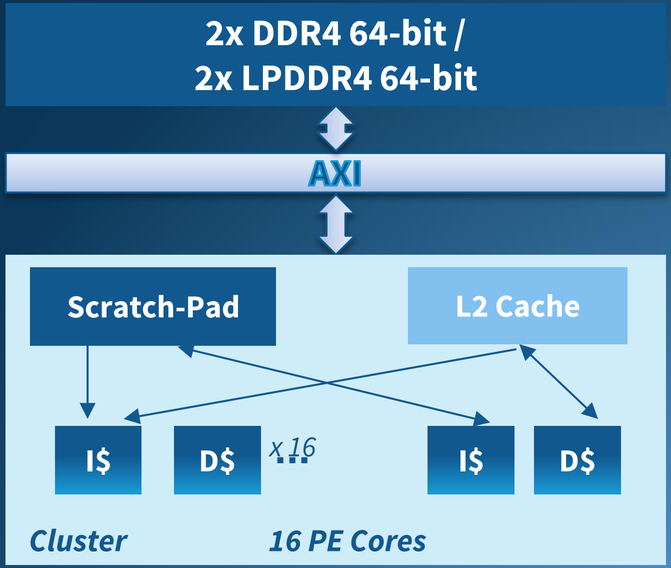
Memory model adapted to OpenCL and to multi-node ROS

VLIW Core L1 Caches

- 16KB / 4-way LRU instruction cache per core
- 16KB / 4-way LRU data cache per core
- 64B cache line size
- Write-through, write no-allocate (write around)
- Coherency configurable across all L1 data caches

Cluster L2 Cache & Scratch-Pad Memory

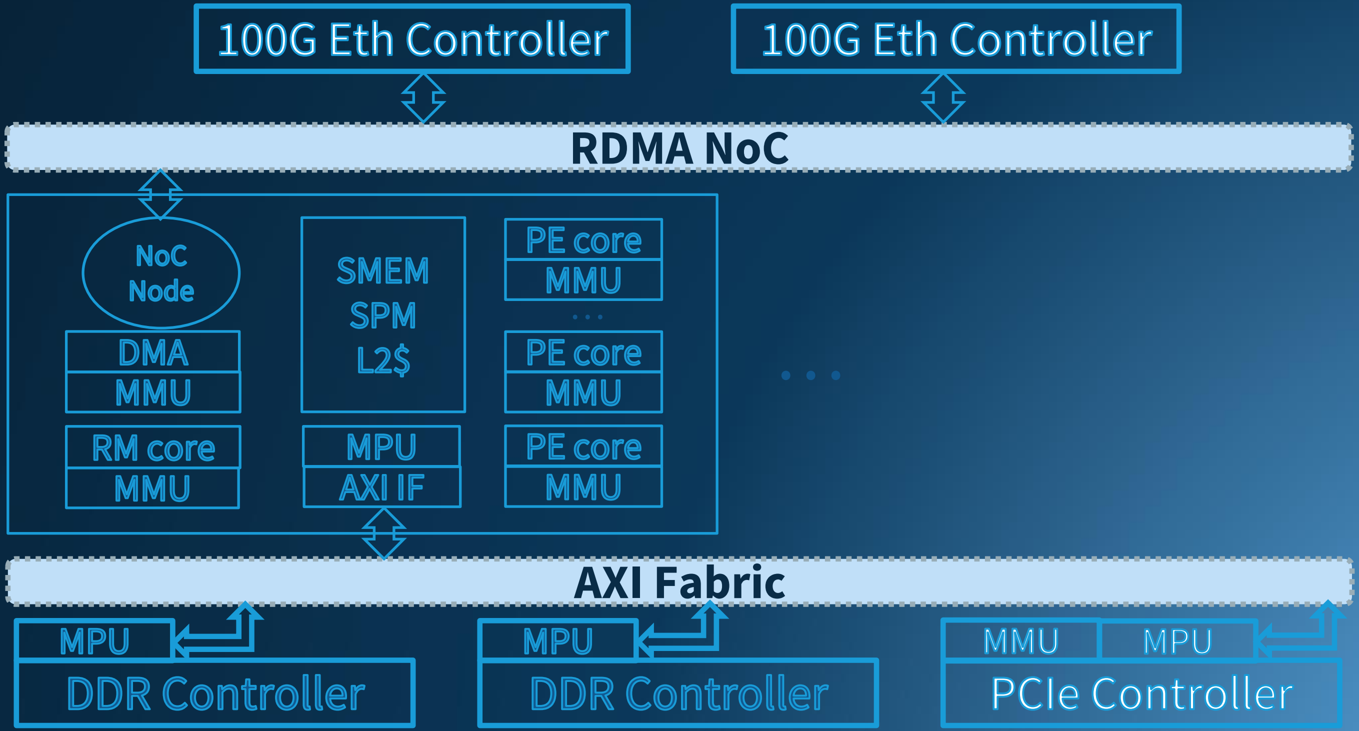
- Scratch-pad memory from 2MB to 4MB
  - 16 independent banks, full crossbar
  - Interleaved or banked address mapping
- Configure Cluster L2 cache from 0MB to 2MB
  - 16-way Set Associative
  - 256B cache line size
  - Write-back, write allocate



L1 cache coherency	L2 cache coherency
enable /disable	Not enabled Across clusters

KALRAY

MPPA®3 Global Interconnects



MPPA®3 v2/IP Processing Element Improvements

VLIW Core

Vector-scalar ISA

- More capable vector shuffle, insert, extract
- SIMD instructions for 8-bit element vectors
- Masked load and stores at byte granularity

FPU capabilities

- 256-bit x 256-bit + 128-bit → 128-bit
- 256-bit op 256-bit → 128-bit
- FP16x8 SIMD 16 x 16 + 16 → 16
- 4x FP32 FDMDA (16 FP32 operations/cycle)
- FP32 Matrix Multiply Accumulate 2x2x2

Tensor Coprocessor

Datapath improvements

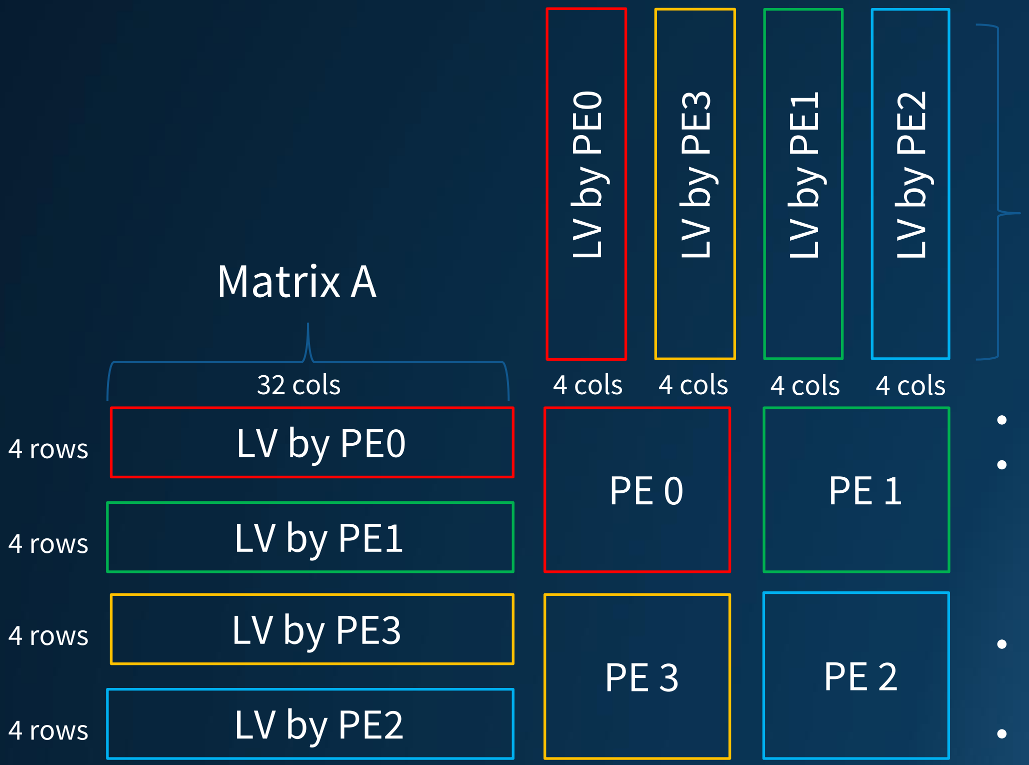
- Extend register file to 64x 256-bit (was 48x)
- Load from cache in addition to load bypass cache
- Gather-store to complement existing scatter-load
- 256-bit ring communication between 4 PEs
- Using TCA register as load stream buffer

Basic Linear Algebra Unit

- Improve 2x INT8.32 performance to 256 MAC/cycle
- Improve 8x FP16.32 performance to 128 FMA/cycle
- Add x8 hybrid MAC FP32*INT32 + INT32 → INT32

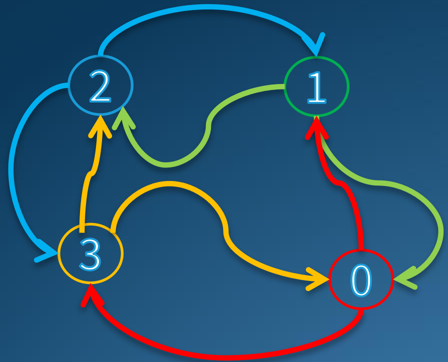
KALRAY

PE to PE Communication for Tensor Operations



Matrix B

32 rows



- **New INT8.32 operation (4x16) . (16x4) += 4x4**
- Macro-scheme executed by 4 PEs
  - 8 256-bit memory loads per PE
  - 8 256-bit data exchanges per PE
  - 8 matrix multiply-add per PE
- Matrix A and B are loaded by quarter by each PE which exchange one quarter with 2 different PEs
- Kernel for INT8.32: **(16 x 32) . (32 x 16) += 16 x 16**

Outline

MPPA® High-Performance Programming Models

STANDARD PROGRAMMING ENVIRONMENTS

OPENCL 1.2 Programming

Standard accelerator programming model

- POSIX host CPU accelerated by MPPA device (OpenAMP interface)
- OpenCL 1.2 conformance based on POCL and LLVM for OpenCL-C

OpenCL offloading modes:

- Linearized Work Items on a PE (LWI)
- Single Program Multiple Data (SPMD)
- Native functions called from kernels

C/C++ POSIX Programming

Standard multicore programming model

- MPPA Linux and ClusterOS
- Standard C/C++ programming
  - GCC, GDB, LLVM, Eclipse
- POSIX threads interface
- GCC and LLVM OpenMP support

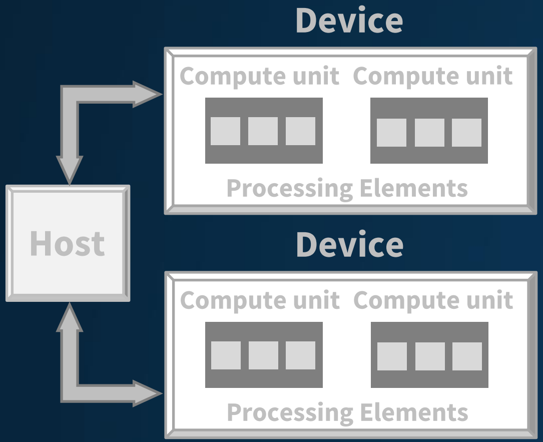
Exposed MPPA® communications

- RDMA using the MPPA Asynchronous Communication library (mppa_async)

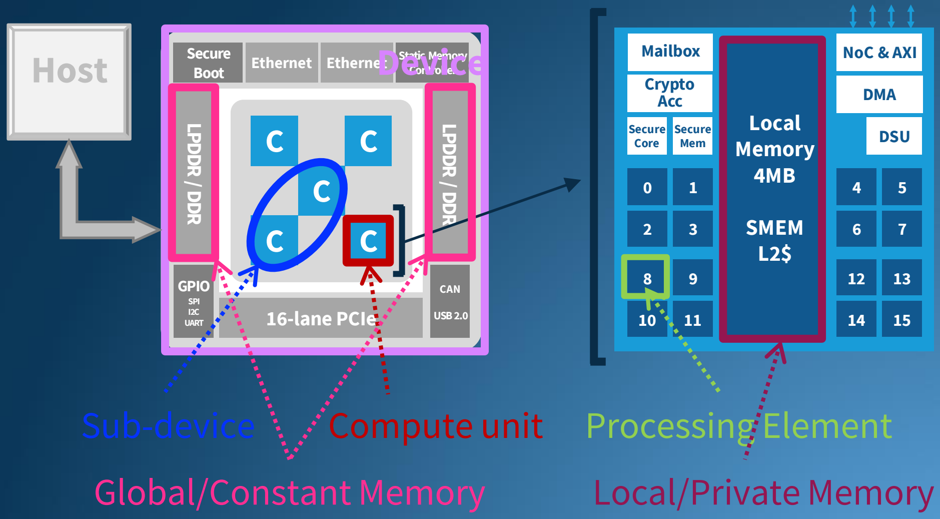
MPPA® OpenCL Compute Platform Mapping

OpenCL Compute Platform Model

Topology: Host CPU connected to one or several Device(s)
Host: CPU which runs the application under a rich OS (Linux)
Device: Compute Unit(s) sharing a Global Memory
Hierarchy: Multi-Device => Device => Sub-Device => Compute Unit(s) => Processing Elements



OpenCL 'SPMD' Mapping to MPPA® Architecture



KALRAY

MPPA® OpenCL Native Function Extension

- Call standard C/C++/OpenMP/POSIX (ClusterOS) code from OpenCL kernels
- **Generalization of TI 'OpenMP Dispatch With OpenCL' for KeyStone-II platforms**
- Used by the Kalray KaNN deep learning inference compiler
- Used by BLAS and multi-cluster libraries

```
void
my_vector_add(int *a, int *b, int *c, int n)
{
  #pragma omp parallel for
  for (int i = 0; i < n; ++i)
  {
    c[i] = a[i] + b[i];
  }
}
```

```
__attribute__((mppa_native))
void my_vector_add(__global int *a, __global int *b, __global int *c, int n);

__kernel void vector_add(__global int *a, __global int *b, __global int *c, int n) {
  my_vector_add(a, b, c, n);
}
```

MPPA Asynchronous One-Sided Operations API

generalization of OpenCL async_work_group_copy() callable from C/C++

Dense Transfers

- mppa_async_get
- mppa_async_put
- mppa_async_get_spaced
- mppa_async_put_spaced
- mppa_async_get_indexed
- mppa_async_put_indexed
- mppa_async_get_streamed
- mppa_async_put_streamed

Asynchronous Events

- mppa_async_event_wait
- mppa_async_event_test

Sparse Transfers

- mppa_async_sget_spaced
- mppa_async_sput_spaced
- mppa_async_sget_blocked2d
- mppa_async_sput_blocked2d
- mppa_async_sget_blocked3d
- mppa_async_sput_blocked3d

Remote queues

- mppa_async_enqueue
- mppa_async_dequeue
- mppa_async_dequeue_copy
- mppa_async_discard

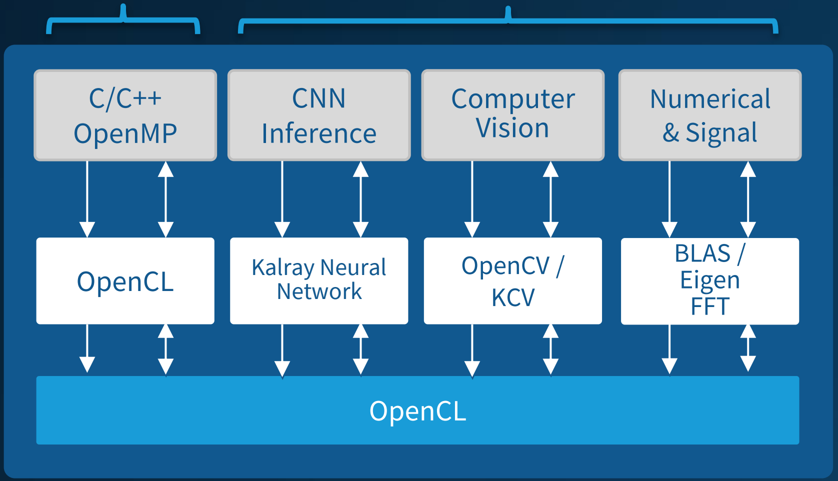
Global Synchronization

- mppa_async_quiet
- mppa_async_fence
- mppa_async_peek
- mppa_async_poke
- mppa_async_postadd
- mppa_async_fetchclear
- mppa_async_fetchadd
- mppa_async_evalcond

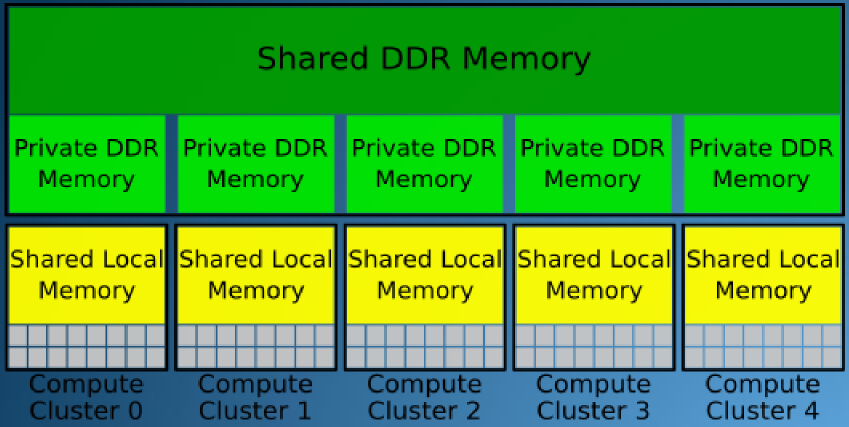
Kalray Acceleration Framework (KAF™)

A integrated way to program a manycore accelerator
based on OpenCL Sub-Devices and Native Functions extension

Direct programming API / Lib programming



Memory Model for Native Functions



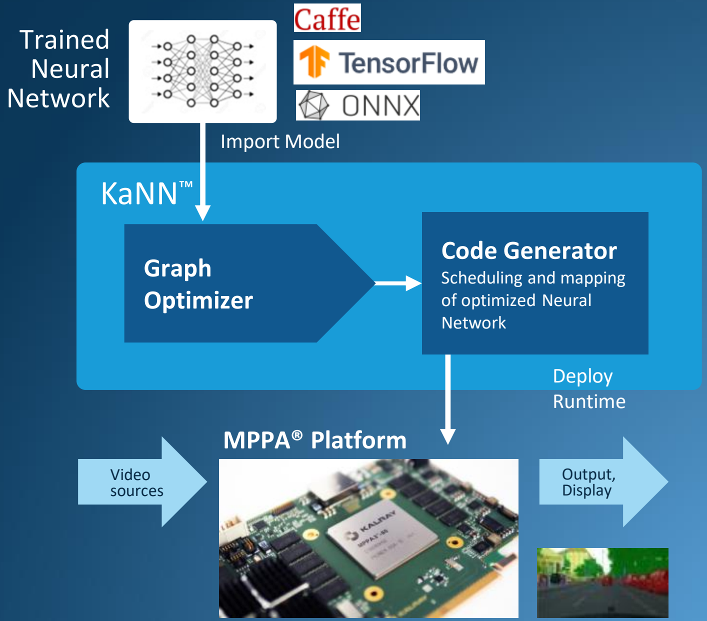
KALRAY

KaNN™ the Kalray Neural Network Compiler

From trained models in standard CNN frameworks
To inference code generation, setup & concurrent CNN inferences

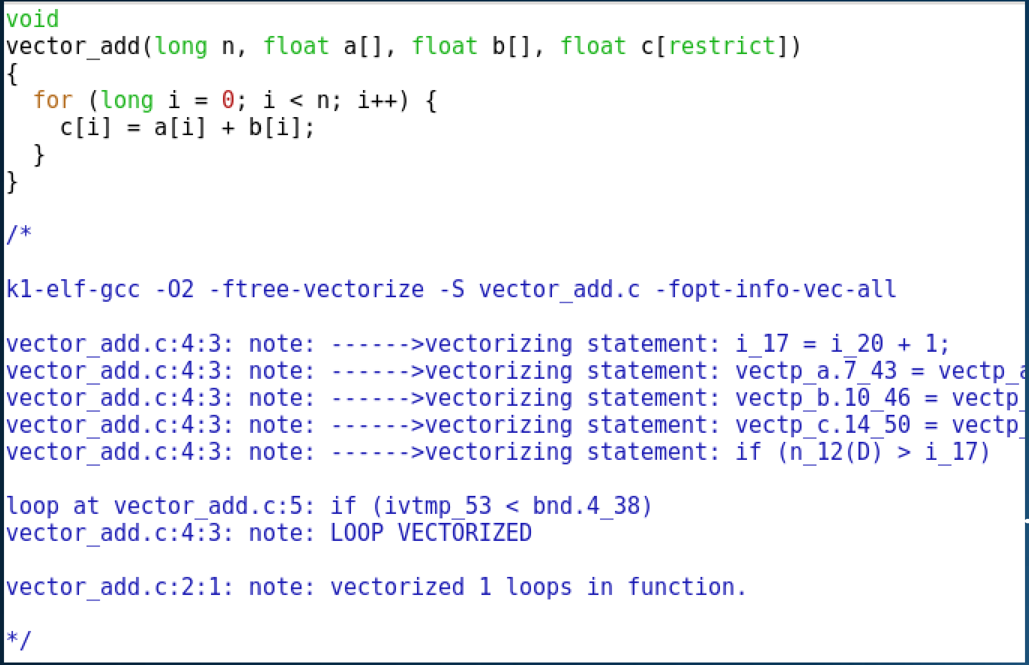
Graph Optimizer
- Dummy quantization
- Copy & concatenation elimination
- Scale layers folding
- ReLU layer merging
- Convolution padding
- Fusion of element-wise layers

Code Generator
- Activation memory allocation
- Mapping to libtensor kernels
- Command buffer generation
- Parameter buffer allocation
- Static Profiling

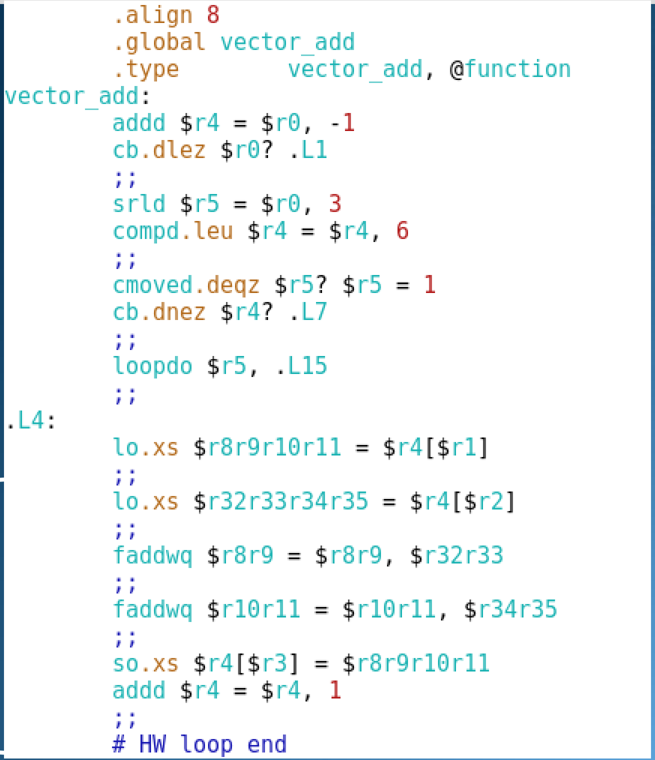


C/C++ Compiler Support of Kalray VLIW Core

Generic optimization apply (here GCC autovectorization)



```
void
vector_add(long n, float a[], float b[], float c[restrict])
{
  for (long i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
  }
}

/*

k1-elf-gcc -O2 -ftree-vectorize -S vector_add.c -fopt-info-vec-all

vector_add.c:4:3: note: ------>vectorizing statement: i_17 = i_20 + 1;
vector_add.c:4:3: note: ------>vectorizing statement: vectp_a.7_43 = vectp_a
vector_add.c:4:3: note: ------>vectorizing statement: vectp_b.10_46 = vectp_
vector_add.c:4:3: note: ------>vectorizing statement: vectp_c.14_50 = vectp_
vector_add.c:4:3: note: ------>vectorizing statement: if (n_12(D) > i_17)

loop at vector_add.c:5: if (ivtmp_53 < bnd.4_38)
vector_add.c:4:3: note: LOOP VECTORIZED

vector_add.c:2:1: note: vectorized 1 loops in function.

*/
```

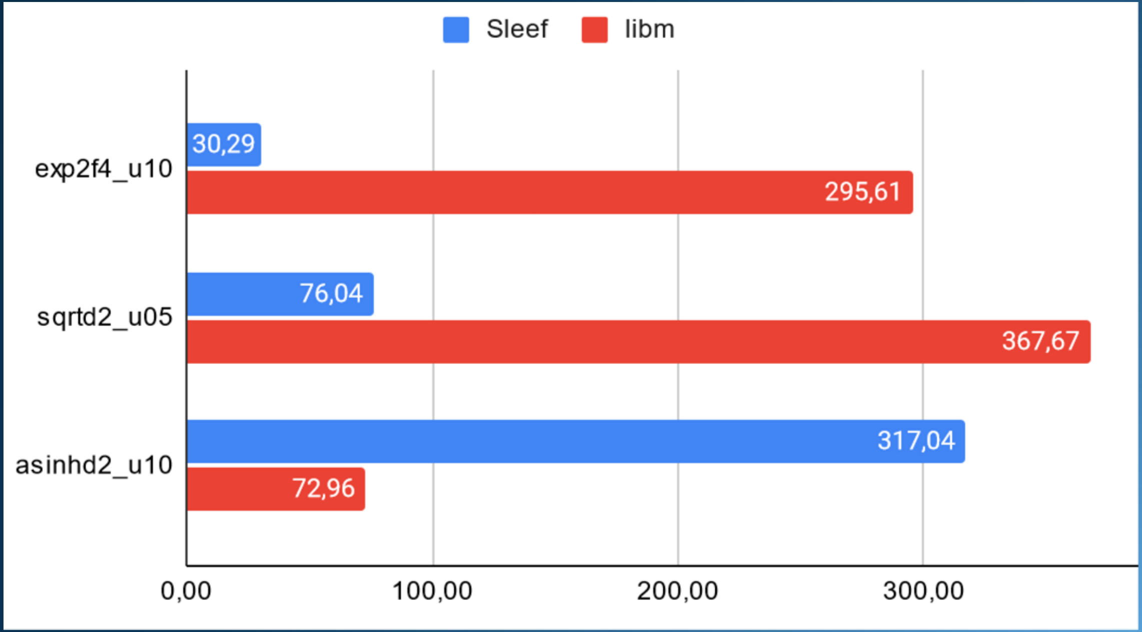


```
        .align 8
        .global vector_add
        .type       vector_add, @function
vector_add:
        addd $r4 = $r0, -1
        cb.dlez $r0? .L1
        ;;
        srld $r5 = $r0, 3
        compd.leu $r4 = $r4, 6
        ;;
        cmoved.deqz $r5? $r5 = 1
        cb.dnez $r4? .L7
        ;;
        loopdo $r5, .L15
        ;;
.L4:
        lo.xs $r8r9r10r11 = $r4[$r1]
        ;;
        lo.xs $r32r33r34r35 = $r4[$r2]
        ;;
        faddwq $r8r9 = $r8r9, $r32r33
        ;;
        faddwq $r10r11 = $r10r11, $r34r35
        ;;
        so.xs $r4[$r3] = $r8r9r10r11
        addd $r4 = $r4, 1
        ;;
        # HW loop end
```

SLEEF (SIMD Library for Evaluating Elementary Functions)

- Open source distributed under the Boost Software License
- Implements manually vectorized versions of all C99 real floating point math functions, precise to 1ulp over the whole input range
- Uses approximation polynomials with higher degree than a scalar libm to limit the variability of argument range reduction
- Polynomials evaluated with Estrin's scheme instead of Horner's scheme to expose instruction parallelism

Cycle count per element (1st SLEEF release)



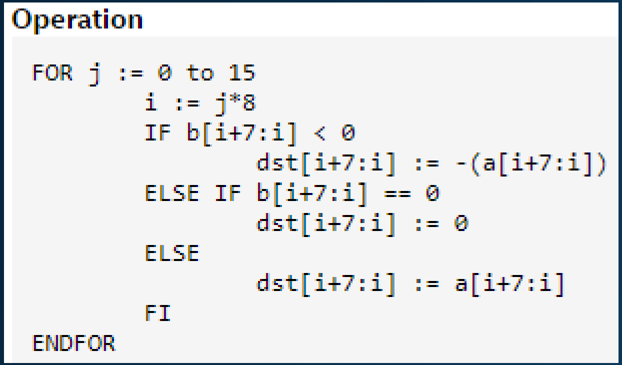
SIMDe Emulation of X86 Builtins (1)

Synopsis

__m128i _mm_sign_epi8 (__m128i a, __m128i b)

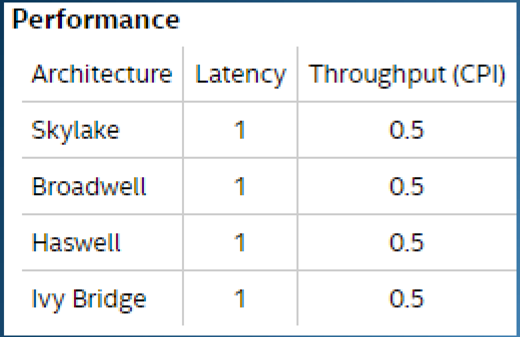
Description

Negate packed 8-bit integers in a when the corresponding signed 8-bit integer in b is negative, and store the results in dst. Element in dst are zeroed out when the corresponding element in b is zero.



Operation

```
FOR j := 0 to 15
        i := j*8
        IF b[i+7:i] < 0
                dst[i+7:i] := -(a[i+7:i])
        ELSE IF b[i+7:i] == 0
                dst[i+7:i] := 0
        ELSE
                dst[i+7:i] := a[i+7:i]
        FI
ENDFOR
```

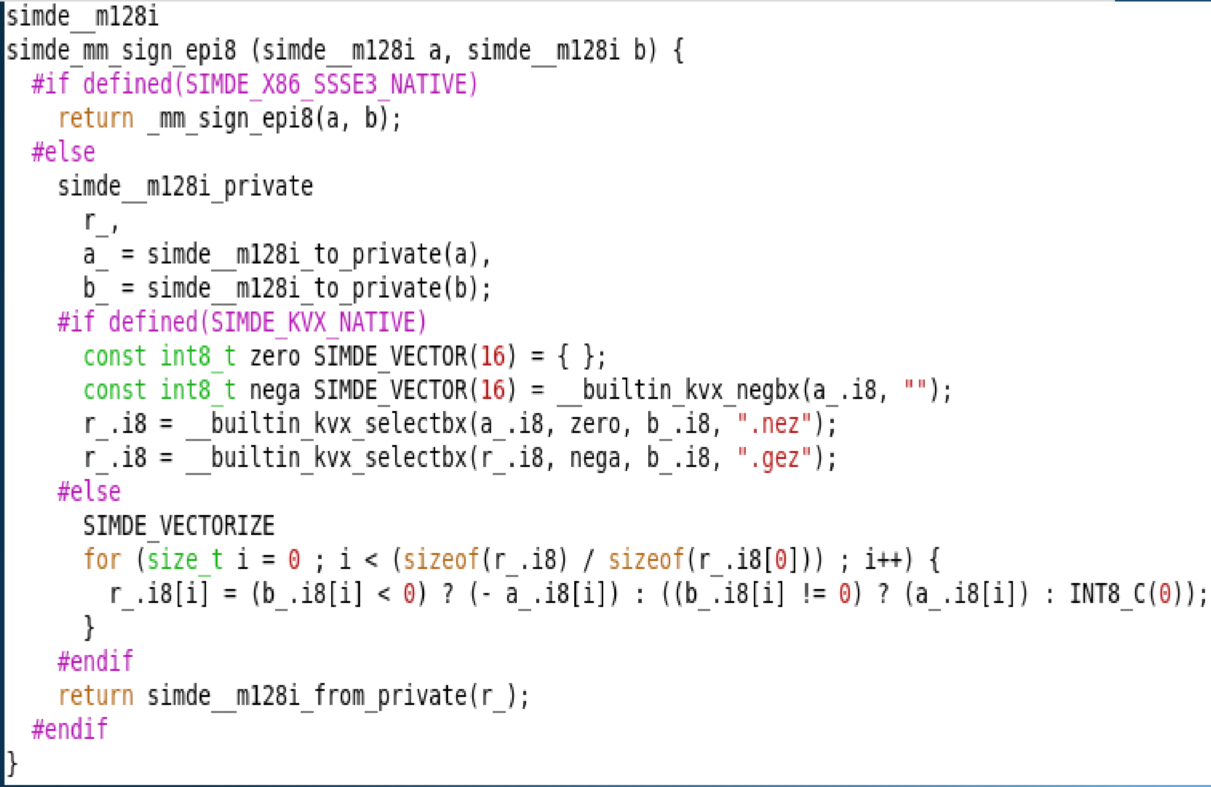


Performance

Architecture	Latency	Throughput (CPI)
Skylake	1	0.5
Broadwell	1	0.5
Haswell	1	0.5
Ivy Bridge	1	0.5

SIMDe Emulation of X86 Builtins (2)

- SIMDe translates all the x86 builtin functions into native call on x86 (SIMDE_X86_SSSE3_NATIVE) and plain C code on other architectures (SIMDE_VECTORIZE)
- Kalray extended SIMDe to provide an optimized translation on KVX using the GCC/LLVM kvx builtin functions (SIMDE_KVX_NATIVE)

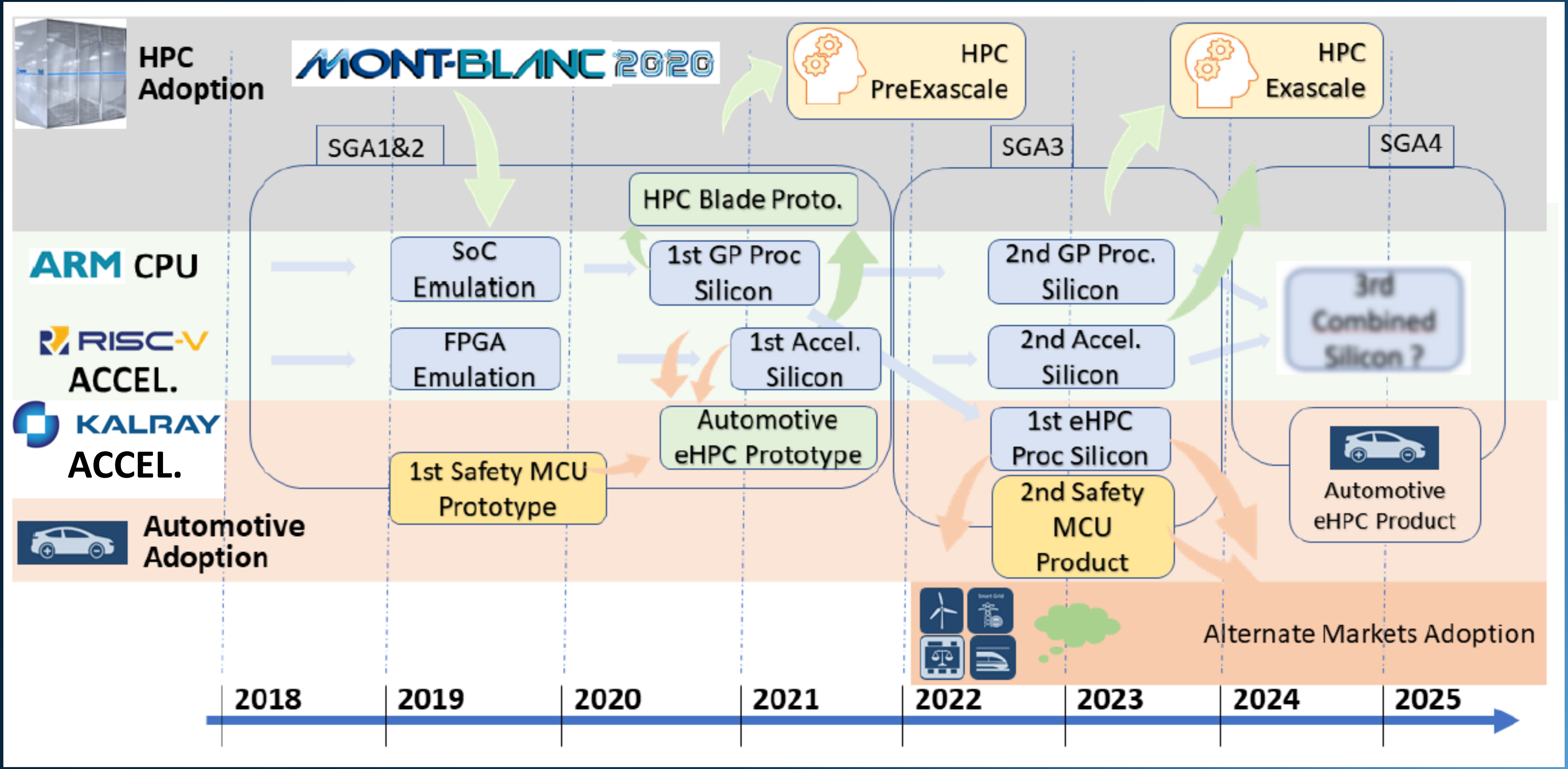


```c
simde__m128i
simde_mm_sign_epi8 (simde__m128i a, simde__m128i b) {
  #if defined(SIMDE_X86_SSSE3_NATIVE)
    return _mm_sign_epi8(a, b);
  #else
    simde__m128i_private
      r_,
      a_ = simde__m128i_to_private(a),
      b_ = simde__m128i_to_private(b);
    #if defined(SIMDE_KVX_NATIVE)
      const int8_t zero SIMDE_VECTOR(16) = { };
      const int8_t nega SIMDE_VECTOR(16) = __builtin_kvx_negbx(a_.i8, "");
      r_.i8 = __builtin_kvx_selectbx(a_.i8, zero, b_.i8, ".nez");
      r_.i8 = __builtin_kvx_selectbx(r_.i8, nega, b_.i8, ".gez");
    #else
      SIMDE_VECTORIZE
      for (size_t i = 0 ; i < (sizeof(r_.i8) / sizeof(r_.i8[0])) ; i++) {
        r_.i8[i] = (b_.i8[i] < 0) ? (- a_.i8[i]) : ((b_.i8[i] != 0) ? (a_.i8[i]) : INT8_C(0));
      }
    #endif
    return simde__m128i_from_private(r_);
  #endif
}
```

Outline

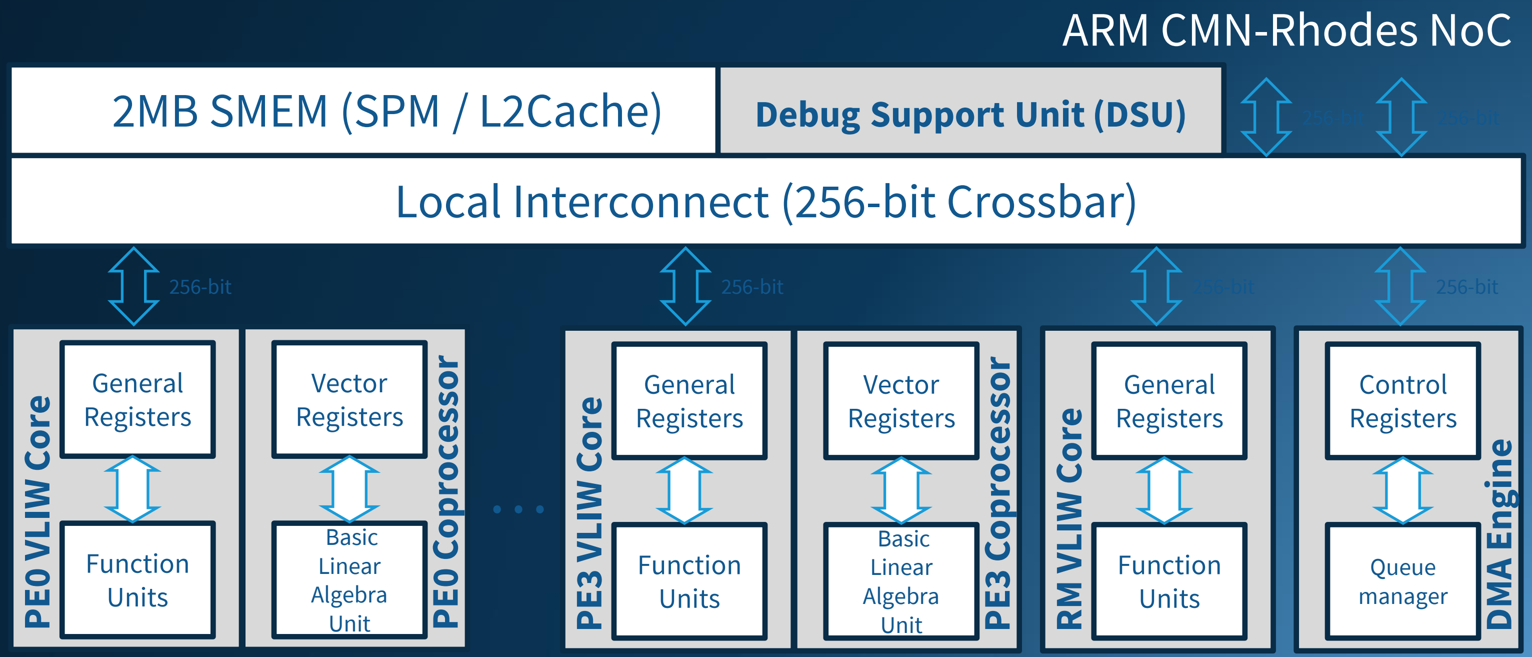
1. Manycore Accelerators
2. Edge Computing
3. MPPA3 Processor and IP
4. Accelerator Offloading
5. Outlook & Conclusions

Mont-Blanc 2020 and EPI Projects



KALRAY

MPPA Accelerator Tile Delivered to EPI Project (TSMC 6nm)



KVX Accelerator Tile for the EPI SGA-2

RISC-V Cores

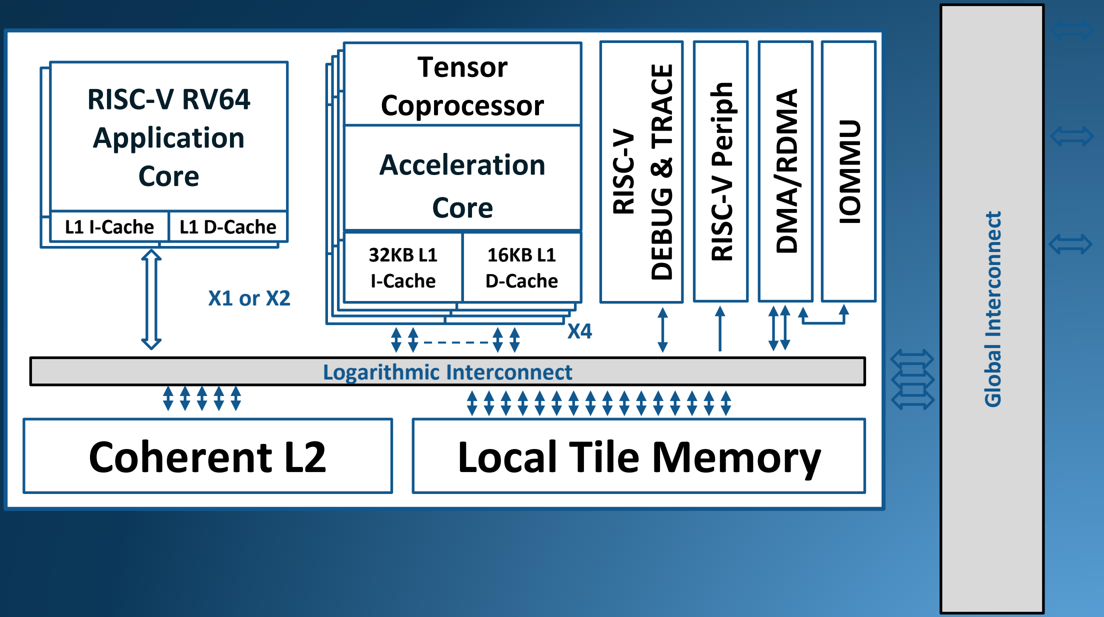
- A general-purpose 64-bit application core is provided for running RDMA, MPI and storage software stacks

KVX Cores + coprocessors

- 4 Kalray VLIW cores, each with a dedicated tensor coprocessor, provide the HPC/Edge floating-point performance 128 DP FLOP/cycle

Local multi-banked memory

- Supports the required local load/store bandwidth (32 bytes per KVX core)
- Data move by DMA/RDMA engine



Summary and Conclusions

Co-designing for accelerated edge computing

- CPU-based manycore processor architecture for scalability, time-critical computing and multicore programmability

VLIW + tensor coprocessor PE architecture

- It is possible to design a compiler-friendly VLIW architecture for intensive computing and machine learning
- Avoids the complexities of high-end superscalar implementations, but requires advanced software pipelining

Manycore accelerator architecture challenges

- Among the SMP, I/O, Accelerator cache coherencies, I/O coherency is the most critical when porting software
- Global interconnect with Ethernet termination / RX load balancing / TX flow-control
- Global interconnect with cache coherence and memory access scheduling

Acknowledgement

- Mont-Blanc 2020 and EPI SGA1 Projects
- GCC, GDB, LLVM, QEMU, SoftFloat, SLEEF, SIMDe, NewLib, POCL, Eclipse, FreeRTOS, Linux kernel, DPDK, SPDK, …

KALRAY

Thank You

KALRAY S.A.
Corporate Headquarters
180, avenue de l'Europe
38 330 Montbonnot, France
Phone: +33 (0)4 76 18 90 71
contact@kalrayinc.com

CERTIFIED ORGANIZATION ISO 9001
CERTIFICATION IN PROGRESS ISO 26262

KALRAY INC.
America Regional Headquarters
4962 El Camino Real
Los Altos, CA - USA
Phone: +1 (650) 469 3729
contact@kalrayinc.com

KALRAY JAPAN - KK
Represented by MACNICA Inc. Strategic Innovation Group
Macnica Building, No.1, 1-6-3 Shin-Yokohama
Kouhoku-ku, Yokohama 222-8561, Japan
Phone: +81-45-470-9870

KALRAY S.A.
Sophia-Antipolis
1047 allée Pierre Ziller
Business Pôle – Bâtiment B, Entrée A
06560 Sophia-Antipolis, France
Phone: + 33(0) 4 76 18 09 18

www.kalrayinc.com