Time-Critical Computing on a Single-Chip Massively Parallel Processor

Benoît Dupont de Dinechin
Kalray S.A.
benoit.dinechin@kalray.eu

Duco van Amstel
Kalray S.A.
duco.van-amstel@kalray.eu

Marc Poulhiès
Kalray S.A.
marc.poulhies@kalray.eu

Guillaume Lager
Kalray S.A.
guillaume.lager@kalray.eu

***Abstract*—The requirement of high performance computing at low power can be met by the parallel execution of an application on a possibly large number of programmable cores. However, the lack of accurate timing properties may prevent parallel execution from being applicable to time-critical applications. We illustrate how this problem has been addressed by suitably designing the architecture, implementation, and programming model, of the Kalray MPPA®-256 single-chip many-core processor.**

The MPPA®-256 (Multi-Purpose Processing Array) processor integrates 256 processing engine (PE) cores and 32 resource management (RM) cores on a single 28nm CMOS chip. These VLIW cores are distributed across 16 compute clusters and 4 I/O subsystems, each with a locally shared memory. On-chip communication and synchronization are supported by an explicitly addressed dual network-on-chip (NoC), with one node per compute cluster and 4 nodes per I/O subsystem. Off-chip interfaces include DDR, PCI and Ethernet, and a direct access to the NoC for low-latency processing of data streams.

The key architectural features that support time-critical applications are timing compositional cores, independent memory banks inside the compute clusters, and the data NoC whose guaranteed services are determined by network calculus. The programming model provides communicators that effectively support distributed computing primitives such as remote writes, barrier synchronizations, active messages, and communication by sampling. POSIX time functions expose synchronous clocks inside compute clusters and mesosynchronous clocks across the MPPA®-256 processor.

I. Introduction

Multi-core or many-core platforms are motivated by applications that require high performance, low power, and software programmability. However, such computing platforms appear difficult to exploit for time-critical applications. These applications can be defined by the association of time constraints with information manipulation activities such as acquisition, processing, transport, storage, coordination, and delivery [1].

On time-critical applications, the main issue with multi-core or many-core platforms is ensuring timeliness of computation and communication, given the logical (e.g. code critical sections) or physical interference (e.g. memory hierarchy resources) of tasks that execute concurrently. Addressing this issue involves relying on suitable models of computation to describe applications, and the selection of a computing platform whose run-time software, architecture and implementation support to some extent the following properties:

978-3-9815370-2-4/DATE14/©2014 EDAA

- Deterministic computations: given the same system environment, inputs, and event timing, computation results should be the same.
- Deterministic response times: given the same system environment, inputs, and event timing, computing output should take the same time.
- Predictable response times: given the same system environment, inputs, and event timing, time for computing output should be predictable.
- Composable execution and communication times: updates of the system functionality should have commensurable effects on timing properties.
- High utilization of system resources: the benefits of multi-core or many-core platforms are realized at high utilization of system resources.
- Graceful degradation of properties: over-subscribing system resources should only lead to a gradual loss of timing properties.

In Section II, we present the architecture and implementation features of the MPPA®-256 many-core processor that support time-critical computing. In Section III, we introduce a MPPA® programming model inspired by the POSIX process, IPC, and threads, which is currently used for several time-critical applications including a mixed-criticality use case.

II. Architecture and Implementation

A. MPPA®-256 Processor Overview

The MPPA®-256 processor integrates 288 cores on a single 28nm CMOS chip, and has been designed for applications that require high performance, low power, and software programmability. The distinguishing architectural features of the MPPA®-256 processor are:

- a single VLIW instruction set architecture (ISA) for all the cores, whether used for resource management (RM), or as processing engines (PE);
- a distributed memory organization, based on compute clusters with 16 PE cores and one RM core sharing a local memory, without any direct addressing of the other memory spaces whether on-chip or external;
- a dual network-on-chip (NoC), whose services can be guaranteed by the selection of routes and flow

regulation at the source, and that can be extended across processors;

- and 4 I/O subsystems for the control of peripherals, each of them managed by a quad core CPU based on the same VLIW ISA, that sees the attached DDR memory in the local address space.

An overview of the MPPA®-256 processor global architecture appears in Figure 1, where the grey and blue areas represent the local memory spaces of the compute clusters and the I/O subsystems respectively. Each square box corresponds to a switching node and interface of the dual network-on-chip (NoC), for a total of 32 nodes: one per compute cluster and four per I/O subsystem. The four I/O subsystems are represented on the four sides, two of them include the Ethernet 10G controllers and the two others the PCIe Gen3 controllers. The dual NoC topology is based on a 2D torus augmented with direct links between I/O subsystems, and with NoC extension (NoCX) links that are used for tiling MPPA® processors together or connecting them to a companion FPGA.

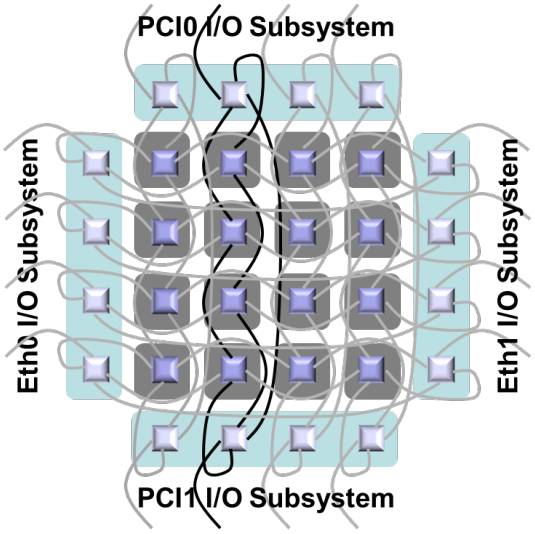

Fig. 1. MPPA®-256 processor memory spaces and dual network-on-chip.

The 16 compute clusters and the 4 I/O subsystems of the MPPA®-256 processor are connected by two parallel NoC with bi-directional links, one for data (D-NoC), and the other for control (C-NoC). Each NoC node is associated with a D-NoC router and a C-NoC router. The two NoC are identical with respect to the nodes, the 2D torus topology, and the wormhole route encoding. They differ at their device interfaces, and by the amount of packet buffering in routers. NoC traffic through a router does not interfere with the memory buses of the underlying I/O subsystem or compute cluster, unless the NoC node is a destination for the transfer.

Each NoC packet is composed of a header and a payload. The header contains a bit-string that encodes the route as direction bits for each NoC node, a 'tag' that is similar to an Internet port number, a EOT (end-of-transfer) bit, and an optional offset. The route bit-string encodes unicast, multicast, or broadcast transfer. For multicast, additional bits in the route trigger packet delivery at any NoC node on the route to the underlying compute cluster or I/O subsystem. All multicast

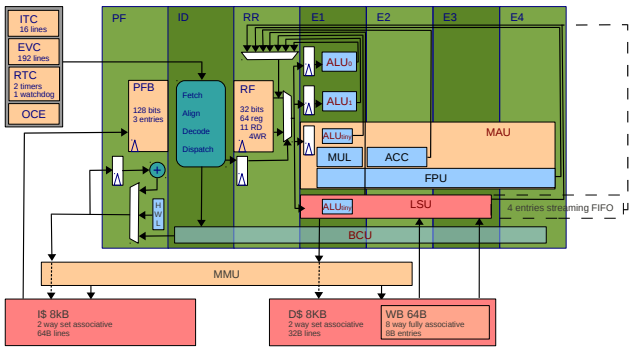

Fig. 2. MPPA® VLIW core instruction pipeline.

or broadcast targets receive the same payload, tag, EOT bit, and offset. The D-NoC and the C-NoC both ensure reliable delivery, and messages that follow the same route arrive in order. However, there is no acknowledgment at the source node that a message has been received at the destination node(s).

B. VLIW Architecture and Core

The choice of a VLIW architecture for the cores of the MPPA®-256 processor was initially motivated by the objective to maximize the exploitation of instruction-level parallelism under constrained power and chip area. The other design objective of this VLIW architecture was to meet the needs of both application code and system software, in order to keep a single ISA for all cores on the MPPA®-256 processor. Application code considered is primarily from numerical, signal, and bit-level processing domains. System software that must execute efficiently includes run-time for the programming models, operating system kernels, device drivers, and I/O stacks.

Running such system software efficiently and safely mandates a full-featured memory management unit (MMU). In turn, this constrains the core to be fitted with a single access port to data memory. Another design principle of the MPPA® VLIW core architecture is to maximize the use of the multi-ported register file and associated bypass logic, which is an expensive resource. As a result, all data types, whether predicates, addresses, integers, floating-point, share the same 32-bit registers, and register pairs are used for the 64-bit data types. Predication only applies to data move and memory access instructions, with their predicate being computed by an implicit 32-bit comparison of a register operand to zero.

The instruction pipeline of the MPPA® VLIW core is displayed in Figure 2. Up to five RISC-like instructions may issue every cycle: branch/control (BCU); load/store (LSU); multiply-accumulate or floating-point (MAU); and two general-purpose 32-bit ALUs that can be paired to operate as a single 64-bit ALU. A subset of the 32-bit ALU instructions may also be executed on the LSU or the MAU. On the instruction side, the core is connected to memory through a prefetch buffer and a private instruction cache. The data side connection goes through a private write-through data cache with a write buffer.

A significant benefit of VLIW architecture with regards to time-critical computing is that a core can be implemented so that timing anomalies are limited or eliminated. A timing

anomaly is a situation where a local worst-case execution time does not contribute to the global worst-case [2]. Timing anomalies prevent accurate static timing analysis at the core level, as a result execution times may no longer be predictable or composable within manageable margins. The MPPA® VLIW core implementation eliminates timing anomalies and supports accurate static timing analysis by the following:

- instruction and data caches implement the LRU replacement policy, which is free of timing anomalies and performs best for static timing analysis [3];
- instruction pipeline and execution pipelines do not have hazard, except for the double-precision floating-point multiply which stalls one cycle irrespective of the values processed;
- there is no branch prediction, except for a hardware looping feature that branches back to the loop header with zero overhead and has constant initial penalty;
- other sources of non-deterministic or data-dependent timing, such as OoO execution in a superscalar core, do not exist in a VLIW implementation.

In fact, preliminary work done by AbsInt[1] in the setting of the CERTAINTY FP7 project[2] has concluded that the Kalray VLIW core qualifies as *fully timing compositional*, that is, does not exhibit timing anomalies. As a result, timing analysis can safely follow local worst-case only [2].

C. Compute Cluster Memory System

The MPPA®-256 compute cluster (Figure 3) is a multi-core whose local memory (SMEM) is shared by 17 identical VLIW cores without cache coherency. The 16 first cores, referred to as processing elements (PEs), are dedicated to application code processing. The 17th core, referred to as the resource manager (RM), is distinguished by its privileged connections to the NoC interfaces through event lines and interrupts. The other bus masters on the cluster shared memory are the NoC Rx (receive) interface, the NoC Tx (transmit) interface, and the debug support unit (DSU).

The compute cluster shared memory comprises 16 independent memory banks of 16384×64-bit words (not including ECC bits), for a total capacity of 2MB. Each memory bank is associated with a dedicated request arbiter that serves 12 bus masters: the D-NoC Rx (receive) interface, the D-NoC Tx (transmit) DMA engine, the DSU, the resource manager (RM), and 8 PE cores pairs (Figure 4). The 16 memory banks are arranged in two *sides* of 8 banks, called *left* and *right*. The connections between the memory bus masters are replicated in order to provide independent access to the two sides.

The access path from a PE core to a memory bank starts with a path private to a PE core pair. Access to this private path is arbitrated round-robin among four possible sources: the instruction cache (IC) and the data cache (DC) of each core in the PE core pair. The private paths of the 8 PE core pairs are connected to the 16 memory bank arbiters. Other bus masters (D-NoC Rx, D-NoC Tx, DSU, RM) have their own private path also connected to the 16 memory bank arbiters.

[1] http:///www.absint.com/
[2] http:///www.certainty-project.eu/

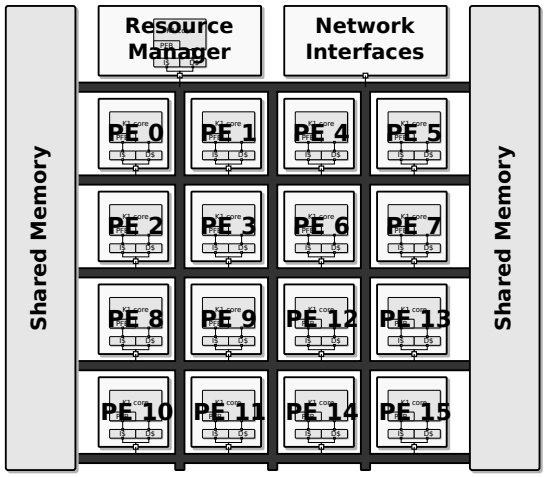

Fig. 3. MPPA®-256 compute cluster overview.

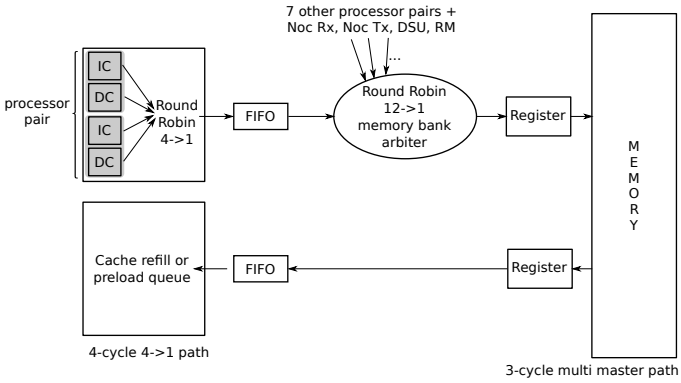

Fig. 4. SMEM memory request arbitration.

The SMEM address mapping can be configured either as *interleaved* or as *blocked*. In the interleaved configuration, bits 6 to 9 of the byte address select the memory bank, so sequential addresses move from one bank to another every 64 bytes (8×64-bit words) as illustrated in Figure 5. In addition, side selection depends on the 6th bit of the byte address, so the selection by sequential addresses alternates between the *left side* and the *right side* every 64 bytes. In the blocked address configuration, each bank spans 128KB consecutive addresses as illustrated in Figure 6. The high-order bit of the address selects the side, so the *right* side covers addresses from 0 to 1MB, and *left* side covers addresses above 1MB.

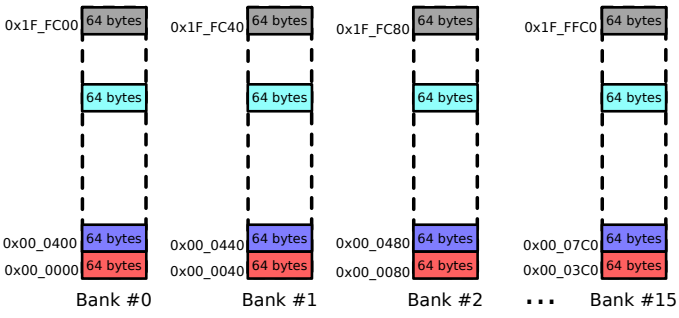

Fig. 5. SMEM interleaved address mapping.

The SMEM address mapping configuration has no func-

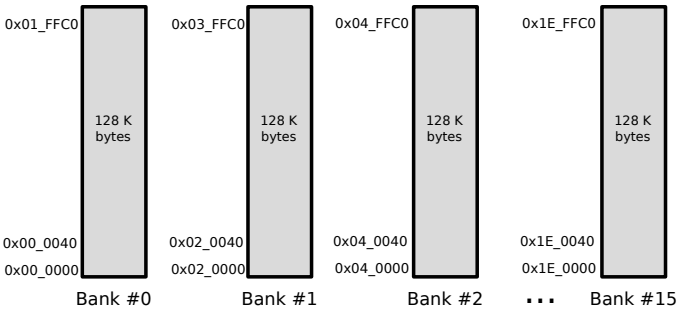

Fig. 6. SMEM blocked address mapping.

tional effects but has significant performance and timing effects. The interleaved configuration is better suited to parallel code regions, where memory references tend to spread evenly across the memory banks so the overall memory throughput is maximized [4]. The blocked configuration is used for time-critical applications to control interference between cores. Precisely, by locating the private code and data of each PE on a different memory bank, and by ensuring that the two banks associated to the two PE in a core pair are on different sides of the SMEM, it is guaranteed that no interference between PE cores occurs on the buses, arbiters, or memory banks.

D. D-NoC Guaranteed Services

The D-NoC is dedicated to high bandwidth data transfers and may operate with guaranteed services, thanks to non-blocking routers and flow regulation at the source node [5]. The flow regulation is currently founded on the (σ, ρ) network calculus [6], which defines a set of linear constraints on the individual link bandwidths ("capacity constraints") and on the router buffer sizes ("backlog constraints"). A flow service obeys (σ, ρ) if for any time interval $[u, v]$ the amount of data serviced is not greater than $\sigma + \rho(v - u)$. An example of such a flow can be observed in figure 7.

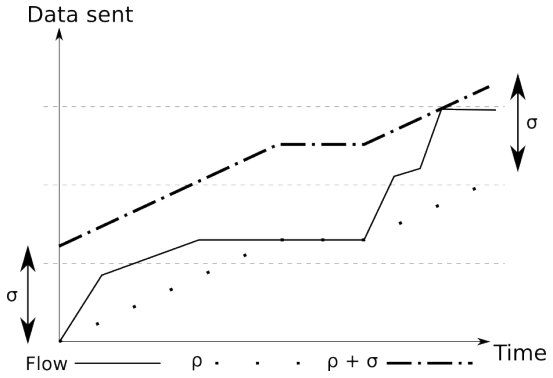

Fig. 7. Flow constrained by a (σ, ρ) model.

In the MPPA®-256 processor, each connection is associated with a bandwidth quota which is enforced at the source node by a regulator also known as *packet shaper*. This regulator is configured via two parameters allowing the user to configure a (σ, ρ) flow:

- `window_length` (τ), global for the NoC node;
- `bandwidth_quota` (β), set for each regulator;

At each cycle, the regulator compares the length of a packet scheduled for injection plus the number of flits sent within the previous τ cycles to β. If not greater, the packet is injected at the rate of one flit per cycle.

The initial (σ, ρ) is thus set at the source node through τ and β (all measured in units of 32-bit flits, including header flits). We link these parameters with the (σ, ρ) model by observing that $\rho = \beta/\tau$. This corresponds to the fact that no regulator may let through more than β flits over any duration τ. On the other hand, the regulator is allowed to emit continuously until having sent β flits within exactly β cycles. This defines a point on the $\rho + \sigma$ linear function and by regression the value of the function at $t = 0$ (corresponding to σ) is: $\sigma = \beta(1 - \beta/\tau)$.

The MPPA® NoC routers multiplex flows originating from different directions without back-pressure. Each originating direction has its own FIFO buffer at the output interface, so flows interfere on a node only if they share a link to the next node. This interface performs a round-robin (RR) arbitration at the packet granularity between the FIFOs that contain data to send on a link (Figure 8). The NoC routers have thus been designed for simplicity while inducing a minimal amount of perturbations on (σ, ρ) flows. Since RR arbitration is a special case of the weighted fair queuing (WFQ) discipline, the worst-case delay of any flow can be computed as [7]:

$$\text{delay} = \frac{\sigma + (n-1)P_{size}}{\rho} + n(d + P_{size})$$

Here, P_{size} is the maximum packet size, n is the number of hops between source and sink, and d is a fixed router delay.

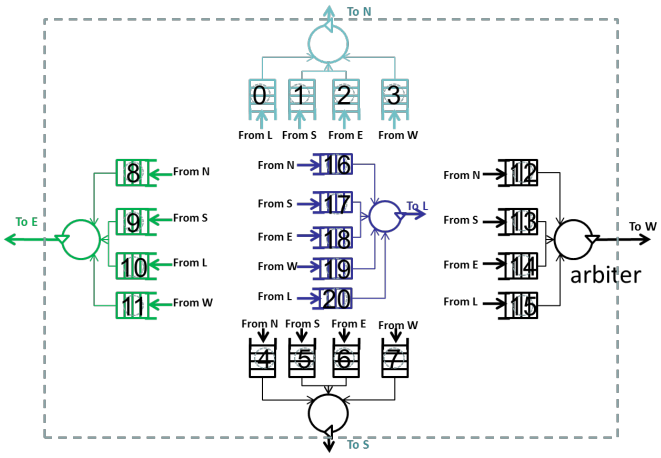

Fig. 8. MPPA®-256 NoC router model.

Configuring the MPPA® D-NoC for guaranteed services is primarily dependent on the selection of routes between nodes. Once the routing phase of all existing flows has been performed, the next step is to express the constraints on the flow rates $\Gamma = (\rho_i)$ only, given that on the MPPA® NoC the (σ_i) values at the source nodes are fixed to $\sigma_i = \rho_i(1 - \rho_i)\tau$:

- Application constraints: Some flows require a minimal injection rate ρ_f.

- Link capacity constraints: $\sum_{f \in F} \rho_f \leq 1$ for any link l shared by a set of flows F.
- Queue backlog constraints: $\sum_{f \in F} \sigma_f^l \leq Q_{size}$ for any FIFO in front of the router arbiter of link l, with Q_{size} the maximum FIFO size.

Assuming that d_f^j is the local delay bound for flow f on link j, the maximum burst size values σ_f^l to consider at link l arbitration grow monotonically along the flow path P_f^l as [7]:

$$\sigma_f^l = \sigma_f + \sum_{j \in P_f^l} \rho d_f^j$$

On the MPPA® NoC, $d_f^j = (m_j - 1)P_{size}$ with m_j the number of directions merging into the router arbiter of link j.

The queue backlog constraints involve the non-linear terms $\sum \sigma_f = \sum \rho_f(1 - \rho_f)\tau$. These can be upper bounded by constants $\frac{n_l - 1}{n_l}\tau$, because $\sum \rho_f \leq 1$ holds for the n_l flows sharing link l. This allows to linearize the queue backlog constraints, while the link capacity constraints are also linear. The full system of linear constraints on flow rates $\Gamma = (\rho_i)$ is then optimized to obtain a solution that guarantees services.

III. POSIX-Level Programming

A. Design Principles

The MPPA® POSIX-level programming model principles are that processes on the I/O subsystems spawn sub-processes on the compute clusters and pass arguments through the traditional `argc`, `argv`, and `environ` variables. Inside compute clusters, classic shared memory programming models such as POSIX threads or the OpenMP language extensions supported by GCC are used to exploit more than one PE core. The main difference between the MPPA® POSIX-level programming and classic POSIX processes & threads programming appears on inter-process communication (IPC).

Precisely, MPPA® IPC is only achieved by operating on special files, whose pathname is structured by a naming convention that fully identifies the NoC resources used when opening in either read or write mode (Table I). Like POSIX pipes, those communication objects have distinguished transmit (Tx) and receive (Rx) endpoints that must be opened in modes `O_WRONLY` and `O_RDONLY` respectively. Unlike pipes however, they may have multiple Tx or Rx endpoints, and support POSIX asynchronous I/O operations with call-back. Following the canonical 'pipe-and-filters' software component model where POSIX processes are the atomic components, we call these communication objects connectors.

B. NoC Connectors Summary

Sync A 64-bit word in the Rx process that can be OR-ed by N Tx processes. When the result of the OR equals -1, the Rx process is notified so a `read()` returns non-zero.

Portal A memory area of the Rx process where N Tx processes can write at arbitrary offsets. The Rx process is not aware of the communication except for a notification count that unlocks the Rx process when the `trigger` supplied to `aio_read()` is reached.

RQueue Atomic enqueue of *msize*-byte messages from several Tx processes, and dequeue from a single Rx process. The RQueue connector implements the remote queue [8], with the addition of flow control. This is an effective $N : 1$ synchronization primitive, by the atomicity of the enqueue operation [9].

Channel A communication and synchronization object with two endpoints. Default behavior is to effectuate a rendezvous between the Tx process and the Rx process, which transfers the minimum of the sizes requested by the read and the write without intermediate copies.

Sampler Message broadcast from one Tx process to several Rx processes. A Rx process is not aware of the communication but is ensured to find a valid and stable pointer to the latest message sent. This connector is provided to implement the communication by sampling (CbS) mechanism [10].

C. Support of Distributed Computing

Split Phase Barrier The arrival phase of a master-slave barrier [11] is directly supported by the Sync connector, by mapping each process to a bit position. The departure phase of a master-slave barrier [11] is realized by another Sync connector in $1 : M$ multi-casting mode.

Active Message Server Active messages integrate communication and computation by executing user-level handlers which consume the message as arguments [12]. Active message servers are efficiently built on top of remote queues [8]. In case of the RQueue connector, the registration of an asynchronous read user call-back enables to operate it as an active message server.

Remote Memory Accesses One-sided remote memory access operations (RMA) are traditionally named PUT and GET [12], where the former writes to remote memory, and the latter reads from remote memory. The Portal connector directly supports the PUT operation on a Tx process by writing to a remote D-NoC Rx buffer in offset mode. The GET operation is implemented by active messages that write to a Portal whose Rx process is the sender of the active message.

D. Support of Time-Critical Computing

A first requirement for the support of time-critical computing is establishing a time reference between tasks. On the MPPA®-256 processor, every compute cluster and I/O subsystem is fitted with a debug support unit (DSU), which includes a 64-bit timestamp counter. Each counter is addressable in the local memory space and can be read by any core. Global initialization of these counters is typically performed by a broadcast message on the C-NoC, resulting in time base offsets between clusters of about a dozen cycles. As the whole MPPA®-256 processor is driven by a single hardware clock, all these counters are mesosynchronous. Each core also implements its own real-time clock, which supports an accurate and lightweight implementation of POSIX timers.

Assuming an initial mapping of tasks to compute clusters has been determined, porting a time-critical application to the MPPA®-256 processor requires engineering in the areas of:

- Controlled execution of tasks on the compute clusters.

Type	Pathname	Tx:Rx	`aio_sigevent.sigev_notify`
Sync	`/mppa/sync/`*rx_nodes*`:`*cnoc_tag*	$N:M$	
Portal	`/mppa/portal/`*rx_nodes*`:`*dnoc_tag*	$N:M$	`SIGEV_NONE`, `SIGEV_CALLBACK`
RQueue	`/mppa/rqueue/`*rx_node*`:`*dnoc_tag*`/`*tx_nodes*`:`*cnoc_tag*`/`*credits*`.`*msize*	$N:1$	`SIGEV_NONE`, `SIGEV_CALLBACK`
Channel	`/mppa/channel/`*rx_node*`:`*dnoc_tag*`/`*tx_node*`:`*cnoc_tag*	$1:1$	`SIGEV_NONE`
Sampler	`/mppa/sampler/`*rx_nodes*`:`*dnoc_tag*	$1:M$	`SIGEV_NONE`

TABLE I. NoC CONNECTOR PATHNAMES, SIGNATURE, AND ASYNCHRONOUS I/O SIGEVENT NOTIFY ACTIONS.

- Intra-processor communication and synchronization through NoC connectors.
- Controlled sharing of resources on the I/O subsystems, starting with DDR memory accesses.
- Local connection to low-latency, high speed I/O data streams through NoC extensions.
- Global communication over synchronized or deterministic variant of Ethernet.

The direct port to the MPPA®-256 processor of a cockpit flight management system (FMS), a time-critical application proposed by the CERTAINTY FP7 project as its motivating use case, illustrates how some of these issues may be addressed.

Task Interference On the compute clusters, the POSIX extension primitive `pthread_attr_setaffinity_np` enforces explicit mapping between threads and PE cores. The blocked SMEM configuration is used in combination with linker scripts to position code and data including stack of each thread on non-interfering memory paths and banks.

Periodic Tasks These tasks are activated at a given period. At thread creation, the current date of `CLOCK_REALTIME` is read thanks to the POSIX primitive `clock_gettime`. At the end of the computation, the period is added to the start time and the POSIX primitive `pthread_cond_timedwait` is called with the resulting time as timeout parameters. That way the task waits until the period expires.

Pseudo-Periodic Tasks These tasks are restarted with a dynamic delay, which may be selected during the computation of the task. At the end of computations, the task has to wait the selected amount of time for restart. This waiting is simply performed with the POSIX primitive `nanosleep`.

Sharing of I/O Resources The sharing of I/O resources, in particular DDR access, can be controlled by implementing a global time-triggered task scheduling approach, as proposed for mixed-criticality cases such as this FMS application [13]. This approach relies on barrier synchronizations, whose implementation is efficient on the MPPA®-256 processor thanks to the Sync connector. A simpler approach applicable when there is a single level of criticality is the sliced execution model [14].

IV. SUMMARY AND CONCLUSIONS

The distinguishing architectural features of the Kalray MPPA®-256 processor include a single VLIW ISA for all the cores, clusters of cores that share a multi-banked local memory system, and an explicitly routed dual NoC with flow regulation at the source. Combining these features was originally motivated by the objectives of energy-efficient computing and architectural scalability. While developing this processor, it became apparent that excellent support for time-critical applications could be provided as well with limited adaptations of the implementation. We believe that the resulting MPPA®-256 processor is ideally positioned for a new generation of applications where high-performance, energy-efficient parallel processing is required to implement time-critical functions.

REFERENCES

[1] J. D. Northcutt and E. M. Kuerner, "System support for time-critical applications," in *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*. London, UK, UK: Springer-Verlag, 1992, pp. 242–254.

[2] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE Transaction on Computer-Aided Design*, vol. 28, no. 7, pp. 966–978, Jul. 2009.

[3] J. Reineke, D. Grund, C. Berg, and R. Wilhelm, "Timing predictability of cache replacement policies," *Real-Time Systems*, vol. 37, no. 2, pp. 99–122, Nov. 2007.

[4] D. Y. Chang, D. J. Kuck, and D. H. Lawrie, "On the effective bandwidth of parallel memories," *IEEE Trans. Comput.*, vol. 26, no. 5, pp. 480–490, May 1977.

[5] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson, "Flow regulation for on-chip communication," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09, 2009, pp. 578–581.

[6] R. L. Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, 1991.

[7] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1396, 1995.

[8] E. A. Brewer, F. T. Chong, L. T. Liu, S. D. Sharma, and J. D. Kubiatowicz, "Remote queues: exposing message queues for optimization and atomicity," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95, 1995, pp. 42–53.

[9] M. G. Katevenis, E. P. Markatos, P. Vatsolaki, and C. Xanthaki, "The remote enqueue operation on networks of workstations," *International Journal of Computing and Informatics*, vol. 23, no. 1, pp. 29–39, 1999.

[10] A. Benveniste, A. Bouillard, and P. Caspi, "A unifying view of loosely time-triggered architectures," in *Proceedings of the tenth ACM international conference on Embedded Software*, ser. EMSOFT '10, 2010, pp. 189–198.

[11] O. Villa, G. Palermo, and C. Silvano, "Efficiency and scalability of barrier synchronization on noc based many-core architectures," in *Proceedings of the 2008 international conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '08, 2008, pp. 81–90.

[12] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," in *Proceedings of the 19th annual International Symposium on Computer architecture*, ser. ISCA '92, 1992, pp. 256–266.

[13] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *International Conference on Embedded Software (EMSOFT)*, Montreal, Oct 2013.

[14] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on cots hardware," in *Proceedings of the 25th International Conference on Architecture of Computing Systems*, ser. ARCS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 98–110.