MPPA® AccessCore 1.4.2 Getting Started Guide

Supercomputing on a chip™

April 2015

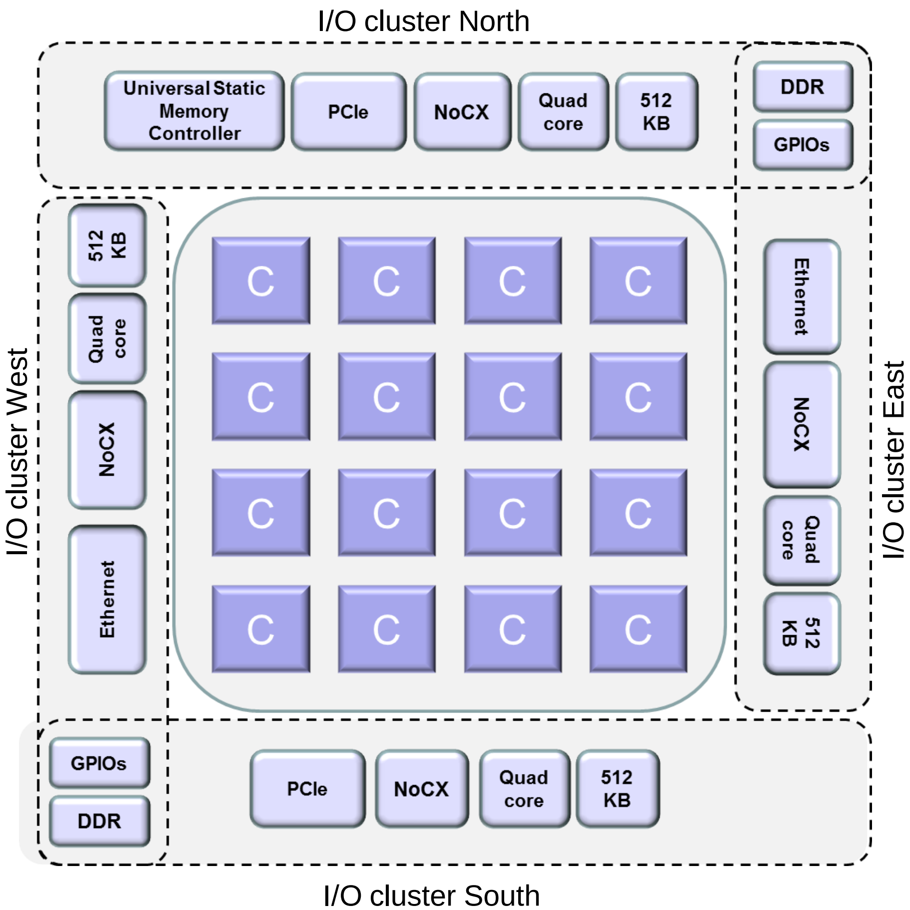
Contacts information

- Europe, Middle-East, Africa (EMEA) and Japan Sales
  - Stéphane Cordova <scordova@kalray.eu>
  - Jean-Pierre Demange <jpdemange@kalray.eu>
- Asia (exc. Japan, Pacific and Americas
  - Christopher Piercy <cpiercy@kalrayinc.com>

- First level support
  - support@kalray.eu

Contents

MPPA®-256 Andey Processor



- 4 I/O clusters with Quad cores
- 16 compute clusters (marked C)
  - 16 cores 2 MB of shared memory each
  - SMP type of architecture
- 2D torus wrap-around network on chip (NoC)
  - 3.2 GB/S full duplex between each cluster and its 4 neighbors
  - NoC extension between MPPA
- DDR3 Memory interfaces
- 2 x 8 lanes PCIe Gen3 interface
- 1G/10G Ethernet interfaces
- Universal Static Memory Controller (NAND/NOR/SRAM)
- GPIO / SPI / I2C / UART / PWM

MPPA® ACCESSCORE 1.4
Kalray Software Development Kit

Standard C/C++ Programming Environment

Simulators & Profilers, Debuggers & System Trace

Operating Systems & Device Drivers

Dataflow Programming
DSP Style

C/C++ - POSIX-Level Programming
CPU Style

Contents

MPPA® ACCESSCORE documentation

- Where to find the documentation?

- General presentation
  - /usr/share/AccessCore/docs

- Reference documents
  - /usr/share/AccessCore/docs/Book
    - Dataflow.pdf
    - PosixProgramming.pdf
    - SimulationTraceDebug.pdf

MPPA® ACCESSCORE release tools

- Dataflow programing model
  - /usr/bin/sc-*
- Posix programing model
  - /usr/local/k1tools/bin
- Eclipse
  - k1-eclipse
  - Dataflow plugin installed
  - Posix plugin installed

MPPA® ACCESSCORE Examples

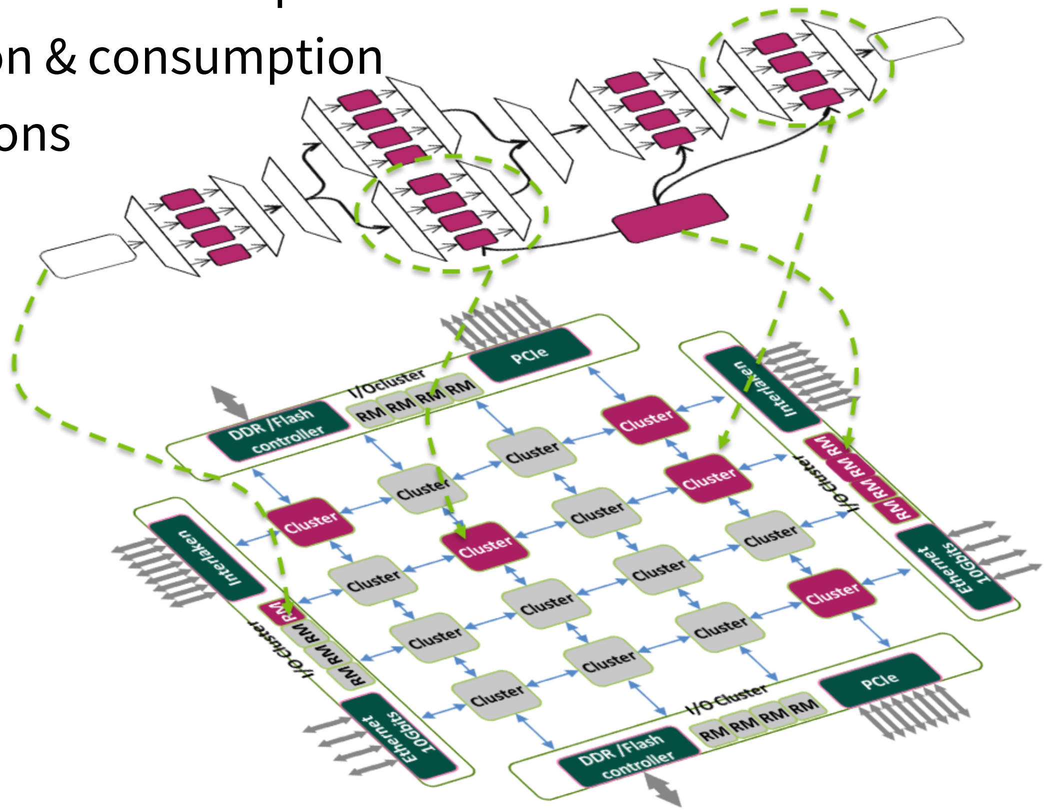
- Where to find application examples with source code?
- /usr/share/AccessCore/Apps
  - Dataflow examples :
    - H264 dataflow code installed into eclipse
    - ViolaJones
    - ScaaL
    - AES
    - Matrix multiply
  - Posix Level examples :
    - AES
    - MPPA IPC (Sobel, Double Buffering, …)

Contents

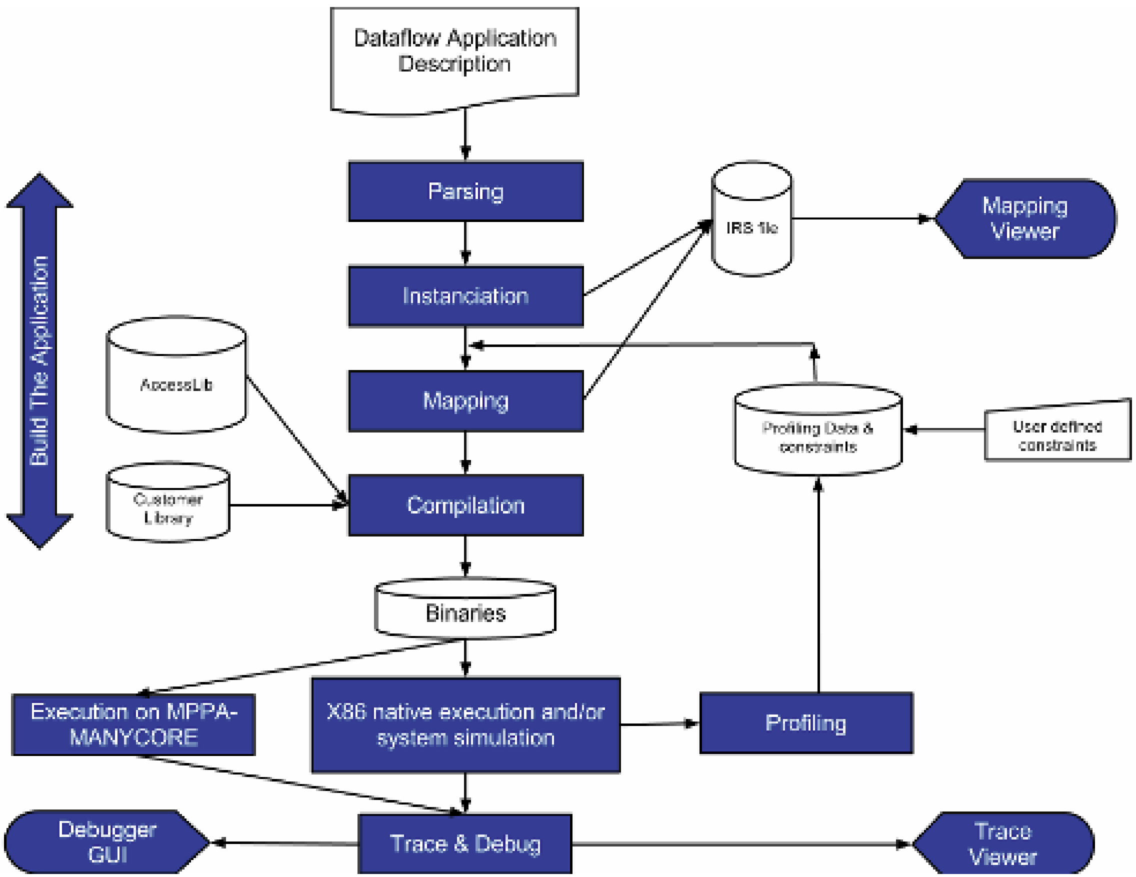
Dataflow Programming

- Computation blocks and communication graph written in plain C
- Supports “task parallelism” and “data parallelism”
- Cyclostatic data production & consumption
- Dynamic dataflow extensions

Automatic mapping on MPPA®
memory, computing,
& communication resources

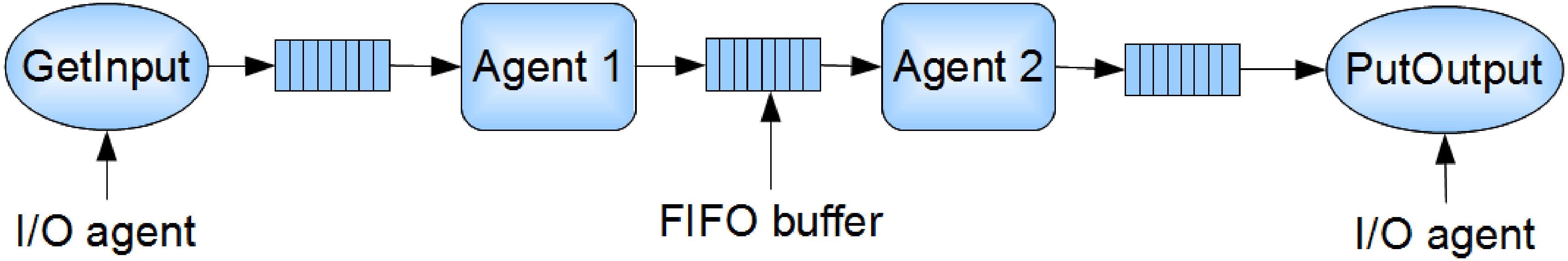


Dataflow Development flow



The Dataflow Model

- Data is explicit, illustrated physically as a FIFO buffer
- Operations are within processing elements with inputs and outputs (« agents »)
- Agents run when inputs become valid
- Execution order determined by movement of data through agents



The Dataflow Model

- No synchronization between processing elements
- Computing agents are scheduled once their dependencies are satisfied
- Communication exclusively via isolated FIFO channels between agents
- Model is inherently parallel
  - Tasks are executed once input arrives so several processing elements can be ready to execute at the same time

∑C Agents : Example

<u>Agent Inverter's function:</u>
- Consume an unsigned char
- Process the input unsigned char
- Produce an inverted unsigned char

agent keyword followed by the name of the agent

Interface section for input/ output channels

Exchange functions to directly manipulate input/ output channels

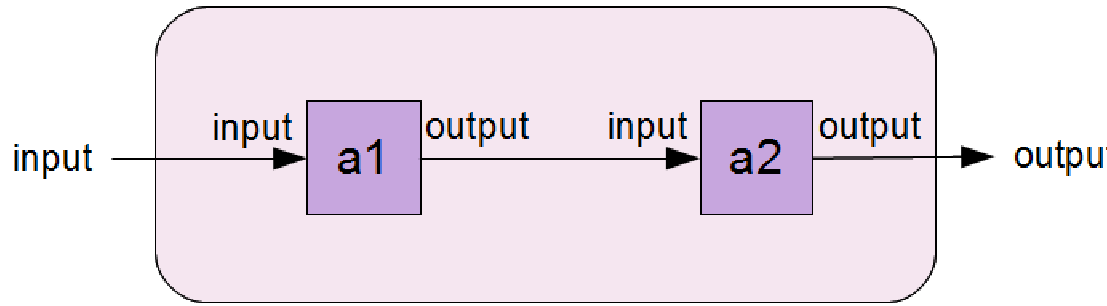
Standard C code within the agent

start function is an infinite loop

```
agent Inverter()
{
    interface
    {
        in<unsigned char> input; /*< input byte stream */
        out<unsigned char> output; /*< output byte stream */

        spec{input; output};
    }

    void invert (void) exchange (input pel_in, output pel_out)
    {
      pel_out = 255 - pel_in;
    }

    void start ()
    {
      invert();
    }
}
```

Connecting Agents

- Agents are connected within the « map » section of « subgraphs »




```
subgraph Subgraph1 ()
{
    interface
    {
        in<unsigned char> input;
        out<unsigned char> output;

        spec{input; output};
    }

    map {
        int N = 1024;
        agent a1 = new Agent1();
        agent a2 = new Agent2(N);
        connect(input, a1.input);
        connect(a1.output, a2.input);
        connect(a2.output, output);
    }
}
```

Agents are instanciated and connected in the map section

Agents are instanciated via the « new » keyword

A parameter « N » added as an argument to the agent

Agent interfaces are connected using « connect »

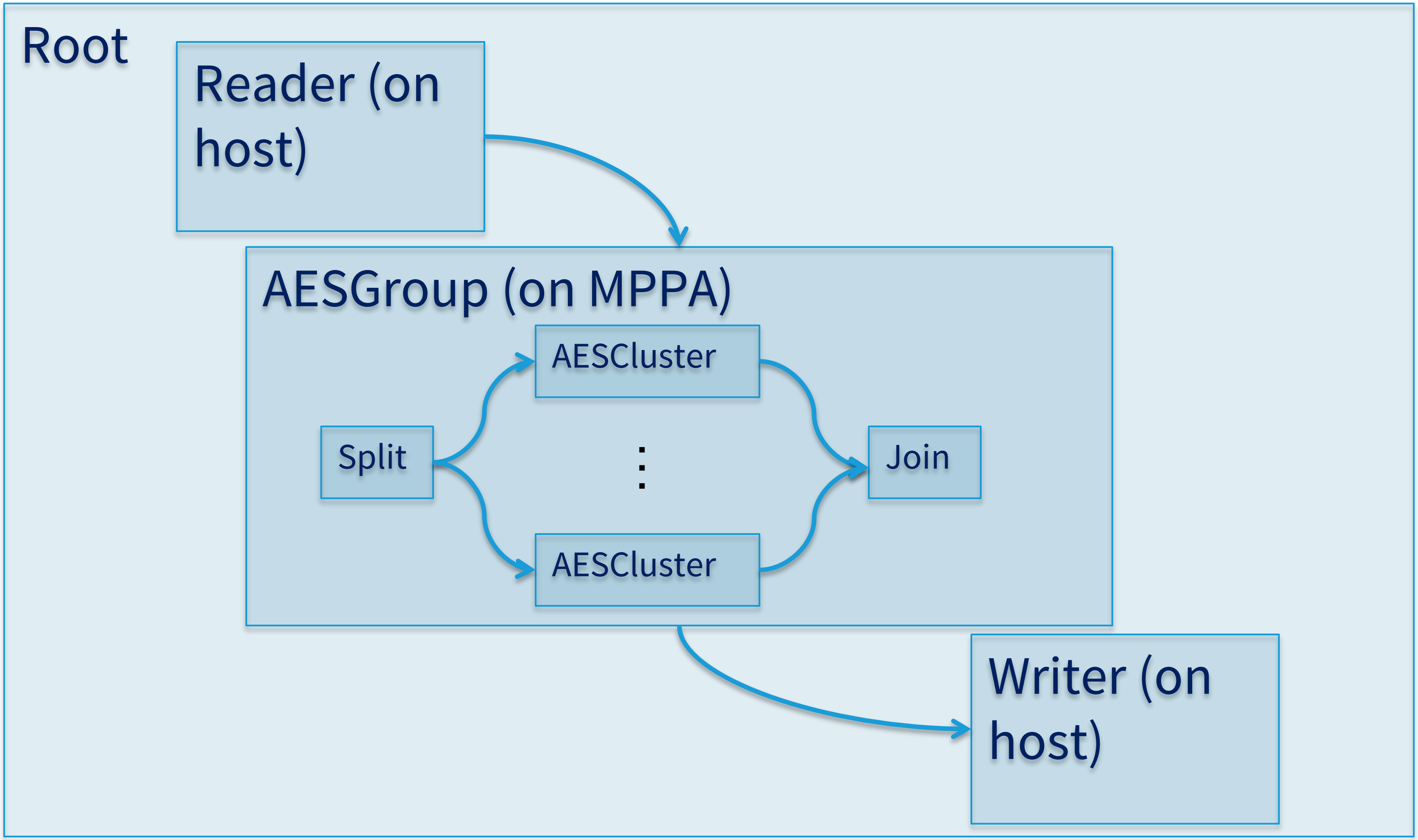
Example Application from AccesscoreApps : AES

- AES encryption
- Found in /usr/share/AccessCore/Apps/AES
- Root.sc: like the « main » process
  - Includes the « root » subgraph (entry point of a sigmaC program)
- IOAgents.sc: file input and output
- Group.sc: IO and cluster level instantiations
- Cluster.sc: group of agents to be placed on one cluster
- Encoder.sc: AES encoder agent definition

Import the Application into k1-eclipse

- Open the k1-eclipse tool
- Select File->Import
- Import project (make sure you check the Copy to workspace box)
  - This will import the project's source files as well as the build and run configurations

AES Block Diagram



AES : Root.sc

Entry point of a sigmaC application: the "root" subgraph

```
subgraph root() {
        map
        {
                const unsigned iterationSize = BUFFER_SIZE * N_PE * N_CLUSTERS;

                const unsigned groupSize = iterationSize * GRANULARITY;

                agent ioIn = new myReader(groupSize);

                agent ioOut = new myWriter(groupSize);

                connect(ioIn.qty_sync, ioOut.qty_sync);

                agent group = new AESGroup(N_PE, N_CLUSTERS, BUFFER_SIZE, 0,
                GRANULARITY);
                connect(ioIn.output, group.input);
                connect(group.output, ioOut.input);
        }
}
```

the amount of data encoded by one iteration of the AESEncoder agents

the amount of data encoded by one iteration of the AESGroup

read the data to be encoded from an input file

write the encoded data to an output file

The myReader agent might produce dummy data to reach the requested size. A control channel is used to transmit the amount of real data sent for encoding

Instantiate an AESGroup subgraph to map to the MPPA and connect it to the reader and writer agents

AES : Group.sc

```
subgraph AESGroup (unsigned nProc, unsigned nClus, unsigned bufferSize, unsigned group_id, unsigned grenularity) {
	interface {
		in<unsigned char>input;
		out<unsigned char>output;
		spec{};
	}
	map {
		agent split = new Split<unsigned char>(nClus, bufferSize * nProc);

		SigmaC_agent_setUnitType(split, "k1-I/O");

		agent join = new FastJoin<unsigned char>(nClus, bufferSize * nProc * grenularity);

		SigmaC_agent_setUnitType(join, "k1-I/O");

		connect(input, split.input);
		connect(join.output, output);

		unsigned i;
		for( i = 0; i < nClus; ++i){
			agent cluster = new AESCluster(nProc, bufferSize, group_id * nClus + i);
			connect(split.output[i], cluster.input);
			connect(cluster.output, join.input[i]);
		}
	}
}
```

Split agent to dispatch to the AESCluster subgraphs

The agent must be in an I/O cluster

Join agent to merge the data from the AESCluster subgraphs. The agent must be in an I/O cluster

Connect the dispatch agents to the subgraph ports

Instantiate the AESCluster subgraphs and connect them

AES : Cluster.sc

```
subgraph AESCluster (unsigned nProc, unsigned bufferSize, int clus_id){
	interface {
		in<unsigned char>input;
		out<unsigned char>output;
		spec{};
	}
	map{
		unsigned i;

		unsigned int affGroup = SigmaC_createAffinityGroup(1000000000);

		agent split = new FastSplit<unsigned char>(nProc, bufferSize);
		SigmaC_agent_addToAffinityGroup(affGroup, split);

		agent join = new FastJoin<unsigned char>(nProc, bufferSize);
		SigmaC_agent_addToAffinityGroup(affGroup, join);

		connect(input, split.input);
		connect(join.output, output);

		for(i = 0; i < nProc; ++i){
			agent crypt = new AESEncoder(bufferSize, clus_id * nProc + i);
			SigmaC_agent_addToAffinityGroup(affGroup, crypt);
			connect(split.output[i], crypt.input);
			connect(crypt.output, join.input[i]);
		}
	}
}
```

Create an affinity group to encourage the placer to place all the cluster subgraph in a single cluster

Split agent to dispatch to the AESEncoder agents

Join agent to merge the data from the AESCEncoder agents

Connect the dispatch agents to the subgraph ports

Instantiate the AESEncoder agents and connect them

AES : Encoder.sc

```
agent AESEncoder(unsigned bufferSize, int id){
        interface {
                in<unsigned char> input;
                out<unsigned char> output;
                spec{input[bufferSize]; output[bufferSize]};
        }

        /* Encryption context structure for OpenSSL */
        struct ctr_state state;

        map {
                SigmaC_agent_resource_setStack(SigmaC_agent_self(), 2000);
        }

        init {
                /* Encoding key */
                unsigned char key[16] = "0123456789abcdef";
                /* Initial vector */
                unsigned char iv[8] = "01234567";

                /* Initialize context for libcrypto */
                state.num = 0;
                state.id = id;
                state.status = 0;

                if (AES_set_encrypt_key(key, 128, &state.key) != 0){
                        __SCIO_printf("Failed to initialize AES context.\n");
                        exit(EXIT_FAILURE);
                }

                memset(state.ivec, 0, 16);
                memset(state.ecount, 0, 16);
                memcpy(state.ivec, iv, 8);
        }

        void start() exchange (input in[bufferSize], output out[bufferSize]){
                /* Encode the data using libcrypto */
                AES_ctr128_encrypt(in, out, bufferSize, &state.key, state.ivec, state.ecount, &state.num);

        }
}
```

Main function, retrieve bufferSize data, encode them and send them

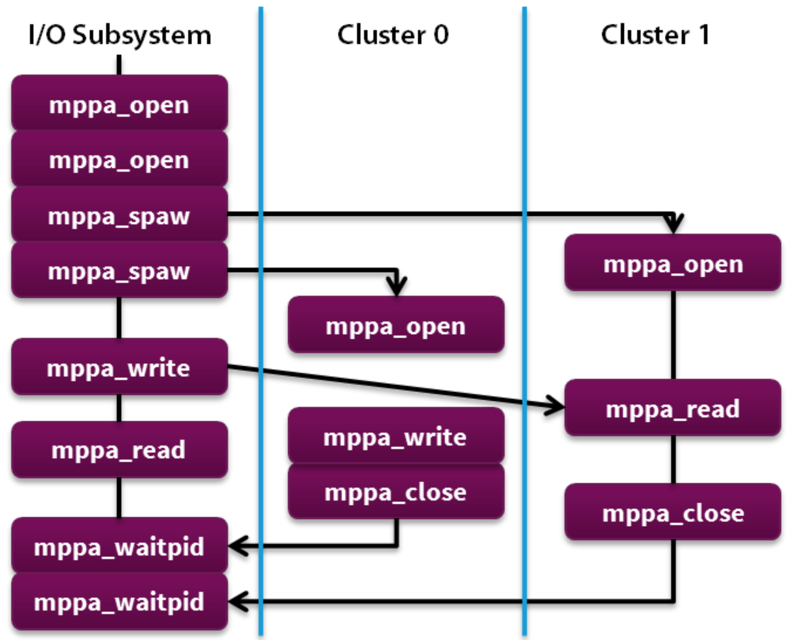
How To Build, View and Run

- Build the application
  - Right-click on the project
  - Select Build Project
- Open the SigmaC Explorer viewer
  - Right-click on the project
  - Select Open IRS -> Debug
  - Here you have all the different views (Dataflow, Mapping and Scheduling, etc)
- Choose a run configuration to run the application
  - Right-click on the project
  - Select Run as -> Run Configurations
  - Choose a pre-configured run configuration from the list
    - either simulator (SigmaC Simulation configuration) or HW run (SigmaC Application on target)

Contents

POSIX-Level Programming

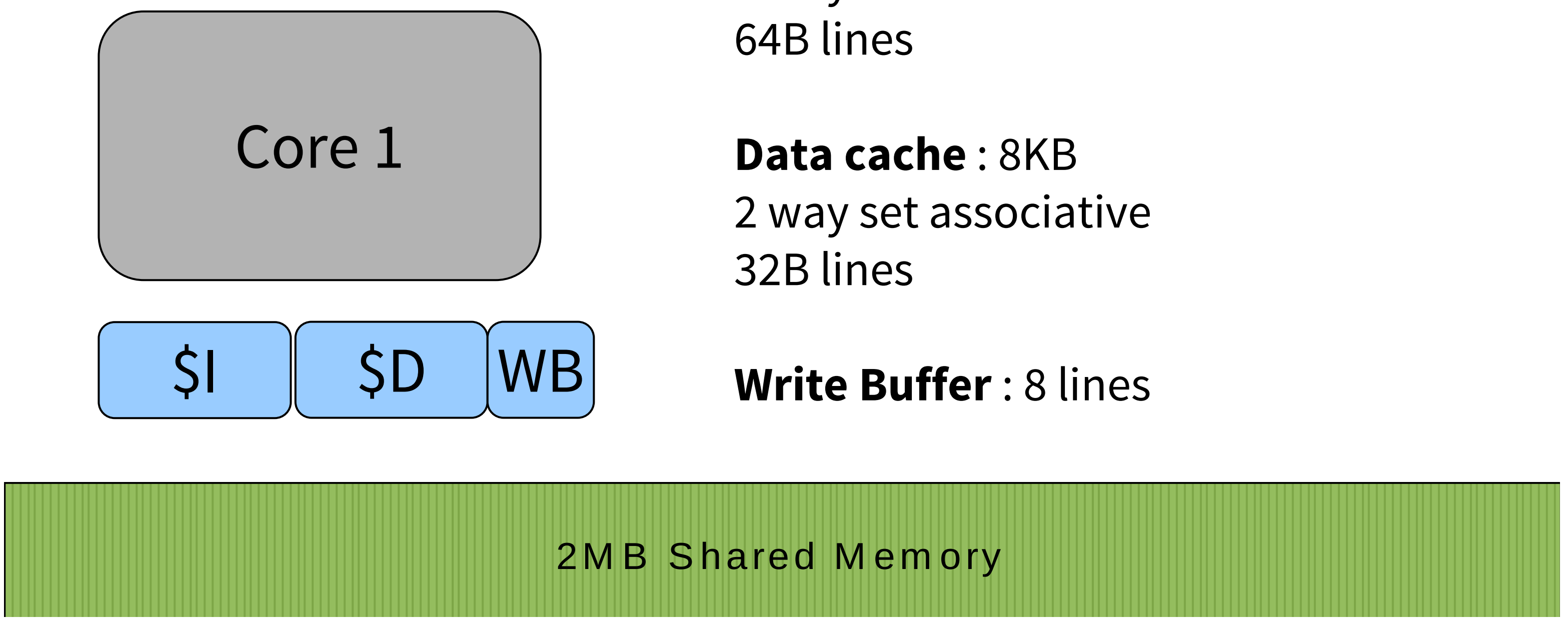
- POSIX-like process management
  - Spawn 16 processes from the I/O subsystem
  - Process execution on the 16 clusters start with main(argc, argv) and environment
- Inter Process Communication (IPC)
  - POSIX file descriptor operations on ‘NoC Connectors’
  - Extension to the PCIe interface with the ‘PCI Connectors’
  - Rich communication and synchronization
- Multi-threading inside clusters
  - Standard GCC/G++ OpenMP support
    - #pragma for thread-level parallelism
    - Compiler automatically creates threads
  - POSIX threads interface
    - Explicit thread-level parallelism



The compute clusters

- Inside a computer cluster:
  - There is no cache coherence
    - This can be managed manually using WBflush, Dcache invalidation, ..
      - Docs: /usr/local/k1tools/Manuals/GCCBuiltins/GCCBuiltins.pdf
    - We recommend to use the POSIX Mutex
      - Docs: /usr/local/k1tools/doc/Specifications/NodeOS/NodeOS.pdf
- The compute cluster boot
  - The BSP is executed on the System Core processor
  - The NodeOS operating system starts
  - NodeOS spawns the *main* function to the PC0 core
  - From the *main* function, the others 15 PC cores can be used using POSIX pthreads or OpenMP
  - A PC core cannot be shared by 2 POSIX pthreads
    - Docs: /usr/local/k1tools/doc/Specifications/NodeOS/NodeOS.pdf

No Cache Coherence (inside a compute cluster)

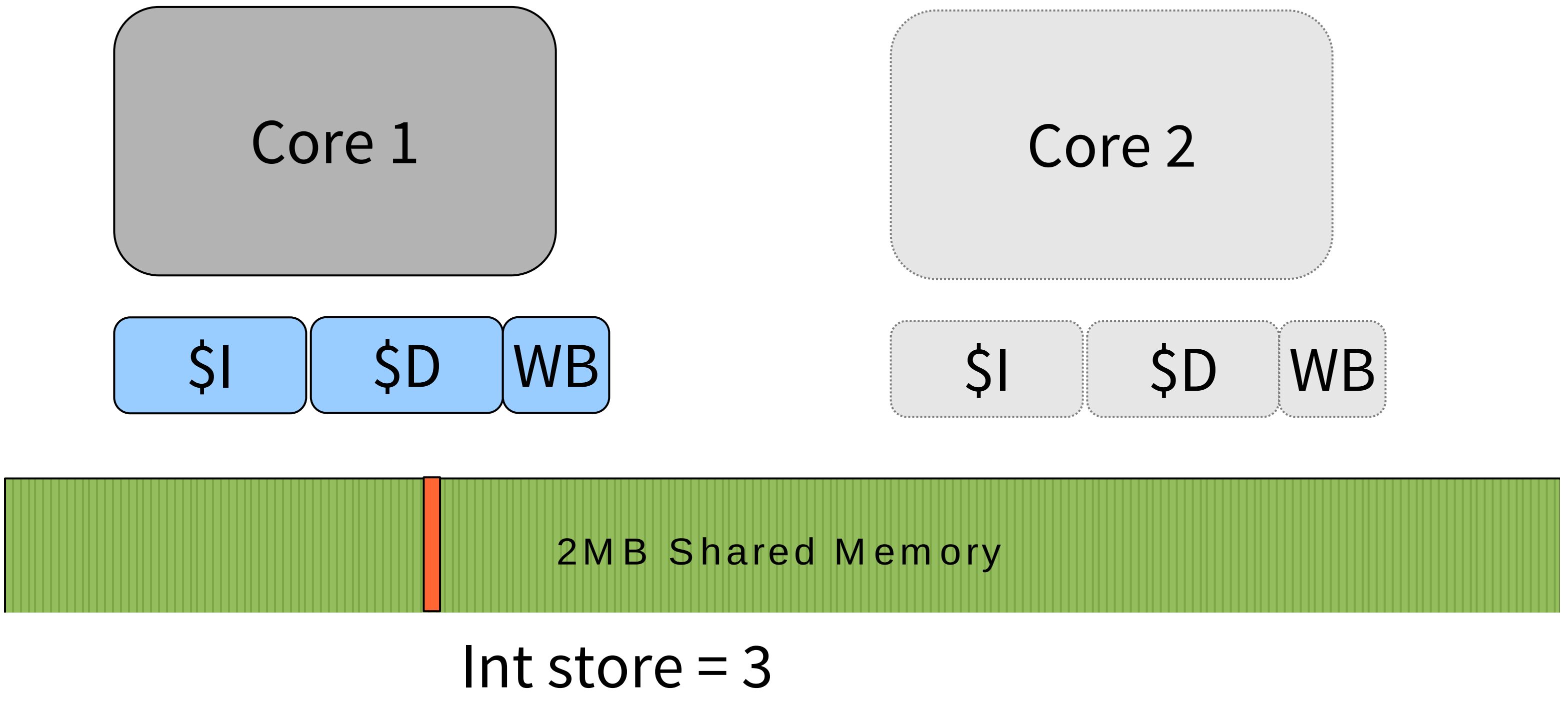


Instruction cache : 8KB
2 way set associative
64B lines

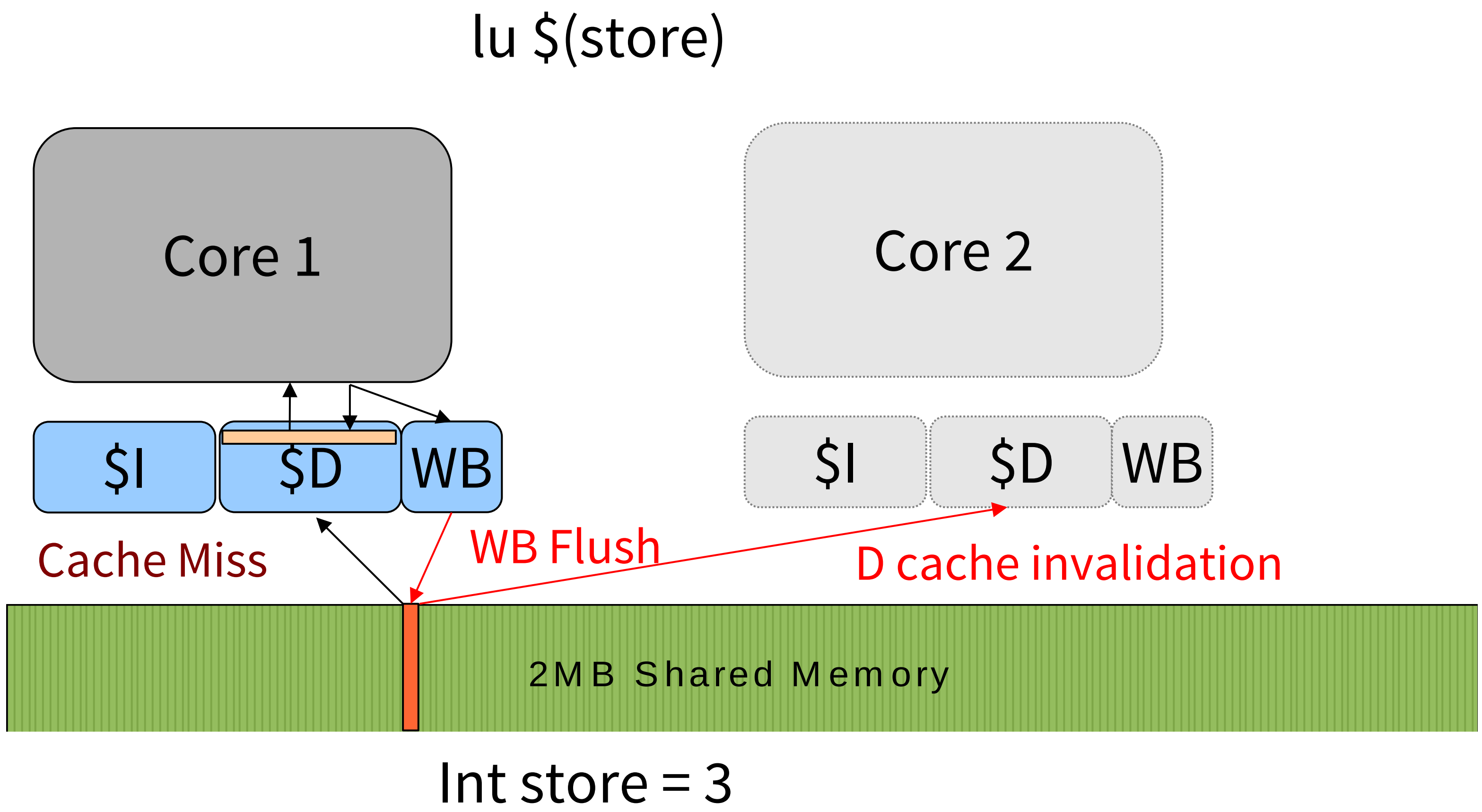
Data cache : 8KB
2 way set associative
32B lines

Write Buffer : 8 lines

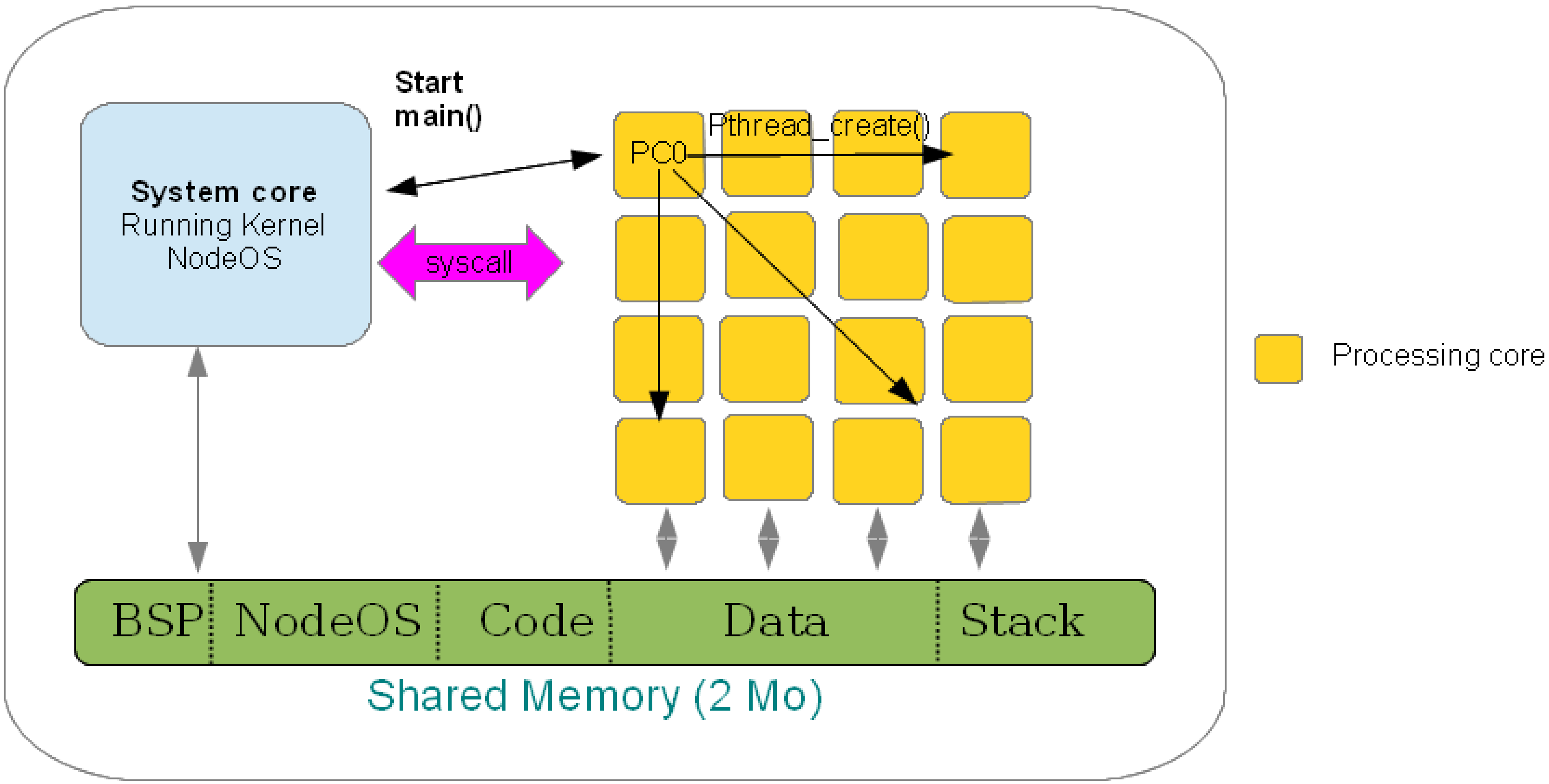
No Cache Coherence (inside a compute cluster)



No Cache Coherence (inside a compute cluster)



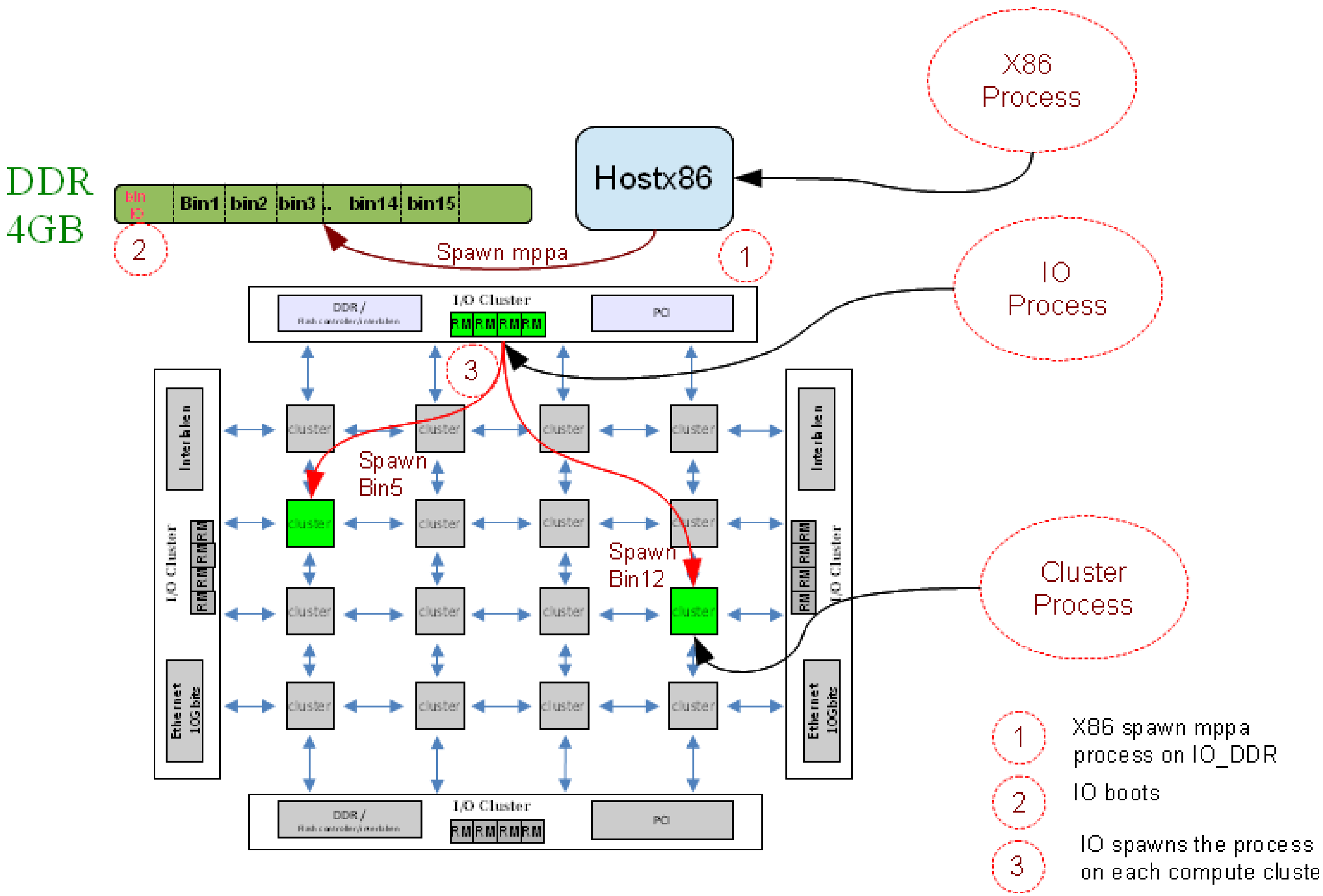
NodeOS boot (inside a compute cluster)



The MPPA boot

- An *helloworld* example is available:
  - /usr/share/AccessCore/Apps/MPPAIPC-Examples/hello_world
- First step: x86 -> IO cluster using PCI
  - A *mppa_spawn* is called from the x86 to transfer a multi-binary to the DDR of the IO cluster using PCIe (./src/host/main_host.c)
- Second step: IO cluster -> Computer cluster (PC0)
  - A *mppa_spawn* is called from the IO-cluster to transfer a binary from the DDR of the IO cluster to the SMEM memory of a computer cluster using the NoC (./src/k1-io/io_main.c)
- Third step: Computer cluster (PC0) -> all PC cores
  - Some *pthread_create* are called to create threads on all the core of the compte clusters (./src/k1-cluser/cluster_main.c)

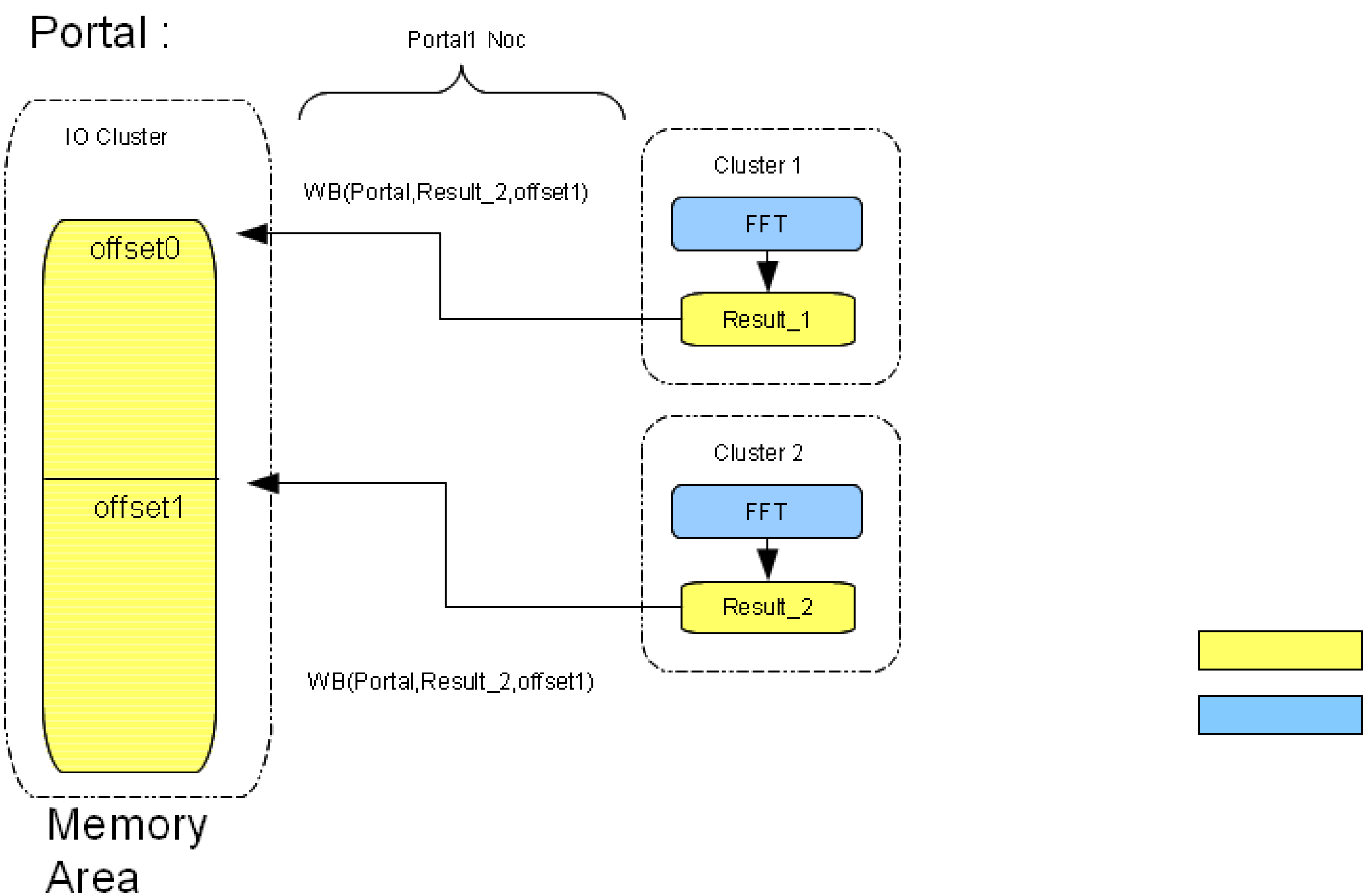
The MPPA boot



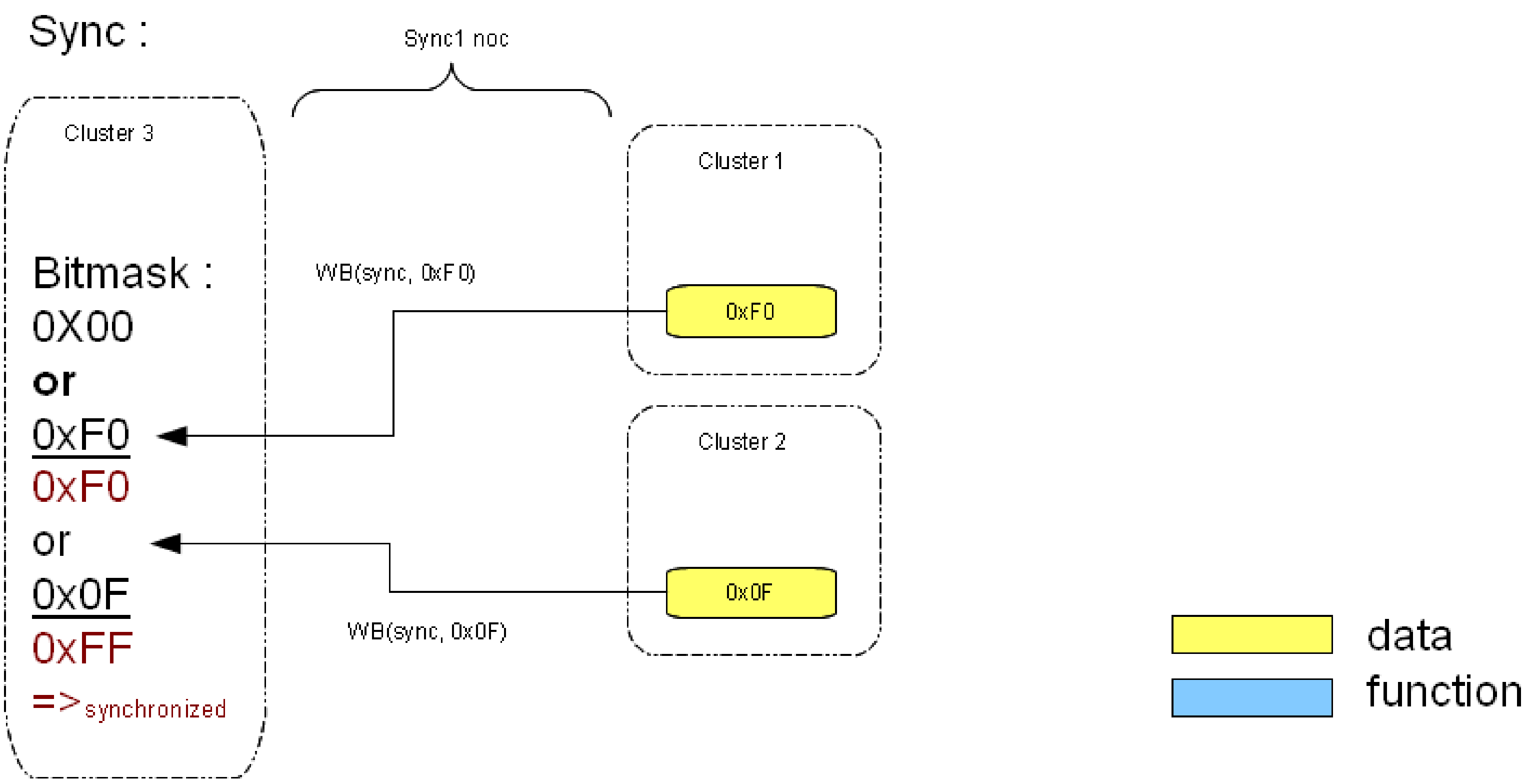
The IPC objects

- There are three communication objects to transfer data between clusters
  - Docs: Docs: /usr/local/k1tools/doc/Specifications/Process/Process.pdf
  - Portal: used to transfer large blocks of data
  - Rqueue: remote queue to transfer small messages
  - Sync: half barrier to synchronise between clusters

MPPA IPC objects (inter-cluster communication)

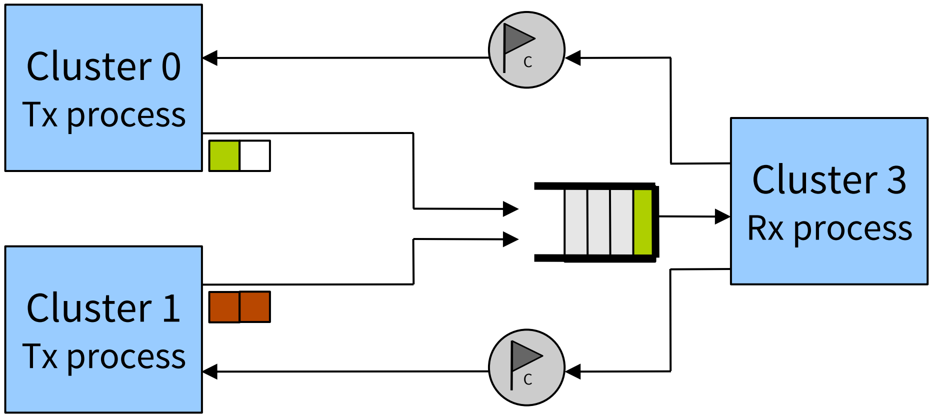


MPPA IPC objects (inter-cluster communication)



MPPA IPC objects (inter-cluster communication)

Rqueue : message passing with credits



Example Application from AccesscoreApps : AES

- AES encryption
- Found in /usr/share/AccessCore/Apps/MPPAIPC-AES
- io_main.c: process running on IO cluster
- cluster_main.c: process running on compute clusters (same process on 5 compute clusters)

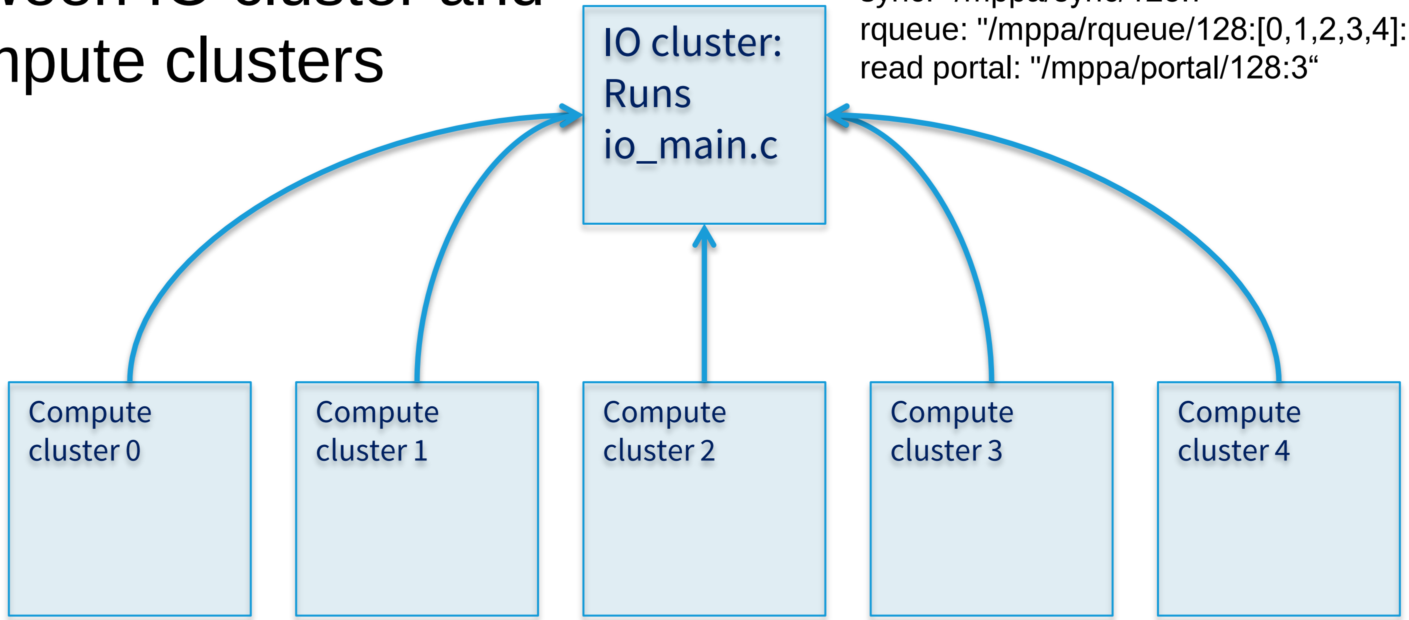
Import the Application into k1-eclipse

- Open the k1-eclipse tool
- Select File->Import
- Import project (make sure you check the Copy to workspace box)
  - This will import the project's source files as well as the build and run configurations

AES Block Diagram

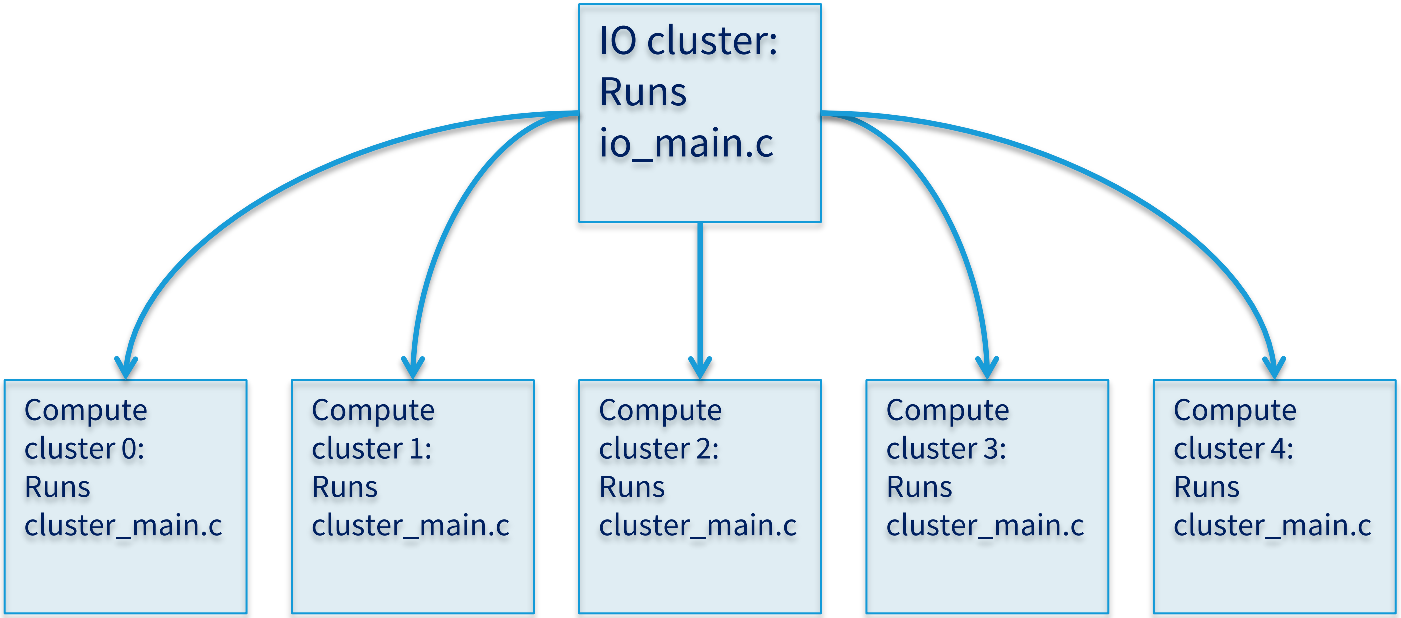
IO cluster: Open queues, syncs and portals for communication between IO cluster and Compute clusters

sync: "/mppa/sync/128:7“
rqueue: "/mppa/rqueue/128:[0,1,2,3,4]:3“
read portal: "/mppa/portal/128:3“



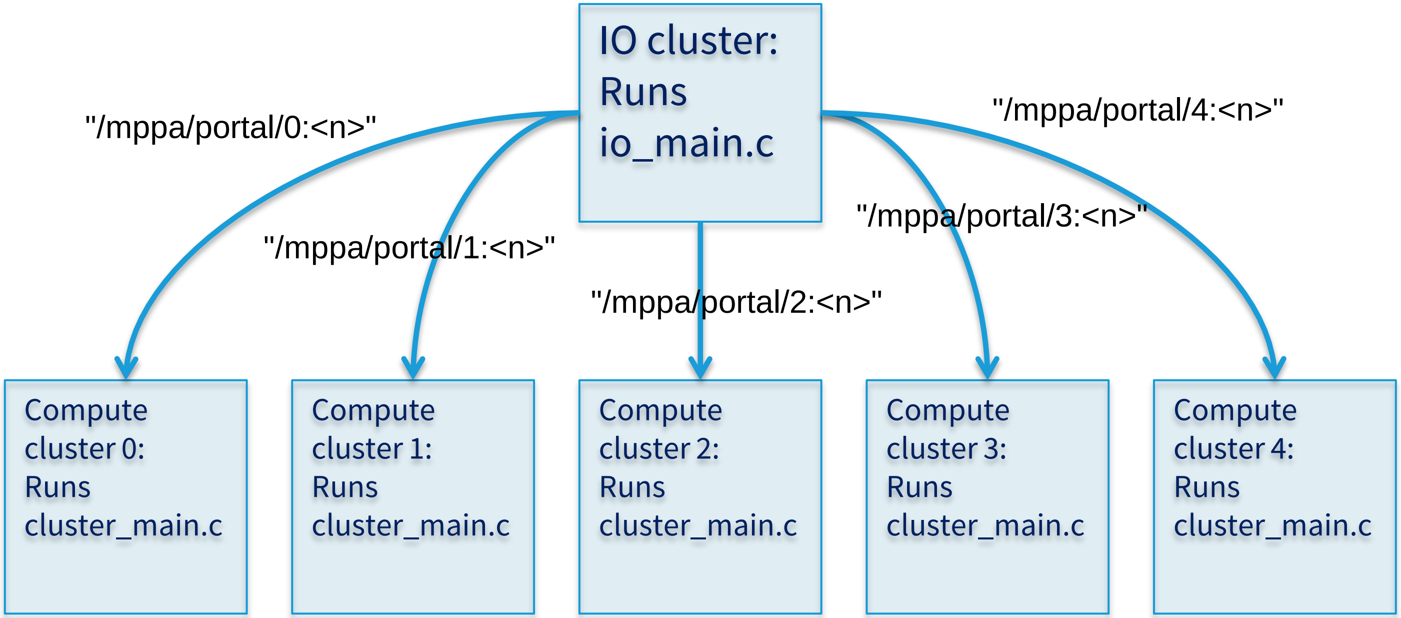
AES Block Diagram

IO Cluster: Spawn cluster processes



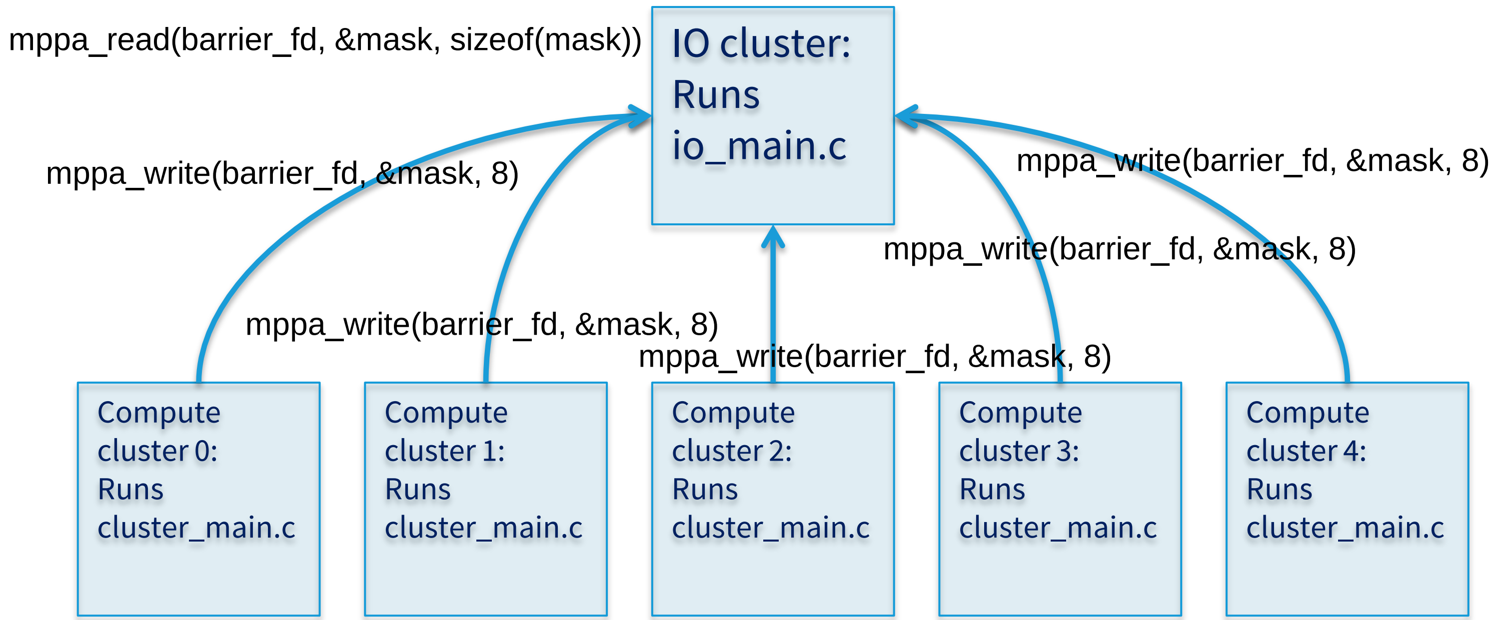
AES Block Diagram

IO Cluster: Open write portals for sending data from IO cluster to compute clusters



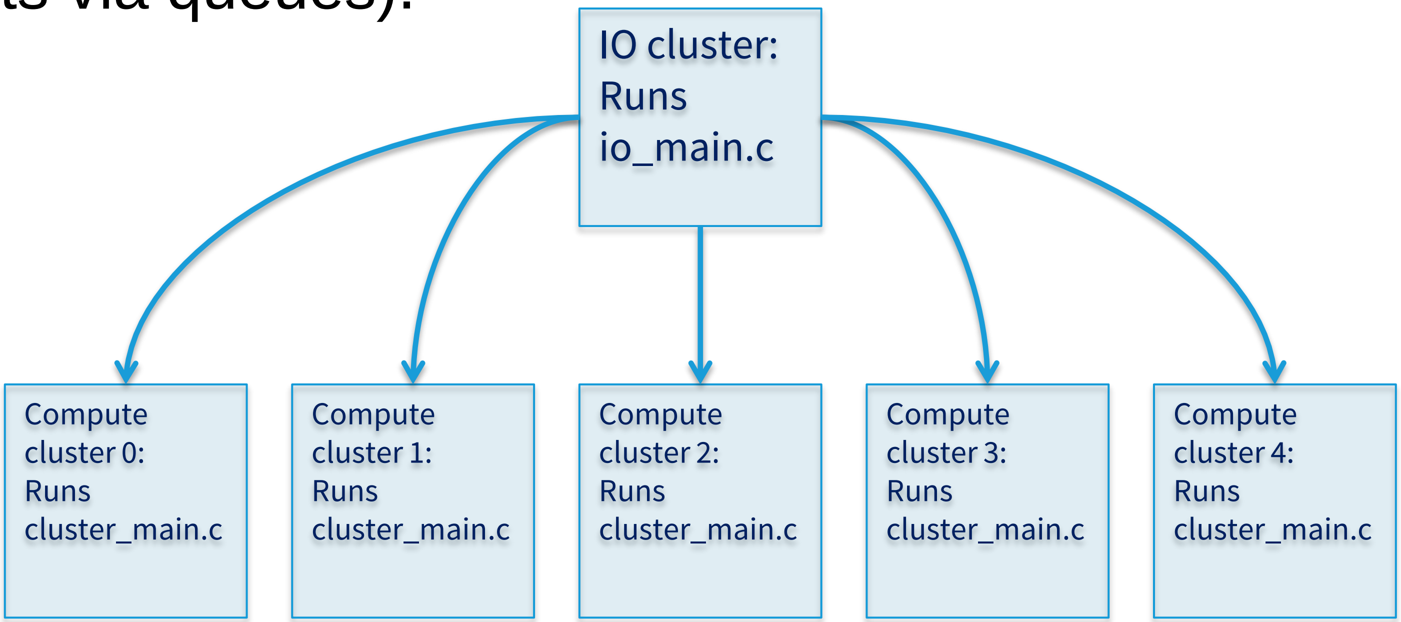
AES Block Diagram

Synchronise all compute clusters with the IO cluster



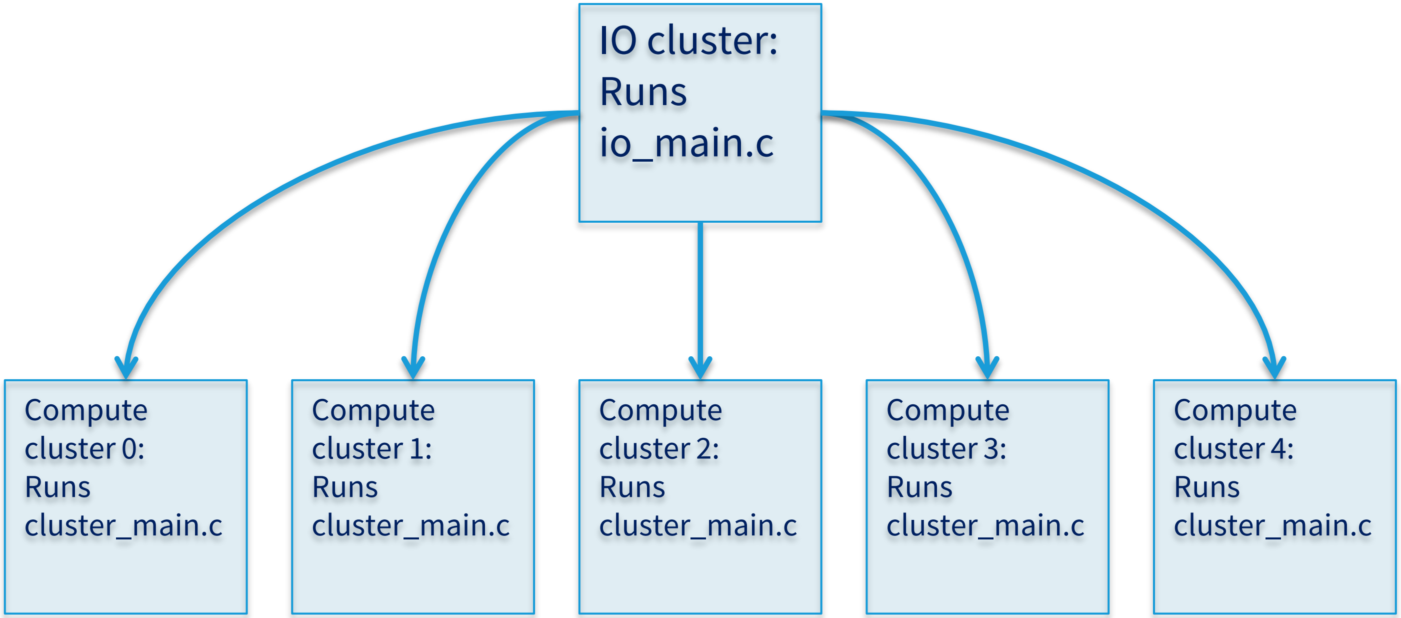
AES Block Diagram

Write data to be processed to cluster portals after synchronisation.
Clusters create pthreads to parallelise the AES task on 6 PEs.
IO cluster sends more data when requested by compute clusters (requests via queues).



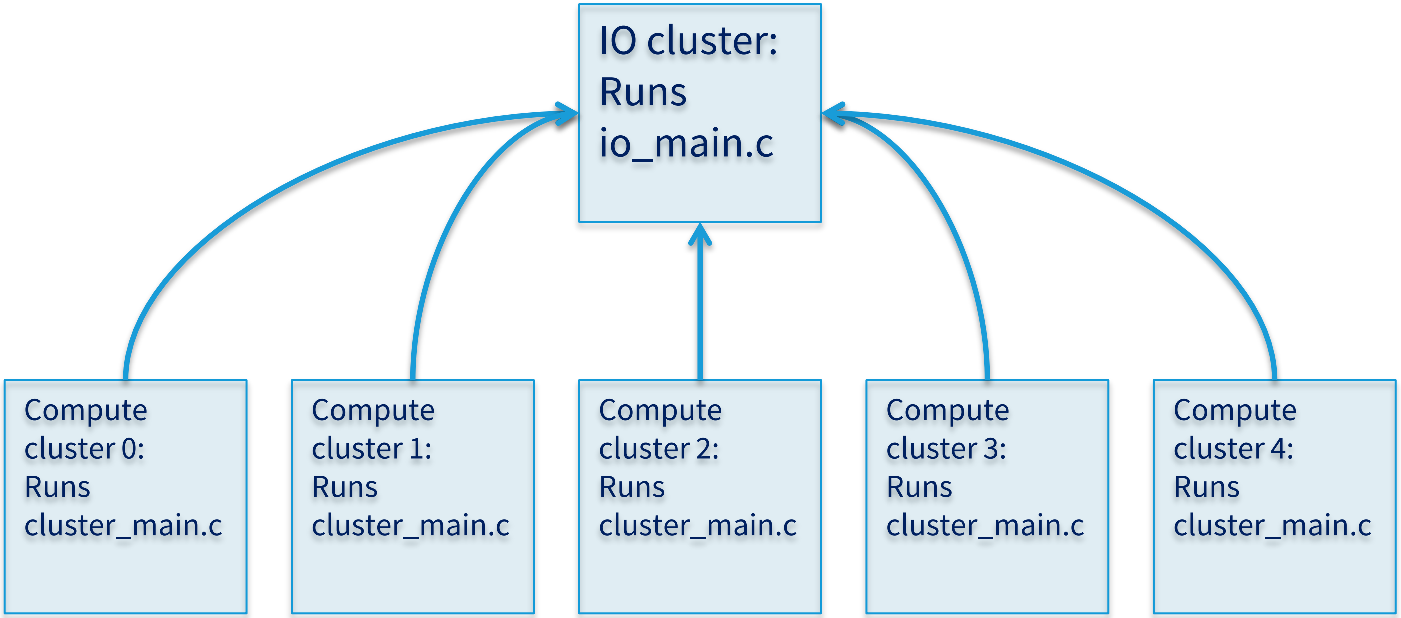
AES Block Diagram

When IO has finished sending data, it will send an End of Transfer message (-1 in this case). This will cause each cluster thread task to complete.



AES Block Diagram

Then IO waits for all the results to arrive on its read portal. It waits for all clusters to exit, and then exits its process.

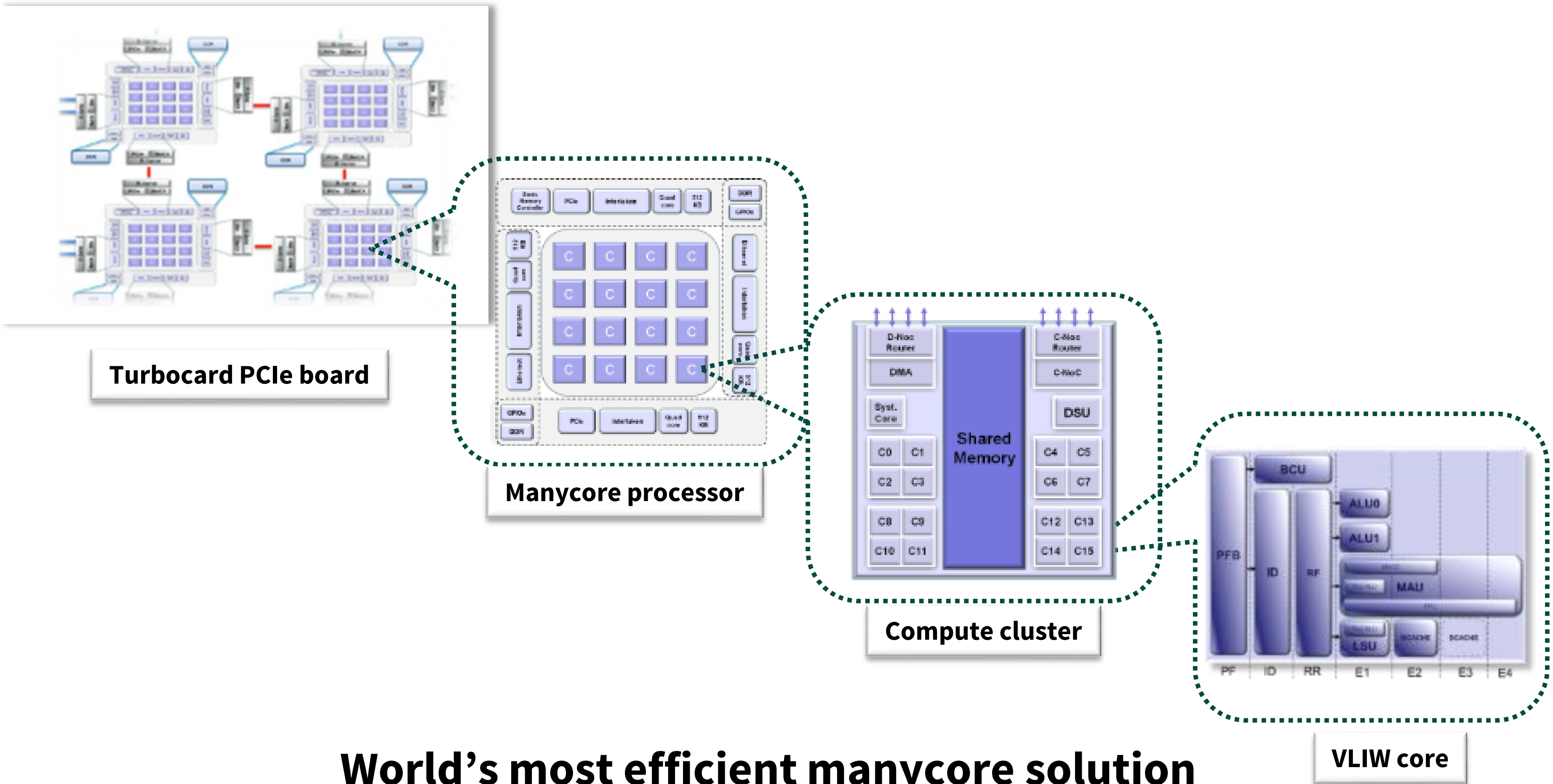


How To Build, View and Run

- Build the application
  - Right-click on the project
  - Select Build Project
- Choose a run configuration to run the application
  - Right-click on the project
  - Select Run as -> Run Configurations
  - Choose a pre-configured K1 Application run configuration from the list (either simulator or HW run)

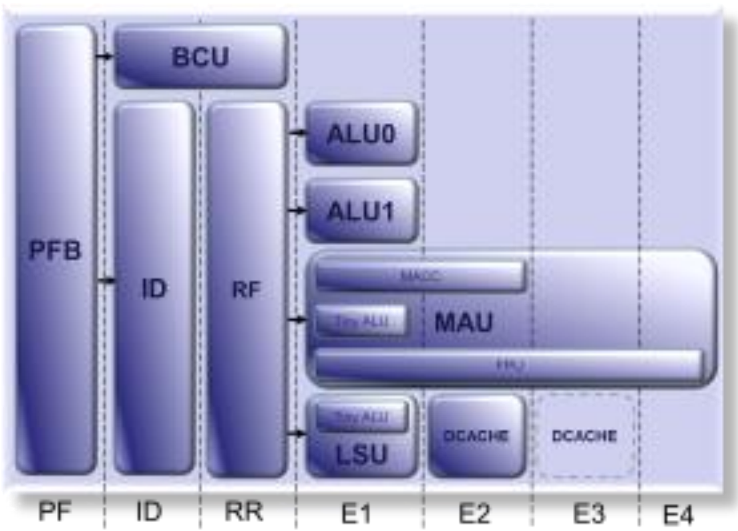
Contents

MPPA® Scalable Massively Parallel Processing



World's most efficient manycore solution for intensive computing market

MPPA®-256 VLIW Core Architecture

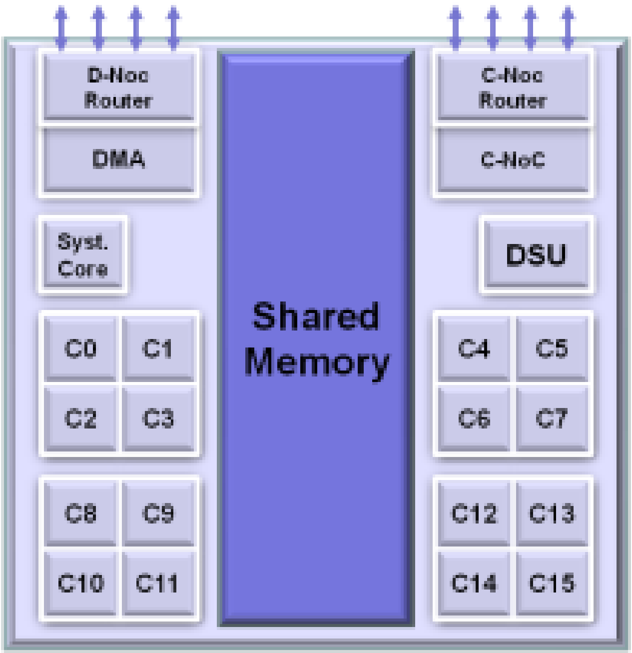


- 5-issue VLIW architecture
- Predictability & energy efficiency
- 32-bit/64-bit IEEE 754 FPU
- MMU for rich OS support

- Data processing code
  - Byte memory alignment
  - Standard & effective FPU
  - Configurable bitwise logic
  - Hardware looping
- System & control code
  - MMU ➔ single memory port ➔ no function unit clustering
- Execution predictability
  - Fully timing compositional core
  - LRU caches, low miss penalty
- Energy and area efficiency
  - 7-stage instruction pipeline, 400MHz
  - Idle modes and wake-up on interrupt

 Confidential Information

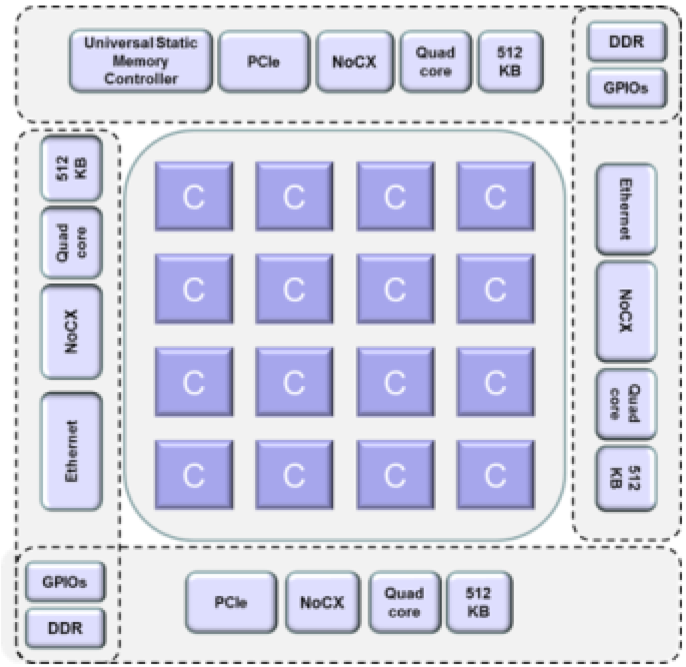
MPPA®-256 Compute Cluster



- 16 PE cores + 1 RM core
- NoC Tx and Rx interfaces
- Debug Support Unit (DSU)
- 2 MB of shared memory

- Multi-banked parallel memory
  - 38,4GB/s of bandwidth @400MHz
- Reliability
  - ECC in the shared memory
  - Parity check in the caches
  - Faulty cores can be switched off
- Predictability
  - Multi-banked shared memory with interleaved or blocked addresses
- Low power
  - Memory banks with low power mode
  - Voltage scaling

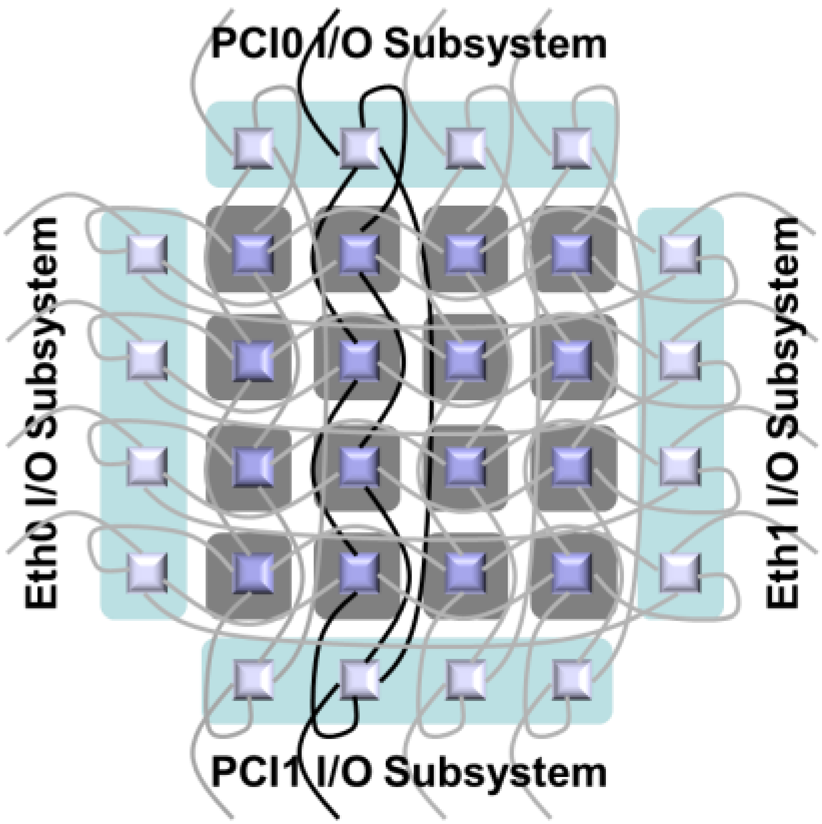
MPPA®-256 Processing Array & I/O Interfaces



- 16 compute clusters
- 4 I/O subsystems with quad-core SMP and DDR memory access
- 2 Networks-on-Chip

- DDR3 Memory interfaces
- PCIe Gen3 interface
- 1G/10G/40G Ethernet interfaces
- SPI/I2C/UART interfaces
- Universal Static Memory Controller (NAND/NOR/SRAM)
- GPIOs with Direct NoC Access (DNA)
- NoC extension through Interlaken interface (NoC Express)

MPPA®-256 Clustered Memory Architecture



- 20 memory address spaces
  - 16 compute clusters
  - 4 I/O subsystems with direct access to external DDR memory
- Dual Network-on-Chip (NoC)
  - Data NoC & Control NoC
  - Full duplex links, 4B/cycle
  - 2D torus topology + extension links
  - Unicast and multicast transfers
- Data NoC QoS
  - Flow control and routing at source
  - Guaranteed services by application of network calculus
  - Oblivious synchronization

Contents

Coming soon : MPPA ACCESSCORE 2.0 SDK

Standard C/C++ Programming Environment

Simulators & Profilers, Debuggers & System Trace

Operating Systems & Device Drivers

C - Low Level Programming
DSP Style

C/C++ - POSIX-Level Programming
CPU Style

OpenCL Programming
GPU Style

Linux

3rd party Real Time OS

eSOL Embedded Excellence

WIND RIVER

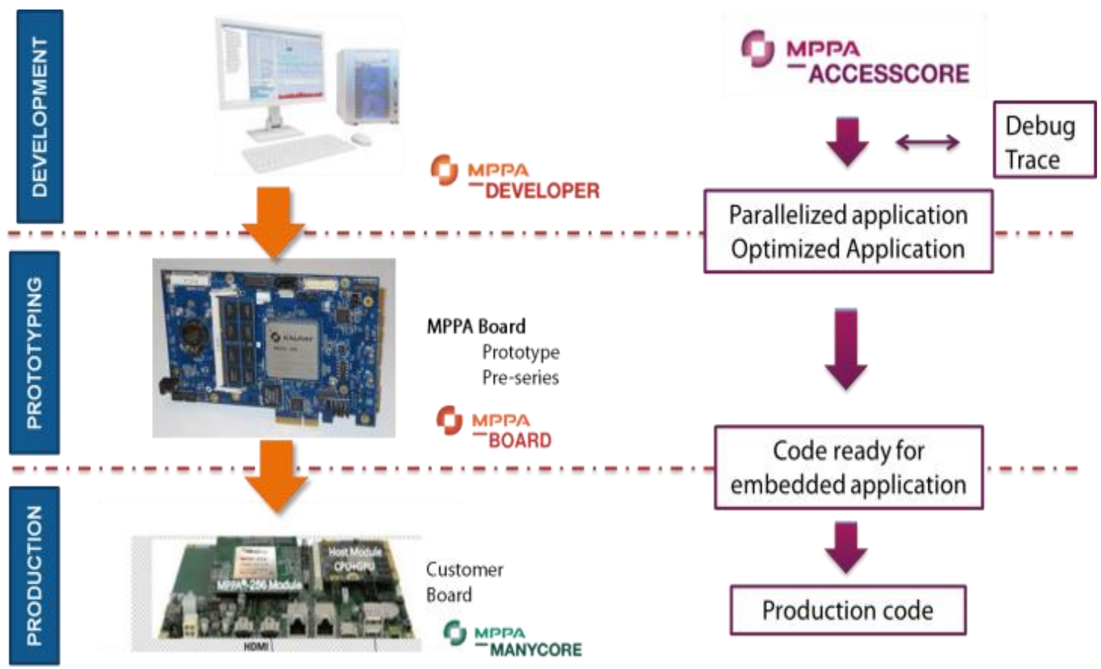
Kalray's Solutions Commercially Available Today

- Processor & Software: Andey MPPA®-256
  - 256 fully C/C++ programmable cores, 28nm (TSMC)
  - High processing performance: 210GFLOPS - 0.7TOPS
  - High energy efficiency: 12W typical
  - Fully scalable (processor tiling support)
  - Standard programming model (standard C/C++/Fortran or OpenCL, GCC, Eclipse tools) with advanced debug capabilities

- Board & Software: Kalray Turbocard2
  - Available since January 2015
  - PCIe Gen3 acceleration card with 4 Andey MPPA chips
  - 1000+ cores
  - 0.9 TFLOPS
  - 100Gb/s FD

EMB01 Board

- EMB01 is the perfect continuation of the MPPA DEVELOPER platform
- EMB01 enables to embed application initially developed on the MPPA DEVELOPER



Kalray's Offering

Solutions	Markets			
	Cloud Acceleration		Critical Embedded	
	Application Acceleration	Networking	Low Volumes (aero. & defence)	Large Volumes (automotive & telecoms)
Boards + Software 	✓	✓		
Processors + Software 		✓	✓	
IP Licensing MPPA —CORE		✓		✓

Kalray 2nd Generation

- 2x to 5x performance improvements for application acceleration and HPEC
- Extra features to address the Intelligent Network Interface Card (INIC) market
- First samples: July 2015 / Ramp-Up: October 2015
- Available as processor, acceleration cards and Open NIC (ONIC)
- Different processor version
  - BOSTAN-N for networking/ONIC
  - BOSTAN-E for eHPC.

Performance

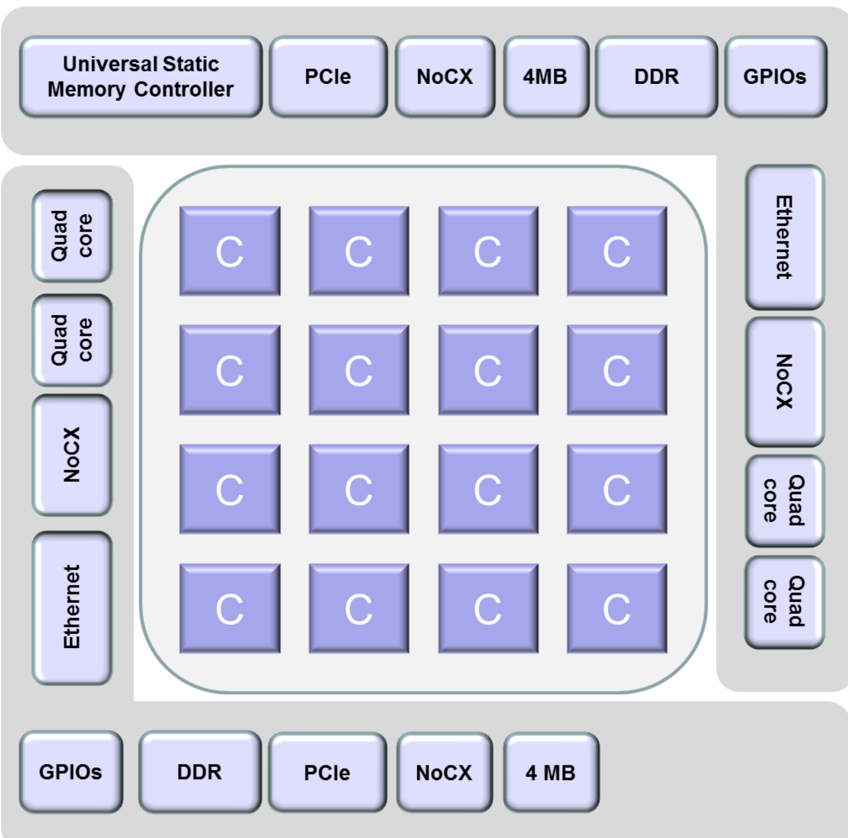
1st Gen → **x2 to x5** → 2nd Gen

Main Extra Feature

- High Speed Ethernet (80Gbps)
- Cores with 64 bits addressing
- Improved Instruction set (floating-pint, video, cryptography, networking)
- Linux SMP support
- PCIe root complex
- Higher frequency

New MPPA®-256 Bostan Processor

64-bit VLIW cores / Ethernet + PCIe subsystem / crypto acceleration

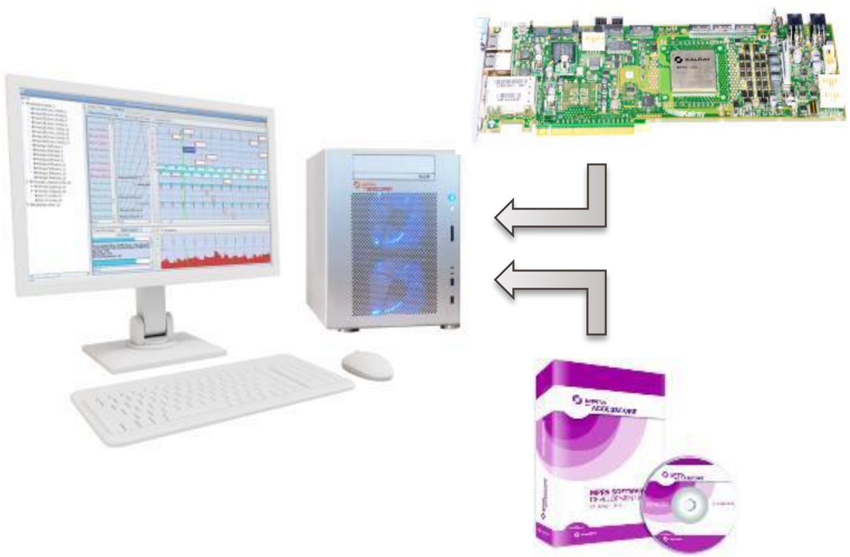


Available Q3-2015

- 288 64-bit VLIW cores
  - **Up to 800MHz (overdrive)**
- Increase Floating-point performance
  - **Simple** 845 GFlops @800 MHz
  - **Double** 422 GFlops @800 MHz
- Network on Chip (NoC) bandwidth
  - 2x 12,8 GB/s @800 MHz
- 40MB of on chip memory
  - **Bandwidth**: 102GB/s
- High Speed Ethernet packet processing
  - **Packet processing**: 2 x 120Mpps @ 64B
- Mission critical support
  - Guaranteed bandwidth
  - Bounded NoC transfers
  - Application isolation between clusters

MPPA® DEVELOPER
Development platform for a quick learning curve

- To develop, optimize and evaluate your applications
  - Access to full computing power of the 256 processors
  - “Ready to develop” concept (no specific set-up)



- A complete package :
  - PCIe board MPPA®-256 Processor
  - PCIe board for debug/probe
  - Intel core I7 CPU 3.6GHz, Linux OS
  - MPPA ACCESSCORE SDK installed
  - Compatible with Multi MPPA board
  - Additional services:
    - Extranet access
    - Support Team access
    - Getting started training

MPPA®-256 PCIe Application Board ACC01
Available in MPPA® DEVELOPER

- Connect to the 4 I/O subsystems
  - 2 PCIe GEN3 x8 interfaces through a x16 PCIe switch
  - 2 DDR3 interfaces
  - 4 Ethernet interfaces (2x10G + 2x1G)
  - 4 Interlaken interfaces
  - NOR flash, GPIOs, leds, buttons, extensions & debug connectors

Kalray’s service and support offers

- Maintenance and support services
  - 1 year of Software upgrades included with the MPPA® DEVELOPER
  - 10 tickets of support
- Application engineering service
  - Consulting, Application support
- Trainings:
  Kalray proposes introductory trainings to master the MPPA® manycore technology whether you are interested with high-level or low-level programming languages: **Dataflow, Posix, OpenCL, Low level**
- Contact sales : info@Kalray.eu

Dataflow training
1 day on site

- Software training
  - Dataflow concepts overview
  - Dataflow language overview
  - Dataflow compilation
  - Running the application : x86 simulation, ISS simulation & Hardware
  - Dataflow over PCIe
- Demos
  - GUI (editor, compilation, simulation, P&R, scheduling ...)
  - Simulation : x86, ISS & Hardware
- Exercises
  - HelloWorld / HelloWorld_cache

POSIX training
1 day on site

- Software training
  - POSIX overview
  - Using the operating system running on compute clusters
  - Using the operating system running on IOs clusters
  - Inter-cluster communication API
  - PCIe interface
- Demos
  - Using Helloworld projects from the eclipse plugin
    - Compilation / simulation / HW run / debugger
    - Assembly trace analysis : k1-cluster -t & k1-dtv
    - K1 Profiling : k1-cluster –callgrind & kcachegrind
- Exercises

OpenCL training
1 day on site

- K1-OpenCL support overview
- How to compile and run (simulation and HW)
- How to trace
- Exercises

Low level training
1 day on site

- Application User:
  - Overview of the LibNoC API
  - Multithreading on a cluster
  - Overview of the LibPCIe API
  - Distributed Shared Memory overview
  - Exercises
- System User:
  - Introduction to the Kalray Hypervisor
  - A simple OS example