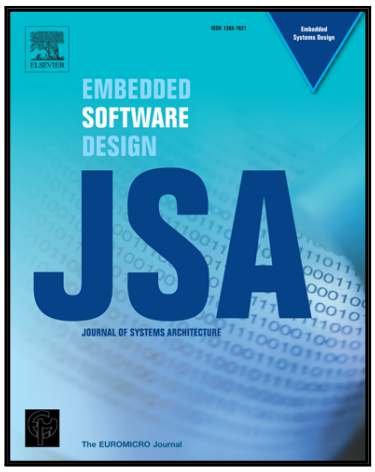

Dynamic Many-process Applications on Many-tile Embedded Systems and HPC Clusters: the EURETILE programming environment and execution platforms

Pier Stanislao Paolucci , Andrea Biagioni , Luis Gabriel Murillo , Frédéric Rousseau , Lars Schor , Laura Tosoratto , Iuliana Bacivarov , Robert L. Buecs , Clément Deschamps , Ashraf El-Antably , Roberto Ammendola , Nicolas Fournel , Ottorino Frezza , Rainer Leupers , Francesca Lo Cicero , Alessandro Lonardo , Michele Martinelli , Elena Pastorelli , Devendra Rai , Davide Rossetti , Francesco Simula , Lothar Thiele , Piero Vicini , Jan H. Weinstock

PII: S1383-7621(15)00142-3
DOI: 10.1016/j.sysarc.2015.11.008
Reference: SYSARC 1322

To appear in: *Journal of Systems Architecture*

Received date: 3 December 2014
Revised date: 20 July 2015

Please cite this article as: Pier Stanislao Paolucci , Andrea Biagioni , Luis Gabriel Murillo , Frédéric Rousseau , Lars Schor , Laura Tosoratto , Iuliana Bacivarov , Robert L. Buecs , Clément Deschamps , Ashraf El-Antably , Roberto Ammendola , Nicolas Fournel , Ottorino Frezza , Rainer Leupers , Francesca Lo Cicero , Alessandro Lonardo , Michele Martinelli , Elena Pastorelli , Devendra Rai , Davide Rossetti , Francesco Simula , Lothar Thiele , Piero Vicini , Jan H. Weinstock , Dynamic Many-process Applications on Many-tile Embedded Systems and HPC Clusters: the EURETILE programming environment and execution platforms, *Journal of Systems Architecture* (2015), doi: 10.1016/j.sysarc.2015.11.008

Dynamic Many-process Applications on Many-tile Embedded Systems and HPC Clusters: the EURETILE programming environment and execution platforms

Pier Stanislao Paolucci[1], Andrea Biagioni[1], Luis Gabriel Murillo[2], Frédéric Rousseau[3], Lars Schor[4], Laura Tosoratto*[1], Iuliana Bacivarov[4], Robert L. Buecs[2], Clément Deschamps[3], Ashraf El-Antably[3], Roberto Ammendola[5], Nicolas Fournel[3], Ottorino Frezza[1], Rainer Leupers[2], Francesca Lo Cicero[1], Alessandro Lonardo[1], Michele Martinelli[1], Elena Pastorelli[1], Devendra Rai[4], Davide Rossetti[1], Francesco Simula[1], Lothar Thiele[4], Piero Vicini[1], Jan H. Weinstock[2]

[1]INFN Roma "Sapienza", [2]RWTH Aachen University – ISS & SSS, [3]Université Joseph Fourier – TIMA Laboratory, [4]ETH Zurich - Computer Engineering and Networks Laboratory, [5]INFN Roma "Tor Vergata"

**corresponding author: laura.tosoratto@roma1.infn.it*

Abstract

In the next decade, a growing number of scientific and industrial applications will require power-efficient systems providing unprecedented computation, memory, and communication resources. A promising paradigm foresees the use of heterogeneous many-tile architectures. The resulting computing systems are complex: they must be protected against several sources of faults and critical events, and application programmers must be provided with programming paradigms, software environments and debugging tools adequate to manage such complexity. The EURETILE (European Reference Tiled Architecture Experiment) consortium conceived, designed, and implemented: 1- an innovative many-tile, many-process dynamic fault-tolerant programming paradigm and software environment, grounded onto a lightweight operating system generated by an automated software synthesis mechanism that takes into account the architecture and application specificities; 2- a many-tile heterogeneous hardware system, equipped with a high-bandwidth, low-latency, point-to-point 3D-toroidal interconnect. The inter-tile interconnect processor is equipped with an experimental mechanism for systemic fault-awareness; 3- a full-system simulation environment, supported by innovative parallel technologies and equipped with debugging facilities. We also designed and coded a set of application benchmarks representative of requirements of future HPC and Embedded Systems, including: 4- a set of dynamic multimedia applications and 5- a large scale simulator of neural activity and synaptic plasticity. The application benchmarks, compiled through the EURETILE software tool-chain, have been efficiently executed on both the many-tile hardware platform and on the software simulator, up to a complexity of a few hundreds of software processes and hardware cores.

Classifications: 20.100: Embedded Systems; 20.050: Distributed Systems; 30.050: Parallel & Distributed Applications; 30.060: Multimedia & Internet; 30.040: Neural Networks; 10.050: Hardware & Software Co-Design; 10.070: Communication Systems & Interconnection Networks; 20.130: Fault-Tolerant Computing

Keywords: Many-tile Hardware Architectures; Many-tile Simulation; Many-process Applications; Embedded Systems; High Performance Computing; Dynamic Multimedia Application; Spiking Neural Network; Synaptic Plasticity; Fault Tolerance; Task Migration; Application Mapping; Hardware dependent Software; Distributed Network Processor; 3D Toroidal Interconnect.

1 Introduction

The next generation of scientific and industrial applications will have performance and energy requirements that are far beyond the capabilities of current hardware technologies in terms of computation, memory, and communication resources. For instance, computing systems embedded into autonomous cars must soon solve multi-sensorial data fusion and artificial intelligence problems in real-time. Similarly, high-impact scientific applications like cortical simulations - requiring computing systems capable of exaFLOPS, i.e., 10^{18} operations per second if whole human brain is the target - are already finding their way into embedded systems for "traffic surveillance" and other "robotic brain" simulations-[1][2] and [3].

Indeed, there is an interesting design space to explore at the intersection between high-end embedded systems and moderate scale parallel/distributed scientific computing.

The identification of efficient solutions at this scale of hardware and software complexity would enable new embedded applications that could be used as building blocks for larger scale scientific applications. Our work explored this convergence region between HPC and embedded systems.

Placed at this convergence point, a paradigm which promises to meet the performance requirements of these applications sees employment of heterogeneous architectures equipped with the raw processing power provided by a few hundred of cores.
However, the available computing power does not necessarily translate into high performance as the system faces reliability issues. Also, performance is usually limited by the latency (and often by the bandwidth) and intercommunication jitter, or the platform software is not able to exploit the provided hardware parallelism. As a result, new techniques must be developed that tackle the challenges arising when designing heterogeneous many-core systems.
In first place, the interconnection of the computing nodes should not limit the performance of the system. In the HPC field it is well known that the interconnection architecture represents a key element for efficient scaling of system performances, giving way to a peculiar ecosystem of solutions, both commodity and proprietary.
Another set of challenges stems from the necessary shift towards more parallelism. Programming models and related programming environments must be developed that provide system designers the possibility to exploit the concurrency in their applications. At the same time, the architecture-specific implementation of the application must be synthesized automatically.
The third set of challenges concerns the validation of the system. If simulation does not keep pace with the evolution of the architecture, it becomes a major bottleneck in the design process. Finally, the last set of challenges is about the dependability of the system, which is threatened by various error sources. Consequently, fault management techniques must be integrated in all steps of the design process including system optimization, software synthesis and hardware architecture design.

The FET FP7 project European Reference Tiled Architecture Experiment (EURETILE) (2010-2014), evolution of the SHAPES project (2006-2009) [4][5], addressed the above mentioned challenges. EURETILE evolves and investigates a novel foundational parallel programming paradigm, applied to a larger scale many-core hardware architecture equipped with a high-bandwidth and low-latency inter-tile communication infrastructure [6].
We show that the proposed tiled distributed hardware architecture and the corresponding programming paradigm can be applied to both HPC clusters and future embedded computing systems.
To this end, we implemented two different platforms:

- a many-tile fault-aware hardware platform, QUonG [7], based on a custom 3D first-neighbour toroidal interconnect called APEnet+ [8] and x86 multi-core processors and
- a scalable many-tile simulator, the Virtual EURETILE Platform (VEP) [9], which is also based on a 3D toroidal interconnect, but includes a processor (IRISC) more representative of embedded systems.

From a software perspective, EURETILE provided a scalable, efficient and wide-ranging programming framework for many-tile platforms, by exploiting the underlying parallelism and investigating some key architectural enhancements [10].
The proposed programming environment, the Distributed Application Layer (DAL), is based on a new model of computation that explicitly exposes either coarse- and fine-grained parallelism present in applications. Automatic tools are then aimed at obtaining predictable and efficient system implementations by optimally matching the exposed parallelism with the underlying many-tile hardware.
The complete software tool-chain [10] aims at optimizing issues related to system throughput while still guaranteeing real-time constraints for applications that operate under such restrictions.
Execution services are provided by DNA-OS, a lightweight operating system that can be customized depending on the requirements of the application and the characteristics offered by the hardware resources. DNA-OS has been developed and ported on several architectures (e.g., ARM, MIPS) and, in the context of the EURETILE project, onto the architectures of the QUonG (Intel) and VEP (RISC) platforms. The main objective was to provide a tool-chain able to take as input the application model as well as characteristics of the target architecture (multi-core, multi-tile, 3D network interface) in order to generate all binary codes for all hardware computation resources.
Fault management was addressed with an inclusive approach: fault avoidance techniques are employed by applying mapping optimization strategies; fault tolerance techniques use duplication to preserve results of critical computations in spite of the presence of faults; a fault detection/awareness design approach, LO|FA|MO [11] exploits the network topology to spread the knowledge about the presence of faults; reaction is implemented by migrating the tasks onto healthy nodes.

In this paper, we summarize the major results achieved within the EURETILE project regarding system integration. We report the mechanisms and methodologies used to integrate high-level system specification, operating system level support and platform level support.
The remainder of the paper is organized as follows. In Section 2 we give a brief overview of the state of the art of the topics that we cover in this paper. Section 3 introduces the concepts of our target architecture. Section 4 proposes the EURETILE design flow. Section 5 describes the implemented hardware and simulated execution platforms. In Section 6, the fault management approach is

discussed. In Section 7, various application benchmarks developed using the proposed design flow are described. Experimental integration results are presented in Section 8. In Section 9 the obtained results are discussed together with the limitations of the proposed design flow.

2 State of the art

In this section we describe the background in which our work is settled.
A project similar to EURETILE, the TERAFLUX project [12], addresses many cores devices (teradevices) exploring how to solve issues like the programmability, the design of a manageable architecture, and the reliability of such systems. Our approach is meant to be more comprehensive as we look at the elementary tiles as a hierarchically organized ensemble. This vision and the corresponding application mapping approach fit well with many core device architecture, but more generally with distributed systems as in the HPC domain where computing nodes are also connected off-chip.

Literature shows that the decision of mapping applications to the available computing resources (i.e., cores) can be done completely at either design time or at runtime, or partially at both times. Strategies that fully map at design time usually yield better solutions since it is possible to perform exhaustive optimization calculations. However, in many cases wherein the application is complex and executes in an unpredictable environment (e.g., desktop/general purpose computing) runtime mapping strategies may be more suitable. Finally, if it is possible to make limited assumption on the execution environment, and if the intended application set is well defined, it may be possible to partly compute the mapping at design time, with the remaining mapping decisions being taken at runtime (i.e., hybrid mapping). Since the scope of EURETILE is to explore the common design space of HPC and embedded systems, we have explored pure design time mapping strategies [13] which leverage evolutionary algorithms, and also hybrid mapping approaches [14], which leverage expandable process networks. The considered application mapping approaches can also help in preventing system faults, thus improving reliability. For example, the hybrid mapping strategy tries to minimize the maximum utilization of any core, thus avoiding temperature hotspots, and consequent thermally induced faults. An excellent survey of existing mapping approaches is available [15].

The network topology and the interconnect features deeply affect performance and reliability of distributed systems. In the range of commodity low latency high bandwidth interconnection networks for HPC systems, InfiniBand stands out leaning on a switched fabric network with fat-tree topology and standard QSFP connectors, offering bandwidth and latency that have made it a de-facto commercial standard, ubiquitous in HPC clusters. Conversely, proprietary interconnect solutions from the top players in the HPC field are more varied: IBM BlueGene/Q features a 5-dimensional torus interconnect with electrical links between nodes within a midplane and optical links between midplanes [16]; the Fujitsu Tofu Interconnect 2 is a system interconnect with 6-dimensional torus topology fully integrated in the processor node [17]; TH Express-2 integrated in the Tianhe-2 world's fastest supercomputer, adopts a fat-tree topology [18]; the XC network design by Cray combines the advantages of fat-tree and torus topologies, implementing a direct network with Dragonfly topology to eliminate the need for external top level switches and reducing the number of long-distance optical links needed to reach a given global bandwidth [19].

One of the most common techniques to tolerate faults is to have multiple copies of the same application (“replicas”) execution simultaneously, also known as *N-version programming.* The usual setup requires a process which can distribute input data to the replicas for computing, and another process which is able to combine the output from each replica into the final output. Though N-version programming is a well-studied problem [20] however, when applications have associated timing properties the design of a fault tolerance system using N-version programming is less than clear. This is because one replica may not read and write data from and to its interfaces in lockstep with other replicas, due to normal jitter in the system, such as the processor availability, network latencies, and also due to design diversity between replicas. In the EURETILE approach, when one or more applications may have associated timing or performance constraints, we have proposed a new fault detection and tolerance framework, which does not require any timing resources, and is also lightweight in terms of memory and computational resources used. More details are presented in Section 6.2, and are also available in [21].

Task migration was always considered to be well known traditional solution in cluster computers since late eighties as proposed in [22], [23] and [24]. It is best used for avoiding temperature hot spots, by shifting the execution of a task to another tile. With ongoing advancements in technology and new integrated platforms with parallel architectures being introduced especially in the embedded domain, task migration has regained its importance. In this domain, small operating systems are usually used, hence, task migration is not easily implementable as in cluster computing. As a result, implementation issues still need to be mitigated. Moreover, system architecture impacts the complexity of task migration implementation. In symmetric multiprocessor as in [25], implementation issues are limited due to the fact that both task code and data reside in the same shared memory and migration means just assigning execution to another core. This is feasible since only one copy of the operating system runs, unlike what happens in fully distributed architectures like EURETILE. In such case, there is no shared memory and main memory is private to every core without the ability to access private memory of another, i.e. No Remote Memory Access (NORMA). This makes task migration challenging. There is work in the literature that addresses task migration in Non-Uniform Memory Architecture NUMA architectures as in [26], [27], however, they benefit from a shared memory among cores that may ease inter-process communication. Although some task migration solution are proposed in the literature for NORMA like in [28], [29] and [30], no solution has been proposed to the best of our knowledge that collects characteristics like transparency to applications, portability to different core architecture and being able to be plugged in automatic generation tool as explained in [31]. Moreover, our solution can be used with a small embedded OS without the need of extra features like dynamic loading and runs on both RISC architecture simulator and on real hardware architecture (a HPC cluster).

3 Target platform

In EURETILE, the architecture is described in terms of a hierarchically organized layout of a many-tile platform, a generalization of the well-known tile-based multiprocessor model [32], which has been successfully applied in academia and industry.

Currently we devise at least 3 levels of hierarchy: at the first (bottom) level lies the elementary *tile*, which contains a distributed network processor (DNP), multiple general-purpose cores and local memories. All cores in a tile may have access to a shared memory. Depending on the target application, the tiles may also include specialized hardware accelerators, like DSPs, ASIPs, FPGAs, or (GP)GPUs. The DNP is responsible for inter-tile communication and provides the interface to the network. Upper levels of the hierarchy are built according to a distributed memory paradigm. At the second level, communication patterns between processes are established and mapped on n-dimensional arrays of hardware tiles, that form an additional entity called “cluster”, which can be controlled autonomously.

At the third level, several of mostly independent (but possibly communicating) applications, will be concurrently executed on clusters of tiles. Note that this layer is dynamic, meaning that applications may dynamically enter or exit the system.

This hierarchical layout lays the groundwork for turning distributed memory and control applications into an inherently fault tolerant system. Clustering allows segregating some resources in order to design and manage fault containment regions but this same idea can be further pushed into creating an upper layer network of supervising tiles tasked with monitoring the health of other clusters, eventually allowing for a no-single-point-of-failure fault management system. Also, spare cores and tiles can be allocated at design time so that the runtime-manager can efficiently react to faults by migrating processes assigned to faulty cores to spare ones (more on that in Section 6.3).

Figure 1 gives a representation of the hierarchy levels of the EURETILE target architecture.

In order to match the requirements of the two reference domains we implemented the presented architecture in two flavours, each one characterized by its own kind of elementary tile, as follows:

- **QUonG**, an hardware platform based on x86 multi-core processors connected by a custom 3D first-neighbour toroidal interconnect called APEnet+ which is the DNP implementation FPGA;
- the **Virtual EURETILE Platform** (VEP), a scalable many-tile simulator whose tile includes a processor (IRISC) more representative of embedded systems domain; the VEP tiles are also connected by the DNP in a 3D mesh.

Figure 2 shows the two elementary tiles, the one used in the QUonG platform and the one used in the VEP platform. More details on the implementation of the two platforms are given in Section 5.

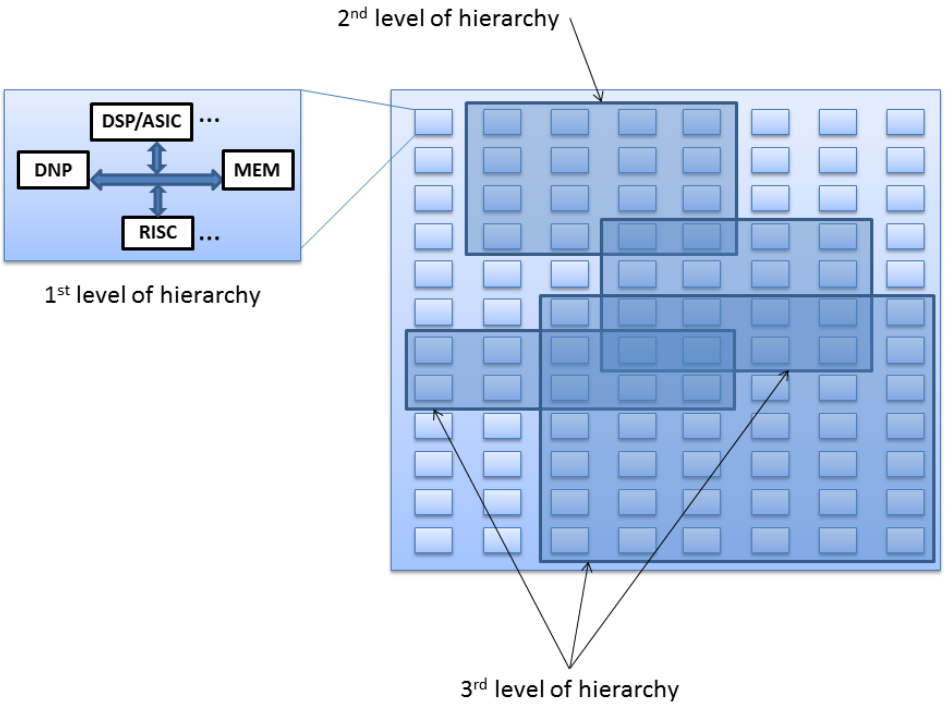

Figure 1-Simplified view of the EURETILE platforms used for experiments on innovations to many-tile hardware and many-process software of future fault-tolerant Embedded Systems and HPC platforms. In both cases the inter-tile communication is supported by a Distributed Network Processor (DNP) that connects nodes in a 3D toroidal mesh (not shown here).

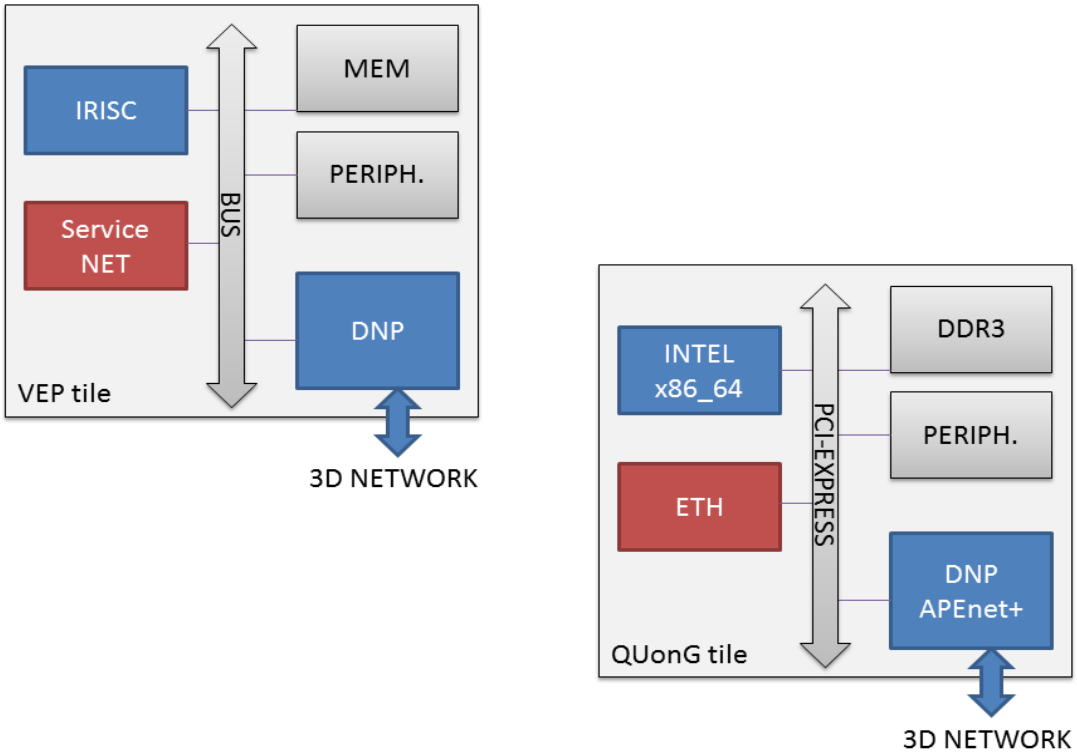

Figure 2: The EURETILE elementary tiles for the the VEP and QUonG platform respectively.

4 Overview of the EURETILE design flow

The EURETILE design flow maps a set of dynamic applications that are specified as a network of processes onto a many-tile platform in a number of steps, as illustrated in figure 3.
Each application is specified, according to the DAL programming model, as an expandable process network (EPN) [33]. An EPN consists of a set of autonomous processes that communicate through point-to-point FIFO channels. By limiting the access to FIFO channels to destructive blocking reads and non-blocking writes, data races, non-determinism or the need for strict synchronization are

avoided. Furthermore, some processes may have associated with it a structure specified as a process network. In other words, it is possible to abstract a process network into a single process. The functionality of each process is implemented in the C/C++ programming language. By specifying each application as an EPN, the degree of parallelism of the application can be selected automatically during design space exploration, minimizing scheduling and interprocess communication overheads [14]. Finally, each application may have its own performance requirements that must be met independent of the others.

Interactions between applications are represented as a finite state machine (FSM) with each state representing an *execution scenario*, i.e., a (sub)set of applications running concurrently, see Figure 4. We call a scenario a certain system state with a predefined set of running or paused applications. As an example, a mobile phone executing only telephony defines one (use-) scenario, whereas the same device executing telephony and wireless radio defines another scenario. Transitions between scenarios are triggered by events generated by either running applications or the operating system. Typically for embedded systems, the number of possible scenarios is restricted and the scenarios are known at design time.

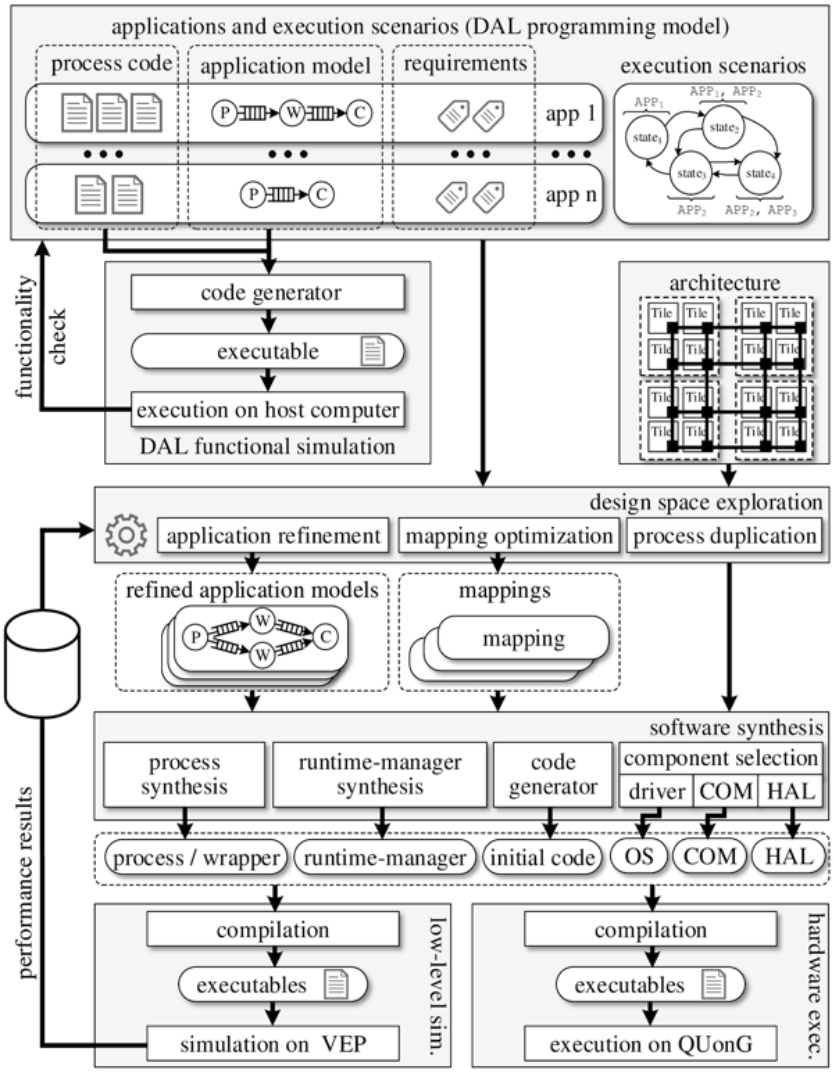

Figure 3 - *EURETILE design flow to map a set of dynamic applications onto a many-tile platform.*

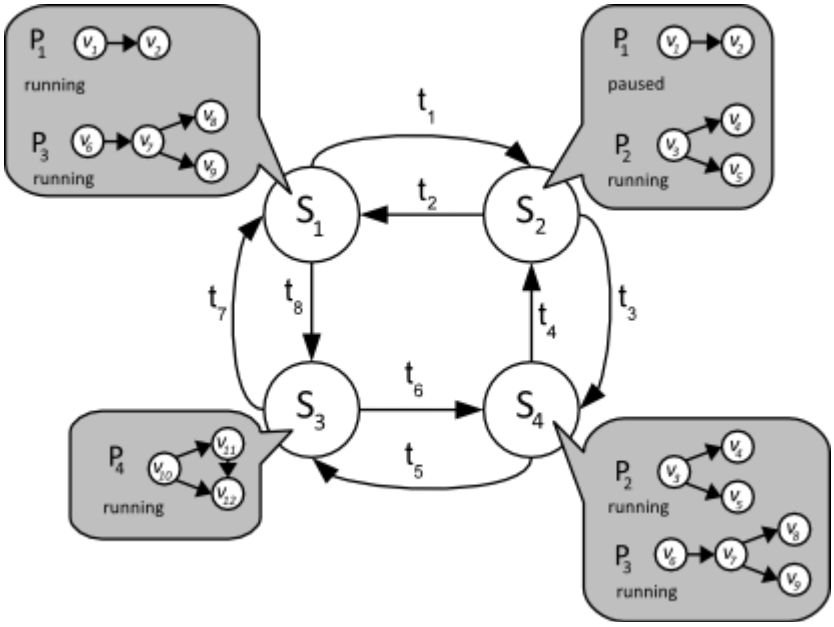

Figure 4 - *A FSM with four execution scenarios. Each state uniquely determines the state (i.e., running or dormant) of each application, and within each application, the state of each process.*

Each application is individually verified by the DAL functional simulator. Specifically, the functional check verifies the structural correctness of the process network (e.g., is each process connected to the correct parents and children, does the channel have sufficient buffering capacity, races). The DAL functional simulator executes each application consistent with the specification (e.g., with the required parallelism) and reports runtime functional profiling data, e.g., number of times a process has fired, the number, and the sequence of data read and write operation by each process, along with the amount of data read or written. For extracting timing sensitive profiling data, such as runtimes of processes, the system is simulated on the Virtual EURETILE Platform (VEP).

During design space exploration, concurrent processes are assigned to cores and the mapping is iteratively refined using the performance results from the VEP until the performance requirements are fulfilled. Alternatively, one may use formal optimization approaches, such as ILP, where the objective may be to minimize the maximum core utilization in order to limit the maximum core temperature. A more detailed discussion is available from [34].

During software synthesis, a four-layer software stack is generated that consists of the application layer, the runtime-system, the OS, and the hardware abstraction layer (HAL), as pictured in Figure 5. The employed tool chain first transforms the DAL processes into low-level threads that can be executed on top of DNA-OS [35]. DNA-OS is a lightweight embedded OS that is used in the proposed software stack. Afterwards, the tool chain generates one instance of the software stack for each tile.

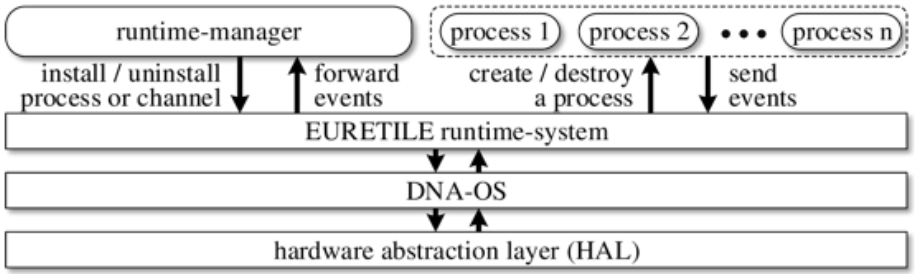

Figure 5 - Software stack generated by the EURETILE design flow.

4.1 Programming model: applications and execution scenarios

Next, the high-level specification of the proposed system to describe applications and execution scenarios is outlined. The basic idea of the programming model is to design each application individually as an EPN [33] and to specify the interactions between the applications as a finite state machine [34]. With this programming model, complex and dynamic interactions between applications can be modeled.

4.1.1 Application specification

The EPN semantics extends the Kahn process network one (KPN) [36] by abstracting several possible granularities in a single specification. More in detail, an EPN-written application has a top-level process network that can be refined by hierarchically replacing individual processes by their structural specification. This latter describes application functionality as another process network thus enabling the automatic exploration of task, data, and pipeline parallelism by two refinement strategies, namely replication and unfolding. Replicating processes increases data parallelism and structural unfolding of a process increases the task and pipeline parallelism by hierarchically instantiating more processes in the process network. In addition, the functionality of each process is also specified in C/C++. The proposed application-programming interface (API) is composed of three procedures. The **init** procedure is executed once when the application is started. Afterwards, the execution of a process is split into individual executions of the **fire** procedure, which is repeatedly invoked. Finally, the **finish** procedure is called before the application is stopped. Each process can read from its input channels and write to its output channels by calling the high-level **read** and **write** procedures. Moreover, each process has the ability to request a scenario change by calling the **send_event** procedure. In addition, the topology of the application, i.e., the connections between processes by FIFOs, is specified in XML format.

4.1.2 Execution scenarios

Dynamic behaviour of the workload is captured by a set of execution scenarios forming a finite state machine (FSM). Each state represents a set of concurrently running or paused applications and each state transition corresponds to an application start, stop, pause, or resume request. Starting an application involves installation of all its processes and FIFO channels. After executing the init procedure of all processes of the application once, the fire procedures are iteratively executed by the scheduler. On the other hand, when an application is stopped, the fire procedure of all processes is aborted and the finish procedure is executed once. Finally, all processes and all FIFO channels are removed.

The number of execution scenarios depends on application complexity and the number of input variables influencing it. Therefore, if every change in the input to the application causes a significant change in functionality (e.g., a new process must be started, a process must be terminated), the number of execution scenarios will be large. The number of possible execution scenarios moreover depends by the number of modes the application can execute (e.g., one or more degraded modes), and it is not unusual to expect several dozen execution scenarios for a typical application. It may be possible to exhaustively enumerate all possible execution scenarios associated with the application if the runtime behaviour of the application is influenced by only a limited number of signals (or inputs).

4.2 Hybrid mapping strategy

The mapping decides process distribution onto the architecture. Since conventional mapping strategies which calculate a single mapping are no longer capable of efficiently utilizing the hardware if the functionality of the system can change at runtime, a hybrid design-time/runtime mapping strategy is elaborated in EURETILE [34] illustrated in Figure 6. During design space exploration, an optimal mapping is calculated for each application and scenario where the application is running, whereby the virtual representation of the architecture is used as target. Then, a runtime-manager controls the dynamic behaviour of the system. Whenever an application is started or stopped, it first selects the appropriate mapping (onto the virtual representation) and then maps the virtual representation onto the physical architecture. This enables the runtime-manager to react to faults without recalculating the mapping simply by adjusting the binding of the virtual representation onto the physical architecture. In the following, we will detail the design time analysis and the runtime management.

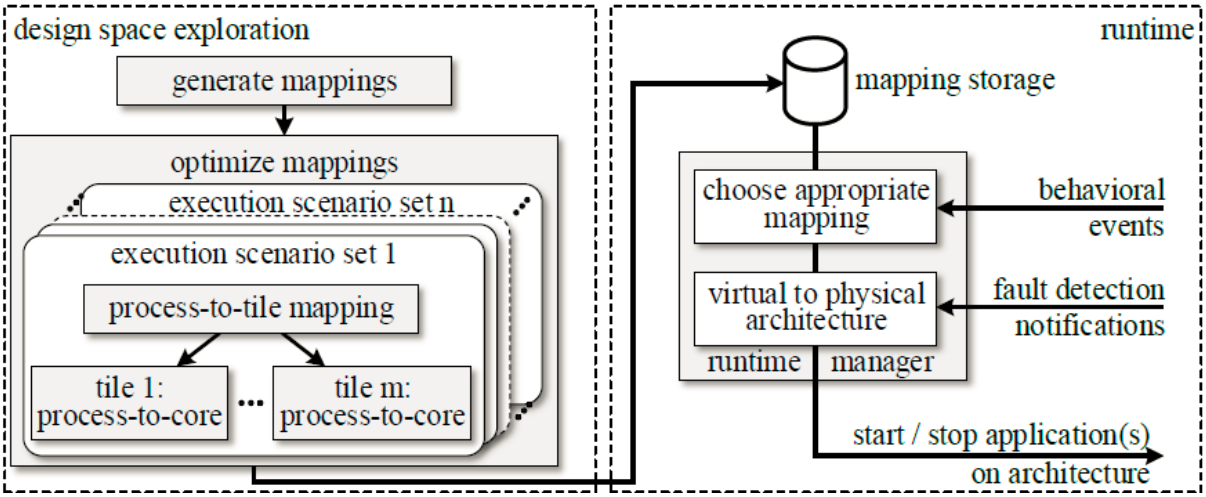

Figure 6 - Hybrid mapping environment.

4.2.1 Design time: analysis and optimization

At design time, an optimal mapping is calculated for each application/scenario pair where the application is running [13]. Therefore, the output of the design space exploration is a collection of optimal mappings with exactly one valid mapping for an application/scenario pair. To minimize the reconfiguration overhead, an application has the same mapping in all connected execution scenarios so that a running application is not affected by the start or stop of another application.

More in detail, a two-step procedure is applied during design space exploration to efficiently calculate the mappings. First, it is calculated which application/scenario pairs must use the same mapping so that no process migration is required. We do that by calculating for each application separately the maximally connected components of a subgraph which only contains the scenarios where the application is running. At the end of this step, one mapping is allocated for each subgraph component. Second, the previously allocated mappings are optimized so that the objective function is minimized and additional architectural constraints (e.g., processor utilization, link bandwidth, chip temperature) are fulfilled. The objective function depends on intended use and may include many objectives.

However, multi-objective meta-heuristics successfully applied to solve multi-tile systems mapping problems are no longer effective for many-tile systems due to the scale of the investigated problem, i.e., said approaches do not scale well given a large design space within which the solution must be found. For EURETILE, there are several hundred cores distributed in several dozen tiles, several applications, each with hundreds of processes, and several dozen execution scenarios for each application. Thus, the EURETILE design space may be too large to be explored by conventional multi-objective optimization.
Therefore, a multi-objective mapping optimization technique is used that overcomes this shortcoming by decomposing the mapping problem into independent subproblems. As illustrated in Figure 6, two different problem decompositions are considered: the execution scenarios are decoupled into independent sets and an architecture-based decomposition is applied that first assigns processes to tiles. Afterwards, and for each tile separately, the processes are assigned to the cores. Additional hierarchical layers might be added if the considered platform is more complex.

4.2.2 Runtime management

The task of the runtime-manager is to send commands to the runtime-system so that system execution semantics is ensured. To this end, the runtime-manager receives and processes behavioural events and fault detection notifications. An event is sent by an application and triggers a scenario change. A fault detection notification is sent by LO|FA|MO (see Section 6.3.1) if a hardware fault has occurred. The latter only changes the binding of the virtual representation onto the physical architecture, but not the mapping of the applications onto the virtual representation.

Consequently, the runtime-manager consists of two components. The first component is responsible of handling behavioural events and ensures the execution semantics. It is just aware of the virtual representation of the architecture. The second component processes the fault events and redirects the commands to the corresponding physical network. If the runtime-manager receives a fault event, it migrates the processes that are assigned to the faulty core (tile) to a spare core (tile) and changes the binding of the virtual representation onto the physical architecture by reconfiguring the second component to redirect the commands to the new target.

4.3 Generation of the executable

We have defined a SW tool-chain that is able to accept a DAL model as input and generates binary code that runs on different architectures: the x86 HPC cluster, QUonG, and the IRISC-based VEP simulator. We can split this tool-chain into two parts: the front-end turns a DAL model into a multithreaded C application; the back-end compiles and links the code on top of DNA-OS, generating the binary code for the targeted processors and architectures. Code generation flow is shown in Figure 7.

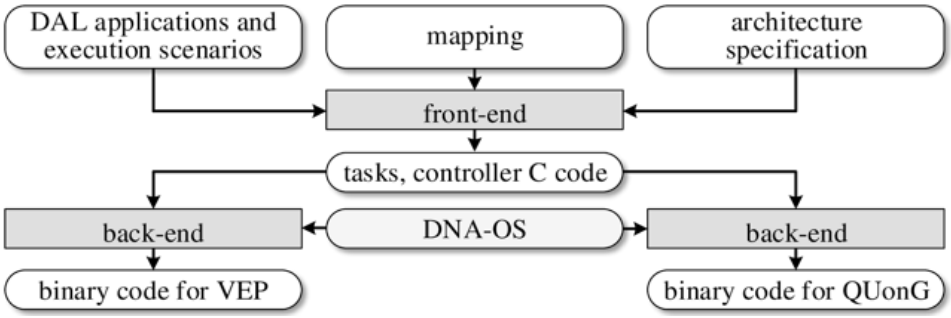

Figure 7 - Overview of the software synthesis tool-chain as elaborated in the EURETILE design flow.

4.3.1 Software synthesis flow and DNA-OS

The tool-chain made for SW synthesis flow is split into a front-end and a back-end, providing application(s) code along with DNA-OS code ready to be compiled and linked in order to have executable files targeted for the corresponding architecture. DNA-OS is an embedded operating system whose architecture is based on exokernel [37]; it is coded in C99 and is characterized by its small memory footprint, simplicity, small overhead and its layered and modularized structure.

On one side, the input to the front-end part is composed of application(s) code along with the XML files that describe the FSM of application(s), the mapping of tasks onto tiles and the platform. The application defines the necessary DAL procedures: app_init, app_fire and app_finish. Output of the front-end is C code for DAL controller processes which manage application(s) flow according to the aforementioned FSM by either starting, stopping, pausing or resuming the running of tasks on tiles.

On the other side, the back-end - different target architectures require specific tool-chains, as is the case of the IRISC-based one for the embedded domain platform and the x86 one for the HPC platform - takes the C code of both generated controllers and application-required components of DNA-OS as input, compiling and linking it into target binaries.

4.3.2 Hierarchical runtime manager

Dynamic system behaviour is captured by a set of scenarios formally described as a finite state machine (FSM). A hierarchical control mechanism (Figure 8) is proposed which follows the architecture structure and is represented in DAL as a process network including three different types of controllers:

- slave controllers are responsible just for the activities inside a tile,
- interlayer controllers are responsible for the activities inside a cluster,
- the master controller is responsible for the complete system and processes all events that cannot be handled by any other controllers.

The controller explicates for each state tasks that are running on each processing unit. When all tasks are created, those that are not supposed to be executed in the initial state are paused. When entering a new state, some are resumed and some are paused to fit with the state task execution specification.

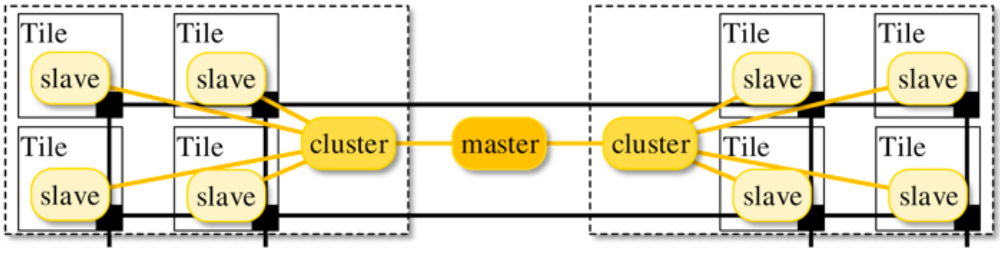

Figure 8 - Illustration of a hierarchically organized runtime manager.

4.3.3 Specificities of generation for QUonG hardware platform

DNA-OS was first ported onto the x86 QUonG hardware platform. A new driver was added to DNA-OS driver layer to support the platform communication medium, i.e. the Peripheral Component

Interconnect express (PCIe). This was necessary to access hardware components such as the inter-tile interconnect card (APENet+) of the QUonG tile, which connects to the on-tile motherboard over a PCIe bus.

4.3.4 Specificities of generation for VEP simulated platform

The IRISC processor is VEP's core processing unit, with a software tool-chain based on the CoSy Compiler system [38] providing a compiler and basic C libraries retargeted to support DNA-OS as host OS. This tool-chain was integrated into DNA-OS code generation system to allow generation, compilation and linking of hardware-dependent software modules and DAL applications for the VEP.

5 EURETILE Execution platforms

In this section we describe the two representative target platforms: QUonG, a many-tile fault-aware hardware platform - based on the custom APEnet+ network card and x86 multi-core processors - and the Virtual EURETILE Platform (VEP), a scalable many-tile simulator using the same interconnect but including a processor (IRISC) more closely representing those prevalent in embedded systems.

5.1 Hardware Experimental Platform (QUonG)

QUonG is a comprehensive initiative aiming at providing a hybrid, GPU-accelerated x86_64 cluster with a 3D toroidal mesh topology, able to scale up to $10^4/10^5$ nodes. A QUonG cluster has been assembled and put into service, with 16 identical tiles (nodes) arranged into a 4x4 network mesh served by one 6-links APEnet+ card per tile. Each tile provides 2 Xeon Westmere-family CPUs, 2 M2075 NVIDIA GPUs, and 48GiB of memory; QUonG tiles also partake in two supporting networks, InfiniBand– and GbE-based. Applications can use either 1- the EURETILE software tool-chain (DAL+DNA-OS for QUonG), or, 2- a GNU/Linux-based environment, offering standard high-level communication semantics like Message Passing Interface (MPI).
A disparate set of scientific codes has been run on QUonG, in order to stress-test the installation, among which the Distributed Polychronous Spiking Neural Network simulator. In parallel, joint work led to a quite stable release of DNA-OS on x86_64 architecture whose highlights are the support of APEnet+ and Intel GbE Ethernet cards as channels for remote communication among DAL processes.

5.1.1 The APEnet+ interconnection system

APEnet+ is a point-to-point, low-latency network controller (Figure 9) for a 3D-toroidal topology integrated in a PCIe Gen2 board based on an Altera Stratix IV FPGA. It is the building block for the QUonG hybrid CPU/GPU HPC cluster inside INFN [7]. Directly connected to the board embedded transceivers, 6 QSFP+ modules are available, 4 placed onto the main board and 2 on a small piggy back card, thus resulting in a 2-slots-wide card mechanically compatible with x16 cards. Each Altera embedded transceiver is capable of up to 8.5 Gbps fully bidirectional data rate and a single remote data link is built up by bonding 4 transceivers into a link operating at up to 34 Gbps.

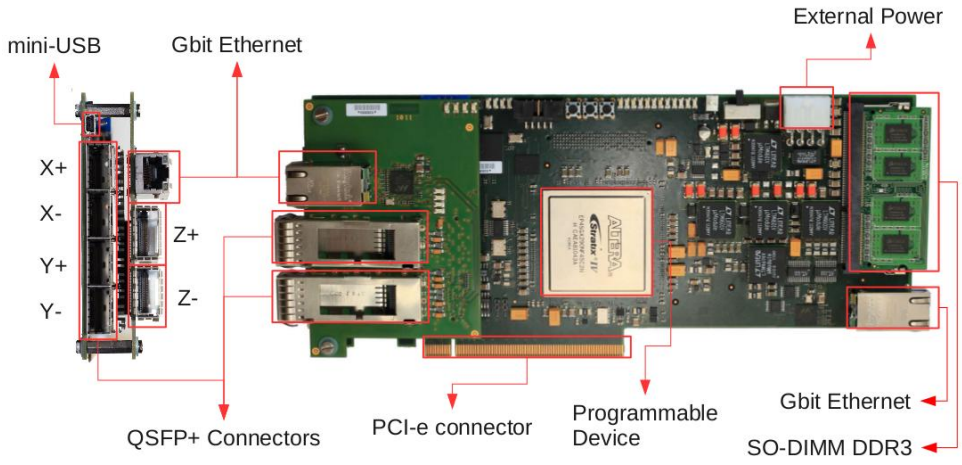

Figure 9 - APEnet+ board: 2011 release.

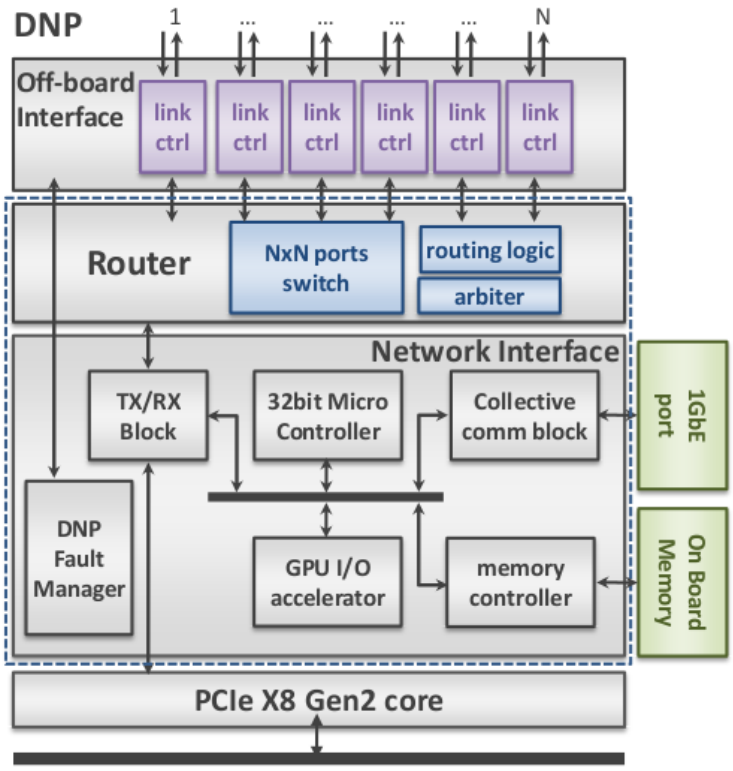

Figure 10 - Main logic blocks of the FPGA architecture.

The DNP (its outline is shown in Figure 10) is the core of the APEnet+ architecture; it acts as an offloading engine for the computing node, performing inter-node data transfers. The Torus Link block manages the data flow by encapsulating packets into a light, low-level word stuffing protocol able to detect transmission errors via CRC. The APElink [8] data transmission system sustains a channel bandwidth up to 2.8 GB/s; its custom logic [39] enables a global APElink bandwidth of 3.0GB/s. The Router component is responsible for data routing and dispatching, dynamically interconnecting the ports of the cross-bar switch; it is able to simultaneously handle 7 flows @3.0 GB/s. The Network Interface is the packet injection/processing logic; it manages data flow to and from either Host or GPU memory and provides hardware support for the Remote Direct Memory Access (RDMA) protocol on the receive side, allowing remote data transfer over the network without remote node CPU involvement. An integrated microcontroller provided by the FPGA platform allows for straightforward implementation of RDMA semantics. The Network Interface is also able to directly access the memory of Fermi- and Kepler-class NVIDIA GPUs implementing GPUDirect V2 (peer-to-peer) and GPUDirect RDMA capabilities [40]. A dedicated component (DNP Fault Manager) provides fault awareness capabilities as described in Section 6.3.1.

The zero-copy, offloaded networking enforced by RDMA requires, to work on the receive side, the NIC be able to directly read/write the data to/from the destination buffer which in turn requires virtual to physical address translation. APEnet+ has two independent implementations of this task: one is solely based on a microcontroller firmware; the other is hardware-assisted by a Translation Lookaside Buffer (TLB) [41] that accelerates buffer search (BSRC) and address translation (V2P).

Peak bandwidth of a buffer transfer using the TLB reaches 2500 MB/s for 128 KB message size; a transfer to GPU memory reaches 2500 MB/s bandwidth in the range of message size 64 KB ÷ 2 MB. In Figure 11 are plotted some results of synthetic tests that were performed between two APEnet+ ards equipping two servers populated with Ivy Bridge-class Xeon CPUs and Kepler-class GPUs.

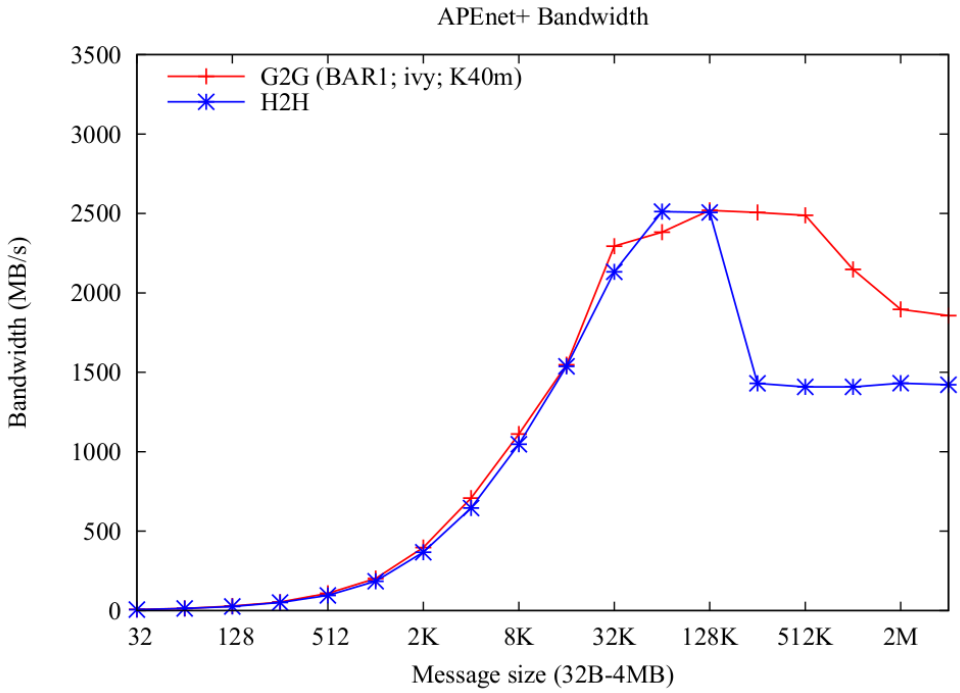

Figure 11 - APEnet+ GPU-to-GPU and Host-to-Host bandwidth with RX hardware accelerator.

On top of the RDMA protocol, the APEnet+ software stack provides an MPI layer, this latter being the de-facto standard communication API for parallel computing platforms. Our implementation is based on the portable, open source OpenMPI one.
Before being driven by DNA-OS in the EURETILE tool-chain, the APENet+ interconnect was tested on a GNU/Linux OS driver and a hybrid MPI/RDMA version of the neuro-synaptic simulator (DPSNN). On the QUonG platform, this code managed simulations of up to 209*10^6 synapses over 1 to 64 MPI processes running on 1 to 16 QUonG nodes. DNA-OS provides a deterministic behavior as shown in Section 8 and shows that an embedded OS could be used in the HPC domain.
Measurements from this kind of benchmark suffer from the overhead due to the presence of the MPI library layers adapted to the APEnet+ peculiarities and the presence of a bulky MPI runtime. Results coming from an RDMA/MPI hybrid code are more promising at the cost of using APEnet+ low level, non-portable RDMA primitives. A real gain, both in terms of performance and usability, therefore comes from introduction of DNA-OS and DAL.

5.2 Simulated Experimental Platform (VEP)

The Virtual EURETILE Platform (VEP) is a SystemC [42] based simulator modeling an embedded version of the EURETILE architecture. Full-system simulation plays an essential role in early design exploration and debugging, system software bringup and behavioural features assessment. Due to the massive parallelism of the EURETILE architecture, requirements for a full-system simulator are:

- Provide means to facilitate debugging and profiling of concurrent software.
- Provide fast and scalable simulation to experiment with a possibly large number of tiles/cores.
- Satisfy different use cases like software debugging and network scalability tests.
- Allow fault injection to experiment with fault-awareness mechanisms.

The VEP builds upon abstract and parallel simulation technologies and provides a set of advanced multicore debugging tools which are described next.

5.2.1 VEP characteristics and architecture

The VEP allows simulating system configurations composed of several tiles arranged in a 3D grid and connected in a 3D toroidal topology (see Supplementary Video 1). VEP tiles contain a small form factor RISC processor tailored for embedded applications, namely the IRISC. The IRISC is created with Synopsys Processor Designer (SPD) in the LISA language [43] and thus allows easy customizations for deriving new specialized ASIPs. A VEP tile also consists of transaction level models (TLM2) [42] of memory blocks, a bus, a serial I/O interface, timers, a real-time clock, an interrupt controller and the DNP. A service network interface used for fault monitoring is also part of a tile. VEP simulation models are augmented with debugging, tracing and fault injection capabilities. For the latter, users can specify faults in different components and at precise points in time. Supported faults include processor and DNP breakdowns, severed DNP connections, isolation of tile groups and random bit-flips, among others. The VEP was created in a flexible way to allow user specification of the number of tiles, memory address maps and processor frequencies, among others, without

recompilation. Various optimizations which are commonplace in full-system simulators, such as temporal decoupling and direct memory interfaces (DMI) are also supported.

Abstraction levels and processor model variants

The processor model in every tile incurs the highest cost in terms of VEP simulation speed and memory footprint. The IRISC instruction-set simulator (ISS) was created at different levels of detail to allow trading off speed and accuracy. First, a cycle-accurate (CA) and an instruction-accurate (IA) models based on just-in-time cache-compiled (JIT-CC) technology [44] were developed in SPD to facilitate two main use cases of the VEP: accurate performance analysis and software development. These SPD models can only be compiled in 32-bit mode and the memory footprint of a single model instance is around 20Mb when the simulator is running. Thus, using them limits the VEP to 200 tiles or else the 4Gb memory wall is hit (in a single simulation host). To overcome this issue, an optimized variant of the IA ISS was created, namely the IRISC IAP, which is a simplified model without debug features but compatible with 64-bit compilation. The IRISC IAP allows instantiating up to 1000 VEP tiles in a single host. Finally, a fourth IRISC ISS, namely the DBT-IA, was created to further increase simulation speed. The IRISC DBT-IA is based on a dynamic binary translation (DBT) engine [45] that, in our implementation, was measured to provide an execution speed ~170x, ~30x and ~7x faster than the CA, IA and IAP models, respectively. This speed comes at the cost of increased memory footprint due to the internal DBT engine and a hard limit of 32 instantiable VEP tiles per host.
The VEP can also run as an abstract simulation platform replacing the ISS with a host-compiled simulator. In this form, every tile is provided with the Abstract Execution Device (AED), a SystemC module that runs target code compiled for the host machine. The AED talks with the DNP and on-tile devices through an interface identical to the IRISC core's and provides features like software timing annotation and limited visibility of tile-scope global variables. AED memory footprint is the lightest one allowing up to 20000 VEP tiles per host. Table I compares the four IRISC models and the AED.

VEP use cases

The VEP allows targeting different use cases. The CA version is good for accurately assessing the performance of applications but the speed of IA models suits development tasks better. The IRISC IA is commonly used for software programming and debugging of mid-size systems (32 to 200 tiles). The IRISC IAP is used for testing in large systems and DNP network stress tests (from 200 to approx. 1000 tiles). The IRISC DBT-IA is used for development and debugging of small systems running computationally expensive applications. In AED mode, very large systems can be simulated, thus the AED is better suited as a DNP-centric simulator to test network scalability and fault awareness.

Table I - Summary of IRISC models and VEP use cases

	IRISC CA	**IRISC IA**	**IRISC IAP**	**IRISC DBT-IA**	**AED**
Stand-alone MIPS	~1	~6	~25	~160	~3000
Number of Tiles	~200	~200	~1000	~32	~20000
Debug Support	Yes	Yes	No	Yes	No
VEP Use Case	Performance analysis	Driver and OS development and debugging	System scalability testing	Application development and debugging	Network scalability testing

5.2.2 Parallel Simulation Technologies

Although the VEP is compatible with the OSCI SystemC reference kernel implementation [46], the performance degradation of large system topologies and the wide availability of multi-core host simulation machines called for the adoption of parallel simulation technologies. Two parallel SystemC kernels are developed together with the VEP, namely parSC and SCope. On one hand, parSC [47] parallelizes SystemC simulations by distributing events, such as CPU clock ticks or interrupts, that happen at the same point in time (i.e., the same SystemC delta cycle) among different threads. Figure 12 shows the structure of parSC, highlighting the traditional *elaboration*, *evaluation*, *update* and *notify* simulation phases of SystemC. This strategy is convenient for detailed simulation models (e.g., CA). For applying it to the VEP, parSC was extended to facilitate race-free operation [9]. This extension allows configuring the parSC scheduler so that all SystemC processes of a tile are scheduled on the

same OS thread. Compared to the OSCI kernel, the VEP with parSC achieves a speed-up of more than **2x** on a quad-core simulation host.
On the other hand, SCope [48] is a SystemC kernel that exploits coarse-grained parallelism in simulations and achieves high speedups when the simulator is composed of faster models (e.g., TLM2 models). SCope groups several simulation modules (e.g., the components of a tile) and then assigns the groups to different host threads that execute all four SystemC simulation phases. The groups then operate with a limited time decoupling between threads, synchronizing after the *notify* phase, as shown in Figure 13. This decreases the amount of synchronizations increasing overall performance. On a quad-core host, SCope gives a speedup of around **4x** compared to the OSCI kernel (see Supplementary Video 1). Dynamic load balancing has also been implemented for SCope so as to enable moving groups from thread to thread at the SystemC module level. Load balancing has proven beneficial on the VEP with unbalanced application scenarios, bringing an additional speedup of 1.35x.

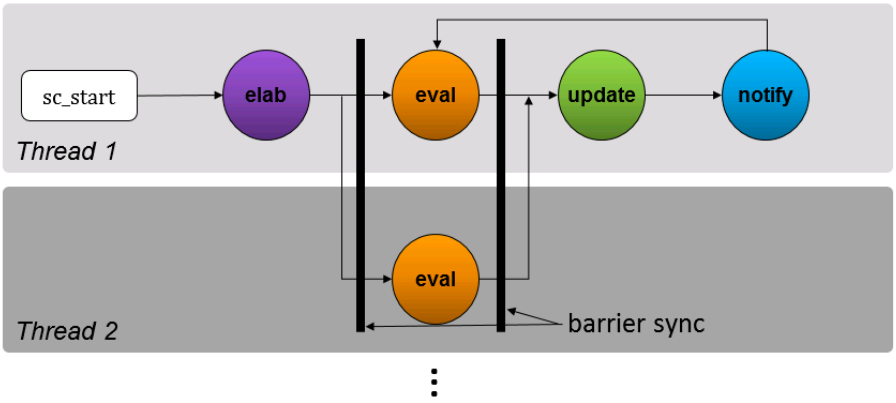

Figure 12-Simulation loop of the parSC SystemC kernel

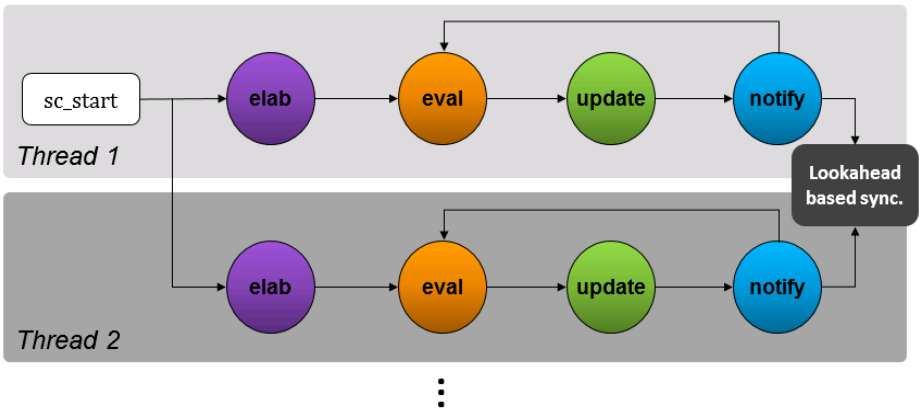

Figure 13 - Simulation loop of the SCope SystemC kernel

5.2.3 VEP Multicore Debugging Tools

Software debugging for parallel systems like EURETILE is one of the most time consuming tasks of the overall design cycle. Although a simulated platform like the VEP provides full visibility and controllability of the system, efficient debug needs tools crafted ad-hoc. Besides the traditional GNU debugger (GDB), the VEP simulator is thus complemented with three tools that address key issues like abstraction, retargetability, scalability and convergence of information from different data sources.
The **Whole-system Debugger** (WSDB) is an interactive source debugger based on a scalable component-based framework [49] developed for the EURETILE system. It can be used for traditional source debugging tasks, e.g., placing breakpoints and examining memory, with a unified interface to interact with all tiles. For instance, it allows iterating on groups of cores and DAL processes for inspection and control. WSDB can be easily extended via the TCL language to perform advanced debug tasks (e.g., setting user-defined callbacks on OS-level events or network packets).
HW/SW issues can also be debugged in the VEP through the **System-Wide Assertions** (SWAT) framework [50]. SWAT allows true full-system HW/SW assertion-based verification by monitoring behavioural system properties, given in a special language for linear temporal logic (LTL) [51], that in turn infer runtime inspection and correlation of software variables and hardware registers/signals spread over different system components and software modules. SWAT is useful for debugging both sequential and concurrency bugs.
Finally, a concurrency exploration framework called **ConcuRex** has been developed based on dynamic analysis of event ordering constraints [52]. ConcuRex monitors, displays and identifies conflicting concurrent interactions among system components, which are in turn used by the

framework to reproduce a previously observed behaviour or intentionally trigger buggy states. This tool also enables precise control of the execution of all individual software tasks and components on the target (at the level of monitored events) that is used by bug finding algorithms to manipulate the system behaviour (e.g., to automatically find atomicity or order violations). With WSDB, SWAT and ConcuRex, the VEP development environment allows the user to tradeoff the generality of a debugging task and the manual effort involved to achieve it.

5.2.4 DNP VEP model

Similarly to the QUonG platform, VEP computing tiles are connected in a 3D mesh via a SystemC TLM2 model of the DNP which simulates data transfers and routing functionality and provides timing annotations based on APEnet+ hardware latencies. The VEP DNP model allowed architectural features validation and scalability testing. Moreover, it was used to prove the validity of the LO|FA|MO approach to systemic fault awareness (see Sec. 6.3.1), before its hardware implementation was available.
A DNP RDMA API allows application level programming but a more user-friendly interface is the PRESTO library that, on top of the DNP RDMA API, provides MPI-like send/receive primitives.
The DNP model is lightweight and frugal in terms of memory consumption and execution time, so it does not hamper the scalability and performance of the VEP.

6 Fault Management

Integrated circuit technology scaling is increasingly making processors more vulnerable to faults. For instance, modern processors are more likely to experience single-event upsets, which were uncommon only a few years ago, and smaller transistors are responsible for higher power densities, which in turn cause temperature hotspots. In EURETILE, a high system dependability is achieved by combining three fault management strategies, i.e., fault avoidance, fault tolerance, and fault reactivity. While the former two strategies are implemented at system-level by elaborating properties of the programming model, fault reactivity involves programming model, OS, and hardware. In the following, we will describe the key concepts of these fault management strategies.

6.1 Fault avoidance

Temperature related reliability issues are avoided by applying thermal-aware optimization strategies [53], [54]. During design space exploration, system designs that do not conform to peak temperature requirements are ruled out using formal thermal analysis methods. The methods are based on real-time calculus that is widely used to analyze and optimize real-time systems.

6.2 Fault tolerance

Applications with stringent reliability requirements are duplicated during design space exploration [21]. As there already exist several relatively mature approaches for detecting value faults, we specifically investigated the harder problem of detecting timing faults. A timing fault is observed when an application computes an output, but not within the expected timing constraints. That is, either the output is computed too early, or it is computed too late. Notice that timing fault does not merely imply a deadline miss, but also covers the case when the node may be writing data at a higher than expected rate, due to software or hardware faults. Thus, the faults may originate either in the hardware, or in the software, or in a combination of both.

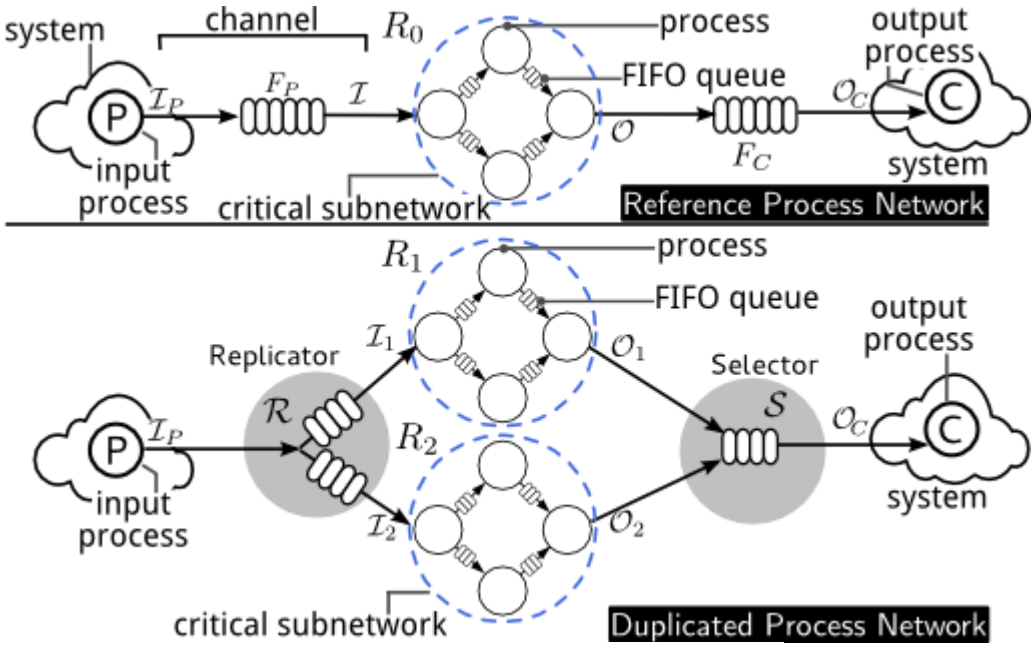

Figure 14: Setup for the Fault Tolerance solution.

The considered framework is shown in Figure 14, wherein the reference (i.e., the reference process network) application consists of a *critical subnetwork* which must be made fault tolerant. The critical subnetwork may be serviced by one or more sources through a FIFO buffered channel. Furthermore, the critical subnetwork may write output to one or more FIFO buffered output channels.
For fault tolerance, we consider a setup consisting of at least two replicas of the critical subnetwork executing concurrently, see Figure 14 (bottom). Replicas have the necessary design diversity in order to avoid common mode faults. Each individual replica has similar timing and performance features as the critical subnetwork in the reference process network. Without loss of generality, we assume that only the replicas (in general, the critical subnetwork) can experience faults. This restriction can be relaxed by replicating other components as well, according to the approach described here.
A special *replicator* process provides input data to each replica, whereas a special *selector* process merges the outputs from each replica into a single, final output. The replicator process is designed to block only on its input channel, I_p and performs a non-blocking write on its output channels. On the other hand, the selector process is designed to block only on its output channel O_c but performs non-blocking reads from both inputs. It is also assumed that both replicas are *fail-silent*, i.e. replicas either compute the correct output or do not write any data to any of their output channels.
Various hardware and software solutions that ensure fail-silence of the nodes are already available and also widely used [55], [56].
The FIFO buffers at the output of the replicator process are sized so as to never be full under no-fault conditions. Estimation of minimum FIFO buffer size can be easily done using network and real time calculus based on the knowledge of rates at which the producer writes tokens and the rates at which the replicas read them. Therefore, if the replicator fails to write a duplicated data packet into any of its output FIFO buffer, then the corresponding replica can be considered to have suffered a (timing) fault.

The selector process design aids in transparent fault detection. Under no-fault conditions, this process receives each data packet duplicated by the replicas. Since each of them can individually meet all timing requirements expected by the output process (see Figure 14), the selector can safely retain the first token of each duplicate pair and discard the late arriving one, resulting in transparent and efficient fault tolerance. Furthermore, it is possible to detect timing faults at the selector process by noticing that since both replicas have similar performance (necessary in order to meet the timing constraints for the final consumer, the *output process*), the number of tokens produced by either replica in *any* time interval must not differ from the other replicator by more than a statically computed threshold. Should this difference be exceeded at any point in time, it is an indicator of a fault suffered by a replica. Under the property of fail silence, the replica supplying fewer tokens thus far is deemed to be faulty.

6.3 Fault awareness and reactivity

At runtime, fault and critical event management is pursued with a *reaction* mechanism. As a pre-requisite, knowledge about presence and nature of the fault must be acquired by means of a fault detection/awareness mechanism. For this purpose, we designed the Local Fault Monitor (LO|FA|MO) [11], a distributed approach that employs additional hardware IPs on every DNP and dedicated software components running on each tile to create a *local awareness of faults and critical events*.

This local awareness is then propagated upward the system hierarchy (Figure 15) so that global response to the fault may be grounded onto the most capillary knowledge of system status.

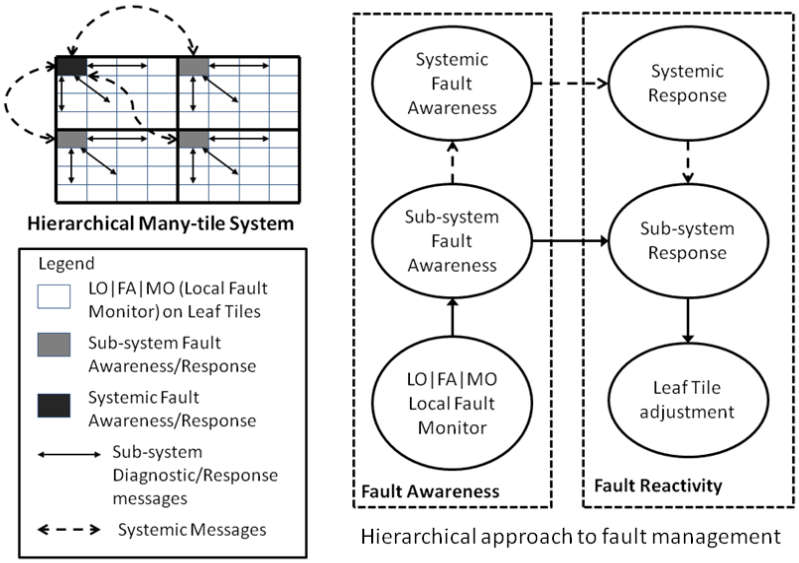

Figure 15 - The Network Processor of each leaf in the many-tile HW system is equipped with its own LO|FA|MO component. The Local Awareness of faults and critical events is propagated towards the upper hierarchy levels, creating Systemic Awareness. Reactions to faults and critical events are autonomously initiated by the sub-system controllers.

The runtime-manager pushes the reaction to the fault down the system hierarchy: it migrates processes assigned to a faulty processor to an alternative one. To include the evaluation of all possible failure scenarios in the design time analysis, spare cores and tiles are allocated during design space exploration and used by the runtime-manager as targets for migration. In the next sections, the LO|FA|MO approach and the proposed task migration mechanism, i.e., the system capability of dynamically transferring running tasks from their original source node to another one, are presented.

6.3.1 LO|FA|MO: Fault Detection and systemic Awareness for QUonG and VEP

Local Fault Monitor (LO|FA|MO) is a systemic approach to fault detection and awareness for distributed systems. We wanted this approach to guarantee a *no-single-point-of-failure* fault awareness, meaning that the presence of a faulty component should not prevent the system to become aware of it. With these guidelines, we designed LO|FA|MO as based on the following elements:

- A LO|FA|MO-enabled 3D toroidal network interface, i.e. the DNP, featuring a dedicated hardware IP (DNP Fault Manager-DFM), able to assess the DNP status, plus registers encoding the status of the DNP, of the host and of the first neighbouring hosts in the 3D network.
- A dedicated software (Host Fault Manager-HFM), running on each host, able to assess the host status and the DNP status as explained in the next point.
- A Mutual Watchdog scheme between each node host and DNP, where both are peers reading each other's status from the watchdog registers and updating their own; periodical read and write timing (with Tread > Twrite) ensure that the peers can determine each other's liveness.
- A Service Network for diagnostic messages accessible by each HFM instance.
- The DNP 3D network as a secondary path for diagnostic messages issued by each DFM.

We dub **Supervisors** those nodes in the mesh that monitor the system at a higher hierarchy level, thus being sinks of all diagnostic messages and center of decisions about fault reactivity. An example of the fault detection and awareness mechanism is given in Figures 16 e 17.

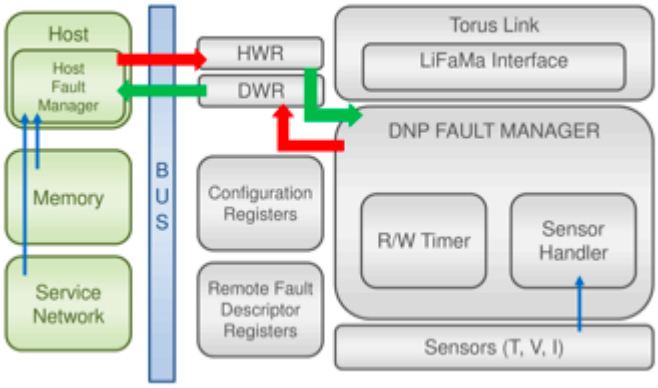

Figure 16 - LO|FA|MO mutual watchdog mechanism in which host and 3D network interface are peers monitoring each other by periodically reading and writing special DNP registers (DNP Watchdog Register and Host Watchdog Register).

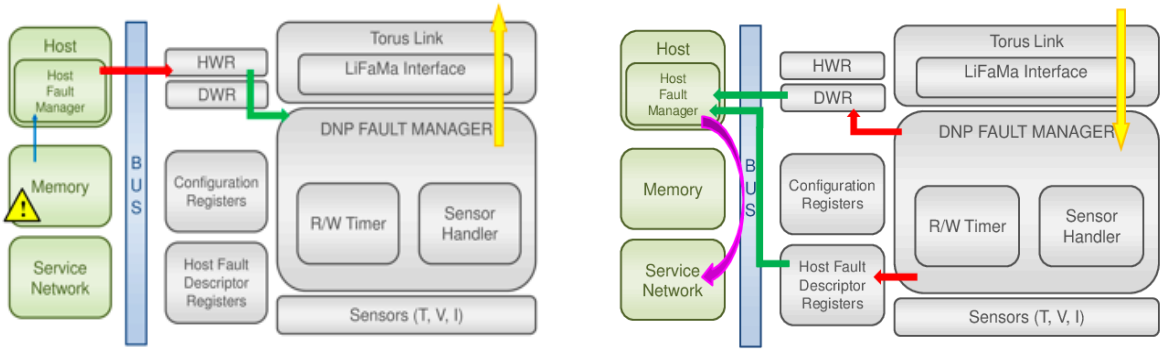

Figure 17 - An example of a host memory fault detection and systemic awareness. (Left) The fault is detected and reported in the Host Watchdog Register, the DNP Fault Manager reads the register and sends a diagnostic message via the 3D network to the first neighbour nodes. (Right) Information about the fault carried by the diagnostic message coming from the 3D network is reported in the other node's DNP Watchdog Register where the Host Fault Manager can read it. The Host Fault Manager can inform the Supervisor node through the Service Network.

The VEP DNP model contains the DNP Fault Manager block as mentioned in section 5.2.4. A Host Fault Manager software implementation runs in the AED processor to complete the mutual watchdog functionality. The entire LO|FA|MO approach has been validated on the VEP by using its fault injection facilities. The validation covered DNP diagnostic logic in the model, VEP service network and AED Fault Manager. VEP configurations with 64 (4x4x4) and 512 (8x8x8) tiles were used for the validation.

LO|FA|MO was implemented and used efficiently on the QUonG cluster [11]. In this environment, the APEnet+ core lodges a DFM component whose resource occupancy (in FPGA gates) is negligible compared to that of the whole DNP core. The Watchdog Registers live on the FPGA and are accessible in target mode over the PCIe bus in ~6 microseconds. Note that at the access rate corresponding to T_{read} and T_{write} in the range ~10-1000 ms the bus occupancy is very low as well.

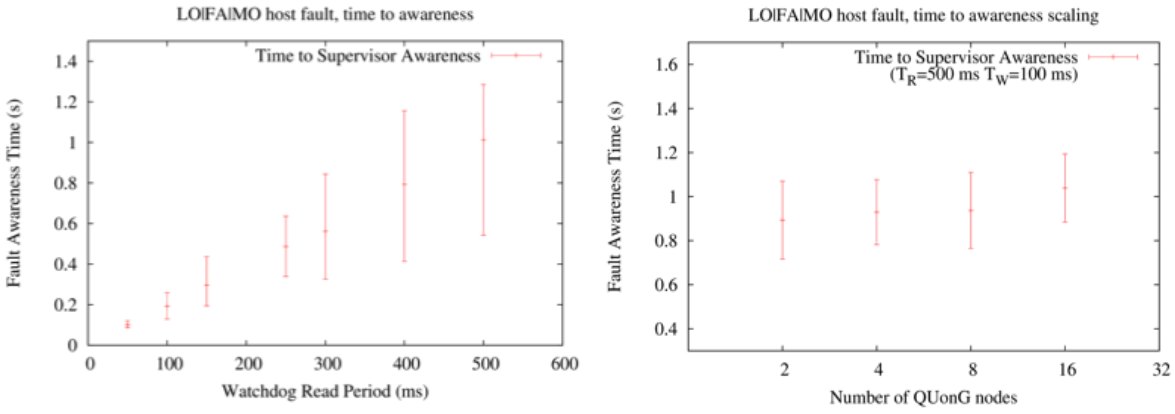

Figure 18 - (left) Time to obtain Supervisor awareness (T_{aw}) in case of host breakdown fault, plotted varying the watchdog period T_{read}. For each T_{read} value the mean value and the minimum and

maximum values are shown for T_{aw}. 21-(right) Time to Supervisor awareness (T_{aw}) scaling the number of QUonG node running LOFAMO. T_{read} = 500ms and T_{write} = 100ms.

The DFM is able to embed diagnostic messages in the physical link protocol of the 3D network, leading to a null impact of these messages on the NIC performance. Finally, the HFM is a Linux multi-threaded daemon running on each node CPU, with negligible CPU occupation.
The approach has been proved to be fast in providing fault awareness and scalable on QUonG, as shown in Figure 18.

6.3.2 Task migration between tiles

The proposed task migration solution is designed to be characterized by the following:

1. It is able to be plugged smoothly on different embedded operating systems taking into consideration the inherent limitations of systems in such domain.
2. It should be designed such that it remains transparent to the programmer. Application development should be unhindered by restrictions or even considerations about migration.
3. It is compatible with DAL specifications, i.e. a migrate-able task should be migrated and plugged in the generation tool flow.
4. It should not introduce heavy hits to the system in performance or memory overhead.

The proposed task migration solution is detailed in [31]. It is an agent-based solution that adopts a semi-distributed hierarchy, meaning that an aiding task (agent) exists in every tile so as to perform the migration once it receives a migration request. The first task is called **migration controller** and the second one is called **broker**. The adopted semi-distributed hierarchy is explained so that there is one broker in a cluster and one migration controller in each tile. The broker is responsible of issuing the request after taking the decision of migration according to system variables like temperature and link integrity (coming from specific monitoring processes). The migration controller, in turn, executes the migration process. This solution fits fully distributed memory architectures and is scalable so that it can be deployed on the system regardless of how many tiles it comprises.

Every migrate-able task has one destination to migrate to if necessary and its source becomes a future destination after migration; this is called single destination migration (SDM). This is due to the use of a scenario-based design flow where a spare tile is chosen to accommodate replicas of migrating tasks code. All destinations are determined statically before compilation and linking which allows taking code copies of the migrate-able tasks to be linked in their destinations. However, the decision to migrate is completely dynamic and can be applied at run-time. In our experiments, migration is limited to be within the same cluster. Nevertheless, it is still possible to migrate a task anywhere in the system at the cost of adding extra communication and synchronization between brokers. This alternative, together with single (instead of many) destinations for each migrate-able task, is chosen for different reasons:

- To avoid memory inflation in case of copying numerous migrate-able tasks to many tiles.
- To minimize the number of scenarios of task mappings which in turn facilitates the system design phase and avoids bloating the number of possible task mappings. This also contributes to making it possible to attain required levels of quality of service.
- To guarantee the quality of service in the destination as it is chosen by the mapping tool taking into considerations all objectives.

In order to function properly and achieve their purpose, both controlling tasks (Broker and Migration Controller) require necessary information about migrating tasks, their neighbour tiles and their destinations. Since task migration is added as a capability to the system, all its necessary tasks, tables and information are generated automatically along with DAL library and the applications in pre-compilation phase. The automatic generation tools is described in [57].

The broker, responsible for migration decisions, uses a table called global view table (GVT) which gives an overall view of all migrate-able tasks in all tiles in the cluster. For every tile, all migrate-able tasks are listed, their destinations, locations of all their neighbours and a field that determines whether migrating this task is possible or not. The migration controller, responsible for actual migration on each tile, uses another table called destination lookup table (DLT) containing all information of all migrate-able tasks which reside in their local tiles along with the information concerning their neighbours.

Task migration challenges and solutions

Several issues and challenges have to be addressed in the design of task migration solution (a complete treatment can be found in [31]), chiefly among them being communication inconsistency and task safe stop-and-resume; more in detail they are as follows:

A. Communication inconsistency

There are two main sources of communication inconsistency; they are:

1. Location change of migrated task: consistent communication resuming after migration is impossible without updating the migrated task address task on its neighbours.
2. Left unprocessed tokens: A migrating task may have unprocessed tokens stored in its input FIFO(s), therefore consistent communication resuming requires moving the unprocessed tokens to input FIFO(s) in the destination.

To overcome the issue of location change, we introduced ***configurable channels*** depicted in Figure 19. These channels differ from ordinary ones as they have two branches in the design time at the migrate-able task side: one is attached to the default location where the migrate-able task starts executing and the other is attached to its migrated destination. Only one branch is active at a time. When the migration controller receives a request, it switches the branches and updates the channel without the need of changing the port in the application layer, which leaves application code untouched and completely unaware to the change.

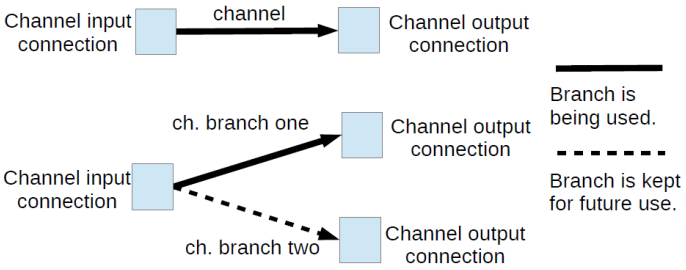

Figure 19 - Configurable channels

To overcome the issue of communication inconsistency, we propose using a communication protocol. This protocol is managed by a thin layer over the driver layer which resembles the data link layer. The protocol is all about writing with copy, i.e. once a task is writing to another, it locally stores all transmitted tokens until acknowledgment from the consuming task is received so that it can flush them from its local FIFO. With this protocol, we are sure that there is a copy of all unprocessed tokens still residing in the FIFO of the neighbour to the migrating task. This allows the re-forwarding of all unprocessed tokens so that communication with the replica can be resumed, ensuring communication consistency.
In Figure 20, a simple illustration for the communication protocol is depicted. On the sending side, T1 sends tokens to T2 keeping a copy in a copy buffer CB; on the receiving side, all incoming tokens are stored in a receive buffer RB, then T2 acknowledges T1 once it consumes them so that T1 can flush them from CB.

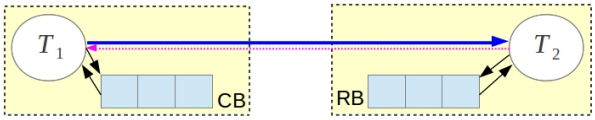

Figure 20 - *Communication protocol*

B. Task safe stop-and-resume

Another important challenge shared among all cases of migration regardless of system architecture, is guaranteeing safe stop and resuming of the migrated task at the destination node/tile. Migrating tasks must be stopped safely preserving their state and they must be resumed safely on their destinations. We use migration points to set predefined points where tasks can be safely stopped and resumed. A migration point is a checkpoint in the code of every migrate-able task that informs it whether a migration request was issued. We benefit from the programming model of EURETILE which supports loop-based data-flow applications so that a migration point is inserted in the process model where application code is supplied in the form of init, fire and finish procedures. In this way, no modification is required on the code in order to insert migration capability. When a task checks the aforementioned point and finds the migration request, it halts, allowing safe collection of task state to be sent to the destination.

7 Test applications and benchmarks

In order to evaluate the proposed design flow, two main sets of applications have been developed using the proposed programming model: 1- a set of dynamic multimedia many-process applications, including a picture-in-picture software for embedded video processing as well as a distributed implementation of a ray tracing algorithm and an H.264 codec pair (see [58]); 2- a many-process simulator of neural activity and plasticity that generates complex inter-process time-variant communication traffic patterns. This benchmark (see [59] is representative of a brain simulation, with

potential applications to embedded robotics. In this section, the two sets of benchmarks are described.

7.1 Multimedia applications

Data-flow kernel applications: To demonstrate the capabilities of the design flow, applications typically used in digital signal processing have been ported: a distributed implementation of an N-point fast Fourier transformation (FFT); an N-order IIR filter; a distributed implementation of a matrix multiplication.

Multi-stage video processing application: A multi-stage video-processing application has been implemented, decoding a motion-JPEG video stream and subjecting it to a motion detection method. The MJPEG decoder can decode multiple video frames in parallel. The motion detector is composed of a Gaussian blur, a gradient magnitude computing using Sobel filters and an optical flow motion analysis.

H.264 codec: The H.264/MPEG-4 AVC (Advanced Video Codec) is one of the most widely used video coding standards in recent years. The considered implementation is similar to one previously proposed for the HOPES framework [60] that supports the coding standard baseline profile.

Ray tracing: Providing a high degree of realism, raytracing is expected to be implemented as a real-time rendering algorithm in the next generation of embedded many-tile systems. Ray tracing applications naturally consist of three logical parts, the first one being the rays generation, the second - and most computationally intensive - being the intersection of the rays with the scene to render and third being aggregation of the calculated values and storage to an image file. This partition is the one used for the process network specification, as it allows us to neatly separate the intersection part, which we are mostly interested in parallelizing as it has the most impact on total runtime.

Recursive array-sorting: The considered sorting algorithm is quicksort which, being based on recursion, cannot be specified using conventional models of process networks. However, the application can be specified efficiently as an EPN. The top-level process network consists of three processes: Process “src” (“dest”) generates (displays) the input (output) array and process “sort” sorts the elements in ascending order. As the quicksort algorithm recursively sorts the array, process “sort” can be replaced by a structural description which divides the array into two smaller ones that can be individually sorted.

Picture-in-picture (PiP) video decoder: As sample of dynamic set of processes starting and stopping at unpredictable time (user interaction) we used the PiP video decoder application.
The PiP video decoder is composed of eight scenarios and three different video decoder applications. The HD application processes high-definition, the SD application standard-definition, and the VCD application low-resolution video data. The software has two major execution modes, namely watching high-definition (scenario HD) or standard-definition videos (scenario SD). In addition, the user might want to pause the video or watch a preview of another video by activating the PiP mode (i.e., starting the VCD application). Due to resource restrictions, the user is only able to activate the PiP mode when the SD application is running or paused, or the HD application is paused.
In section 8 we show the scalability on a multi-tile platform (VEP) of the state (start/stop) switching latency.

7.2 Distributed simulation of polychronous and plastic spiking neural networks

A natively distributed mini-application benchmark representative of plastic spiking neural network simulators (DPSNN-STDP) (see [59]) has been developed, with an architecture largely inspired by the large scale neuro-synaptic simulators developed by [61], [2] and [3]. Processes describe synapses in input to cluster of neurons with an irregular interconnection topology, with complex inter-process traffic patterns broadly varying in time and per process; it can be used to gauge performances of existing computing platforms and drive development of dedicated future parallel/distributed computing systems. The application was designed to be natively distributed and parallel, to be easily plugged to standard and custom software/hardware communication interfaces and, for a given configuration, to have output which is invariant against how many processes or hardware nodes it is run on, to simplify the scalability analysis on different architectures.

7.2.1 Validation of functionality and scaling of the many-process neuro-synaptic simulator

The many-nodes C++ process network simulating neural activity and synaptic plasticity is compatible with EURETILE DAL over DNA-OS and standard GNU/Linux plus MPI. Profiling and strong and weak scaling analysis of the code were performed first on QUonG (on a number of physical cores varying from 1 to 128) and over the commodity network. Grids of columns of Izhikevich neurons projected synapses locally and towards first, second and third neighbouring columns. Simulated network size

varied from 6.6G to 200K synapses (Figure 21). The code demonstrated to be fast and scalable: 128 hardware cores at 2.4GHz in 10s produce 1s of simulated activity and plasticity (per Hz of avg. firing rate) of a 3.2G synapses network (full description in [59]).

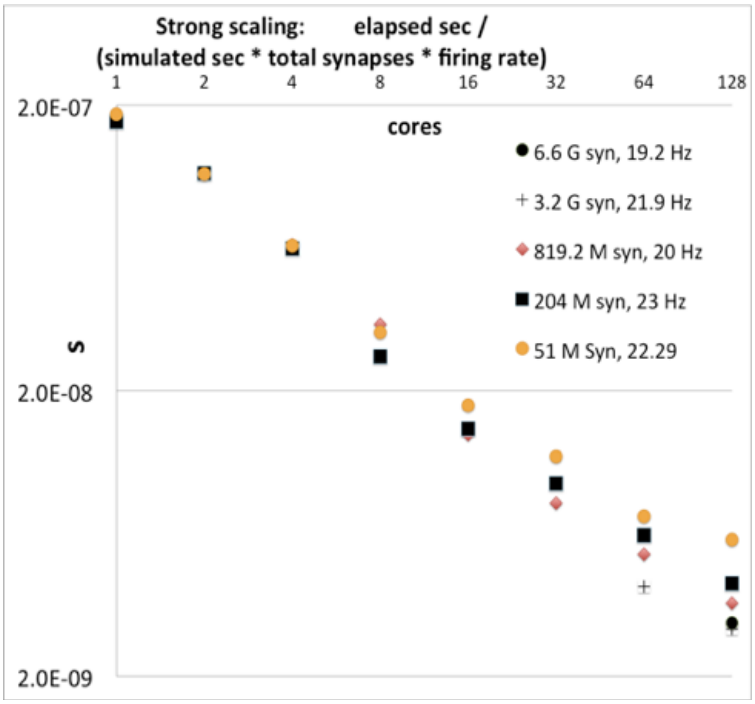

Figure 21 - Strong scaling of the DPSNN-STDP mini-app benchmark. For an ideal scaling the execution time should grow proportionally to the number of synapses and to the firing rate, and should reduce proportionally to the number of cores applied to the execution.

Results produced by a DPSNN benchmark run can be plotted in a “Rastergram”, a diagram describing the collective spiking activity of a neural network; in the one shown in Figure 22, the horizontal axis is the simulation time and the vertical axis is the identifier of individual neurons. Each dot represents a spiking event, i.e. the membrane potential of an individual neuron moving, for a few milliseconds, from a polarized state (around -65 mV) to a depolarized state (peaking around +30 mV). The workload needed to produce the first two seconds of simulated activity can be considered representative, because its production includes the simulation of the faster dynamic behaviour and of the slower plastic behaviour. During each second of simulation, neural dynamic is responsible of the network evolution (computed at submillisecond time resolution), while synaptic plasticity is applied at the end of each second of evolution and his effects are visible on the successive second of activity.

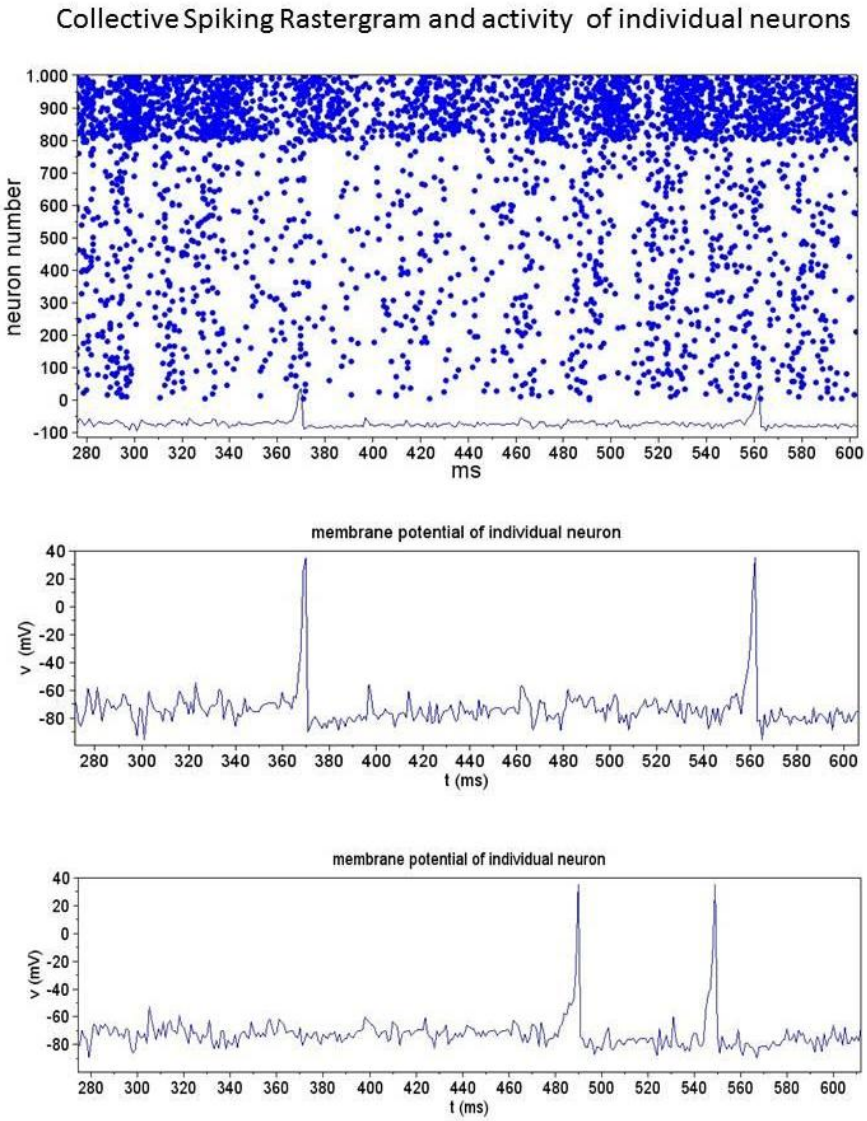

Figure 22 - Neural Activity: Collective Rastergram of the spiking activity of 1000 neurons (upper figure) and evolution of the membrane potential of two individual neurons (second and third graph), each one spiking twice (i.e. peaks of membrane potential).

This code demonstrated its ability to replicate the same spiking activity and neural plasticity produced by an open-source, non-parallel reference code made available by Izhikevich. In particular, we adopted a coding style that allowed to reproduce identical neuro-synaptic activity independently from the number of processes used for the parallelization. The distributed code produces identical spiking activity both on the MPI and DAL execution platforms. This feature simplifies the comparison between the performances on QUonG produced by the EURETILE tool-chain (DAL on DNA-OS) vs. a standard MPI on Linux environment reported in Section 8.

8 Experimental Integration Results

Experimental results from the execution on both the simulated platform (VEP) and the hardware platform (QUonG) are expanded here.

8.1 Experimental results on the QUonG hardware platform

8.1.1 DNA-OS on QUonG

DNA-OS [35] has been ported on different architectures. The QUonG porting includes all the features required to generate, start, and run applications, namely a PCIe driver, a DNP driver, an Intel IGB driver for the Ethernet interface (in order to start and synchronize all QUonGs), as well as support for multi-tile and multi-core. A script downloads all binaries using the IGB driver. Then we start a synchronization process of all QUonG. This is all performed in a fully automatic fashion and does not require any action from the designer.

8.1.2 Executing the DAL version of the neural DPSNN benchmark on QUonG

The DPSNN application has been ported to the DAL programming environment for execution on the EURETILE hardware platform, using DNA-OS and its software tool-chain for binary generation. In the DAL model of the DPSNN application, several parameters may be tuned at each run, e.g. the number of processes, of neurons per process and of synapses per neuron, the kind of interconnection between neurons (network topology) and of external thalamic input, etc. A 16-tiles QUonG cluster - a total of 128 hyperthreaded hardware cores - was used. In the following sections we report about 1- the comparison of the efficiency of DAL on DNA-OS execution vs. a "standard" environment (MPI on Linux) and 2- about the scaling on a full QUonG system (16 tiles interconnected with APENet+).

Efficiency of the EURETILE tool-chain (DAL on DNA-OS) vs "standard" (MPI on Linux)

For the developer of a many-process application wanting to run on a many-tile platform, the EURETILE toolchain provides a set of added values, most importantly high efficiency execution. To prove this, we gauged the runtime and scaling properties of the neural network simulation on a single multi-core server for a varying number of DAL (and MPI) processes. On a single QUonG multi-core node, inter-process channels can be implemented over shared memory and each process can run on a dedicated (HyperThreaded) core. Table II shows the efficiency of our software toolchain, compared with standard MPI in a GNU/Linux environment. We ran a relatively small neural network, 13.1M synapses, active at 12Hz mean rate, projected by a mix of 80% Izhikevich excitatory RS and 20% FS neurons, with a mean of 400 synapses per neuron. Long Term Potentiation/Depression of the synapses is also included in the simulation. Such "small" network challenges the capability to be efficiently distributed over a high number of processes and hardware resources. On each QUonG node we used for the application a maximum of 16 (HT) cores. As expected, best performances were obtained when processes were allocated one per core. Distributing the simulation over a greater number of processes (overcommitting the available hardware cores) led to slower execution.

number of software processes	1	2	4	8	16	32	64
MPI on Linux (exec seconds)	46.9	27.0	11.8	9.9	8.3	11.0	24.4
DAL on DNA-OS (exec seconds)	46.1	23.0	17.4	10.9	9.6	10.3	-

Table II - This table shows that the EURETILE DAL on DNA-OS tool chain has efficiency on-par with "standard" (MPI on Linux) when executing on an Intel server. A 3s activity simulation of a 13.1M synapses neural network was distributed over an increasing number of processes. Best runtime is obtained with one process per hardware core.

Further tests were carried out to show DNA-OS is able to provide the system with a nearly deterministic execution environment. Ten runs with two different configurations of 54.2M synapses were profiled: Configuration 1 contains 8 DAL processes running on 2 QUonG tiles, 4 processes per tile; Configuration 2 contains 16 processes evenly distributed on 4 QUonG tiles. This trial differs from the previous one in the fact that here off-tile communication exists - the DNP is used - hence its time is taken into consideration. The resulting statistics can be found in Table III.

	Configuration 1	**Configuration 2**
QUonGs	2	4
DAL processes	8	16
number of columns	64	128
Average execution time *(seconds)*	**20.8809585**	**23.6005321**
Standard deviation *(seconds)*	**0.007074652**	**0.010195061**
Range (difference between maximum and minimum execution times) *(seconds)*	**0.017256**	**0.03104**

Table III - This table demonstrates that the EURETILE DAL on DNA-OS tool chain is able to get a good determinism for different problem sizes and different QuonG platforms. This should be exploitable to reproduce simulation and to measure performances.

Standard deviation is quite small compared to the execution time. Moreover, DNA-OS provides a better ratio with about 7ms compared with about 200ms of standard deviation on GNU/Linux. Such a determinism could be exploited in simulation and profiling.

DAL on DNA-OS scaling to the full QUonG hardware platform

Another qualifying point was the ability of the EURETILE software toolchain to manage different QUonG platforms for different size problems, exploiting the APENet+ interconnect for communication among different tiles and shared memories for processes on the same tile. Tables IV and V report the execution times of the simulation of 3s for two different configurations.

Table IV - Configuration 1: the network is composed by 104.8 Million synapses:

Number of software processes (mapped on 16 QUonG servers, APENet+ interconnect) - 256 columns	16	32	64
DAL on DNA-OS (exec seconds)	26.3	21.9	14.0

Table V - Configuration 2: the network is composed by 54.4 Million synapses:

Number of software processes (mapped on 8 QUonG servers, APENet+ interconnect) - 128 columns	16	32	64
DAL on DNA-OS (exec seconds)	24.46	20.79	13.06

It is to note that network size is relatively small; a 16 nodes QUonG cluster would greatly benefit from 64 bit support to enable configurations of adequate size. At any rate, when we double the problem or the complexity of the application while also doubling the resources thrown at it (QUonG nodes, in our case), we observe that runtime stays almost the same. However, runtimes in Conf. 1 are slightly greater than in Conf. 2 since the communication increases in the case where more QUonG nodes are working.

8.1.3 Execution of task migration on QUonG server

Functionality of the proposed migration solution is validated on a small square-computing application. The rationale behind this choice is to show that overhead due to migration is still limited even for applications this small. Two instances {App1, App2} of the application are mapped on a 4-tiles QUonG platform as depicted below. The application has three tasks (Figure 23): *Generator, Square, Consumer.* **Generator** outputs floating point data to be squared by **Square** and finally displayed by **Consumer**. A migrating task is determined in each application instance: Square (`App1_square`) in App1 and Generator (`App2_generator`) in App2. The Broker would be mapped on a separate tile to avoid overheating.

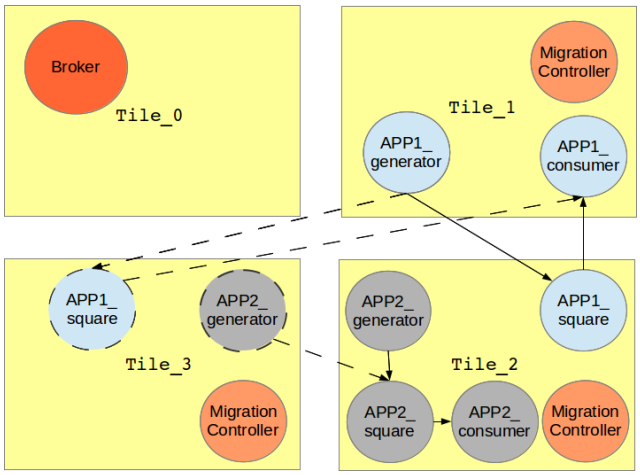

Figure 23 - Mapping of the application with migration agents on a 4-tiles QUonG platform.

The experiments are undergone in three cases, as follows:

1. Case 1: no migration agents exists, hence, this case represents the normal execution without the communication protocol (presented in section 6.3.2) or the migration agents.
2. Case 2: migration agents exist with the communication protocol, however, no migration request is to be issued.
3. Case 3: same as case 2, but with migration so as to bring everything into action.

In the Table VI the results of the experiments are shown, with overheads relative to Case 1. There is a negligible performance penalty (0.02% - 0.06%) due to the existence of migration agents along with

"write_and_copy" protocol incorporated in the inter-processes communications. Performance penalty due to migration is 14.5% - 18.25% depending on the position of migrating task in the task graph. This determines how many neighbours the migrating task has and their mappings and, as a result, the magnitude of inter migration controllers communications which, in turn, impacts migration overhead.

Application	Case	Starting time (µs)	Ending time (µs)	Execution time (µs)	Overhead
App1	1	65,027,461	67,247,538	2,220,077	N/A
	2	28,401,948	30,623,281	2,221,332	0.06
	3	21,754,757	24,379,930	2,625,173	18.25
App2	1	33,163,108	35,387,392	2,224,284	N/A
	2	28,687,143	30,911,856	2,224,713	0.02
	3	40,811,884	43,358,217	2,546,333	14.49

Table VI - Results of task migration experiments.

We show also the overhead variation with the size of task state in Figure 24. The data payload DNP packet can accommodate is 4kB, we have run some experiments on just App1 varying the task state from 4kB to 40kB in 8 steps. Migration overhead shows a linear relation with migrating task state size with approximately constant 45µs added for each 4kB increase in state size. This is because the DNP sends *n* packets where *n* = ceiling(S/4) where S is state size in kB.

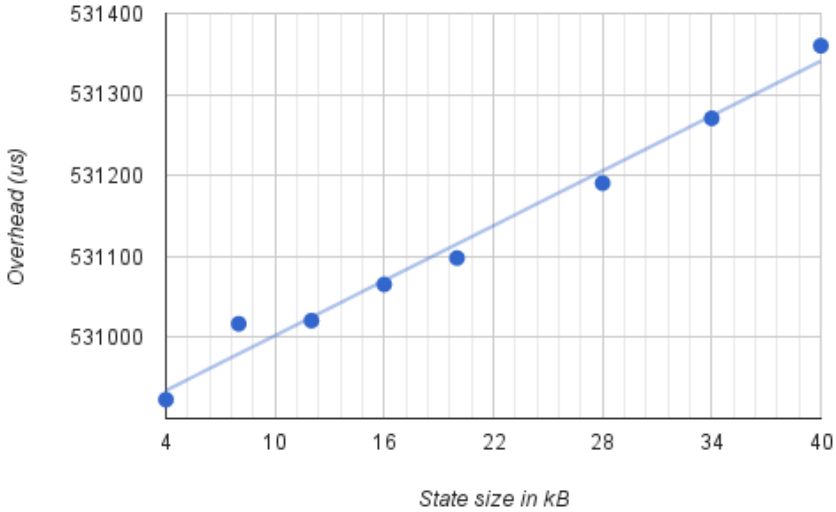

Figure 24 - Migration overhead (in µs) versus migrating task state size (in kB).

8.2 Experimental results on the VEP simulator

Apart from supporting the development of the software toolchain, the VEP was used to assess some essential non-functional characteristics of the EURETILE system. The IRISC in every tile was configured to run at 100MHz, for the sake of low power consumption, and with 4MB of on-tile memory (mimicking the combined capacity of nowadays SRAM and Flash on-chip memories). Further details of the VEP setup can be found in [10].

Data transfer rate: The aggregated data transfer rate between two DAL processes was measured for process mappings causing data packets to travel through a different number of tiles. Cases with 1, 8 and 31 DNP hops and increasing data token sizes were examined. The observed peak data rate is 22.3 MB/s as shown in Figure 25a.

Runtime manager overhead: how long the runtime environment takes to start and stop an increasing number of processes is shown in Figure 25b. Starting and stopping processes is the major cause of overhead when switching scenarios. This experiment compares the cases when several processes are mapped on a single tile and on two different tiles. On a single tile, the time to start and stop the application increases linearly with the number of processes. If the processes are distributed between two tiles, the time to start and stop the application is almost the half of the time for one tile.

Achieved speed-up due to application parallelism: Applications that expose coarse-grained parallelism can better exploit the EURETILE system to achieve speedup. For instance, the array-sorting benchmark (see Sec. 5.1.1) can be executed up to 7.01x faster when it is partitioned and executed on several tiles, as shown in Figure 25c. The cost of splitting an array prevents an even higher speedup.

Process and channel management overhead: If an application exposes fine-grained parallelism, then processes could be too small to absorb the overhead imposed by the process and channel management. This is the case of a 64-point FFT application that can be partitioned into processes executing simple butterfly operations. Figure 25d shows the execution time of FFT-64 when mapped on 64, 128, or 192 tiles compared to a reference on 32 tiles. The execution time is anyway reduced by a factor of 1.36 when going from 32 to 192 processes. This demonstrates low communication latency and low process management overhead of the EURETILE system.

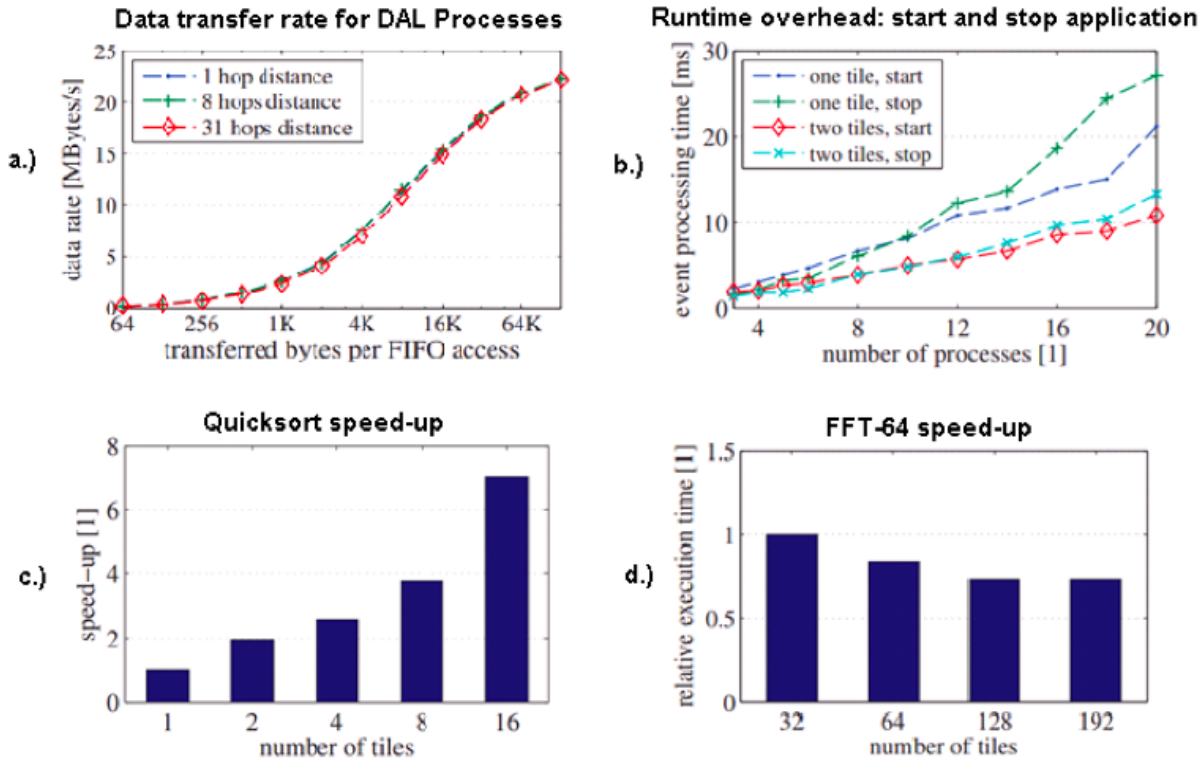

Figure 25 - EURETILE tool-chain characterization using the VEP.

9 Discussion of results

In this section we discuss the results presented in the previous section, highlighting lessons learned from benchmarking our target platforms. Finally, we discuss possible limitations of the proposed approach, both as conceptual limitations discovered thanks to actual integration work performed and conceptual analysis of possible further integration efforts. We also propose related future work.

9.1 Programming model and its scalability

The proposed DAL programming model has been one of the firsts that was able to provide reasoning about the correctness of a dynamic system at design-time and, at the same time, to efficiently exploit the available hardware resources at run-time. Its capability to specify complex systems has been shown in various case studies so that other frameworks have adapted the basic semantics of execution scenarios. Furthermore, the performed case studies demonstrated that the DAL programming model offers low communication latency and low process management overhead. In fact, by specifying the application as an EPN, the application's degree of parallelism can be selected

depending on the available resources so that scheduling and interprocess communication overheads can be minimized.

The current limitations of the programming model are based on the static specification at design-time. In fact, the original system configuration can only be adapted if the corresponding change is described in the original system specification. For instance, if unknown applications can be installed after deployment, system analysis and mapping decisions must be performed at run-time, requiring fast and accurate heuristics, which do not cause long response times. Furthermore, the speedup of a DAL application with respect to the number of available tiles is still limited by the intrinsic scalability of the application. In fact, the benchmarks developed in the context of EURETILE have shown two scalability issues of process network topologies. On the one hand, some of these applications are limited by one, or few processes that are most computing intensive. On the other hand, applications like the proposed neuro-synaptic simulator can be designed using native distributed coding and network topology styles leading to good scalability.

9.2 Fault management approach

Three complementary fault management strategies have been designed and implemented in the context of EURETILE to achieve high system dependability. Temperature related reliability issues are avoided by using thermal-aware mapping strategies. To execute time-sensitive applications, critical processes and subnetworks are duplicated during design space exploration. On the other hand, to reliably execute critical, but non time-sensitive applications with low runtime overhead, a comprehensive fault detection and reaction mechanism has been proposed. The hardware fault detection capability of LO|FA|MO has been tested on both the VEP simulator and the QUonG platform. LO|FA|MO is designed with sufficient redundancy to ensure that the system is always aware of any fault at the hardware layer. Clearly, the concept of fault tolerance and fault reactivity can also be combined to ensure that the system masks multiple faults while maintaining critical time parameters such as latency or throughput.

The limitations of the proposed fault management solutions stem from the chosen design strategy: the design makes an implicit assumption that all faults will either be detected by the LO|FA|MO detection system or will be observed as a timing fault, which does not cover certain fault categories, e.g. value faults. The basis of the chosen strategy is to focus on the harder problem of detecting a timing fault, with the knowledge that several mature value fault detection approaches are already available.

9.3 Runtime system

After several experiments with QUonG platform running both Linux and DNA-OS, it is clear that, when using a full fledged OS targeted mainly for HPC, the execution times spread considerably. On the contrary, running applications on embedded operating systems like DNA-OS keeps these times almost constant. The standard deviation found in the results expanded in Section 8.1.2 is 7ms for 2 QUonG nodes and 10ms for 4 (the standard deviation with Linux + MPI is between 200 to 600ms for the same application). So narrow a range for standard deviation proves that the system gets much more deterministic when running a basic operating system, let alone DNA-OS. We can highlight two main reasons: each experiment begins with the load of binaries, processors and DNA-OS bootstrap and the application start. With no in-cache data at the beginning of each run, this definitely increases determinism. Moreover, Linux has many tasks and services to manage besides the application which leads to higher unpredictability. Therefore, the higher level of determinism enabled by DNA-OS has important advantages like more accurate application profiling and power consumption estimation.

9.4 VEP simulator

Having a virtual platform in the design loop allowed to develop and debug many EURETILE system components before actual hardware was available, e.g. the DNP and its drivers, LO|FA|MO, the OS, thus hastening the design process. The massive parallel nature of EURETILE posed important scalability challenges that needed addressing by including parallel and abstract simulation technologies to support the VEP. The resulting framework can be used to simulate hundreds of tiles executing target binary code, thus enabling to study system scalability, as well. The capability of the VEP simulator is mainly defined by the employed processor model and ranges from 24 simulated tiles with the fastest ISS (IRISC IA-DBT, at 160 MIPS) for debugging purposes to 1000 tiles with a simplified model (IRISC IAP, at 25 MIPS) for scalability testing.

IRISC architectural features limit the capability to execute some types of applications in an efficient way. For instance, no floating-point hardware unit is provided, thus such data types have to be emulated in software by the compiler. Also, both IRISC core and VEP DNP are limited to a 32-bits address space which in turn limits the execution of applications with huge memory requirements on

the VEP. In addition, the maximum data transfer rate of DAL applications is limited by buffering limits of the DNP.

10 Conclusions

The EURETILE project investigated innovations to the many-process software and to the many-tile hardware architecture of future fault-tolerant embedded and HPC systems, for dynamic applications requiring extreme numerical and DSP capabilities. The project delivered: 1- Experimental many-tile hardware and simulation platforms, for the scenarios of dynamic many-process workloads to be run on future fault-tolerant HPC and Embedded Systems; 2- A many-tile programming/optimization environment, exhibiting several foundational innovations, to be applied to such dynamic many-process workload; 3- A set of application benchmarks, for both HPC and Embedded System domains, coded using the new programming environment, including 1- a benchmark representative of distributed simulation of neural activity and synaptic plasticity that generates complex inter-process traffic patterns (see [58]) and 2- a set of dynamic multimedia embedded applications including a picture-in-picture software for embedded video processing systems as well as distributed implementations of a ray-tracing algorithm and an H.264 codec pair .

We experimented the proposed architecture for applications described by a few hundreds of software processes executed on many-tile architectures, both actual and simulated, equipped with hundreds of cores interconnected by 3D toroidal interconnects. We discussed the benefits, lessons learned and limitations of the proposed approach, and in particular we analyzed the proposed programming model and its scalability, the fault management approach, the run-time systems and the simulator.

The way of representing parallelism, concurrency and fault management at system level, the implementation of the distributed real-time operating system and the efficient, fast simulation environment all have a strong impact on the overall optimality of the system implementation. The extensive software design framework proposed - including all challenges described - created a foreground, the reference EURETILE platform, that we think can be used as a starting point for the next years road map in many-tile processing.

11 Acknowledgements

The EURETILE project has been funded by the European Commission Grant Agreement no. 247846 Call: FP7-ICT-2009-4 Obj. FET-ICT-2009.8.1.
The authors acknowledge the contribution of other members of the EURETILE project, that participated during previous years, including:

- ETH Zurich: Hoeseok Yang
- RWTH: Christoph Schumacher, Jovana Jovic
- TIMA: Etienne Ripert, Ikbel Belaid, Julian Michaud, Mohamad Jabber, Alexandre Chagoya-Garzon and Xavier Guérin

12 References

[1] P.A. Merolla et al. A million spiking-neuron on integrated circuit with a scalable communication network and interface*, Science 345, 668 (2014)* (pag. 668-673).
[2] S.D. Modha et al. Cognitive Computing. *Communications of the ACM , 54* (08) 62-71.
[3] S.B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, A. D. Brown. Overview of the SpiNNaker system architecture. IEEE Transactions on Computers, vol. PP, no.99, (2012). doi: 10.1109/TC.2012.142.
[4] P.S. Paolucci, A.A. Jerraya, R. Leupers, L. Thiele, P. Vicini. SHAPES:: a tiled scalable software hardware architecture platform for embedded systems. Proceedings of the 4th Int. Conf. on Hardware/Software Codesign and System Synthesis, (2006). CODES+ISSS '06. , pp.167,172, 22-25 doi: 10.1145/1176254.1176297.
[5] P. S. Paolucci, P. Kajfasz, P. Bonnot, B. Candaele, D. Maufroid, E. Pastorelli, A. Ricciardi, Y. Fusella, and E. Guarino, “mAgic-FPU and MADE: A customizable VLIW core and the modular VLIW processor architecture description environment,” Computer Physics Communications, vol. 139, no. 1, pp. 132–143, September 2001.
[6] P.S. Paolucci, I. Bacivarov, G. Goossens, R. Leupers, F. Rousseau, C. Schumacher, L. Thiele, P. Vicini. EURETILE 2010-2012 summary: first three years of activity of the European Reference Tiled Experiment. (2013), arXiv:1305.1459 [cs.DC] , http://arxiv.org/abs/1305.1459.
[7] Roberto Ammendola, Andrea Biagioni, Ottorino Frezza, Francesca Lo Cicero, Alessandro Lonardo, Pier Stanislao Paolucci, Davide Rossetti, Francesco Simula, Laura Tosoratto, Piero Vicini “QUonG: A GPU-based HPC system dedicated to LQCD computing. In Application

Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on, pages 113–122, 2011.
[8] R Ammendola, A Biagioni, O Frezza, A Lonardo, F Lo Cicero, P S Paolucci, D Rossetti, F Simula, L Tosoratto, and P Vicini. APEnet+ 34 Gbps data transmission system and custom transmission logic. Journal of Instrumentation, 8(12):C12022, 2013.
[9] C. Schumacher, J.H. Weinstock, R. Leupers, G. Ascheid, L. Tossorato, A. Lonardo, D. Petras, T. Groetker. legaSCi: Legacy SystemC Model Integration into Parallel SystemC Simulators. 1st Workshop on Virtual Prototyping of Parallel and Embedded Systems (VIPES), Boston, USA, May. 2013.
[10] L. Schor, I. Bacivarov, L.G. Murillo, P.S. Paolucci, F. Rousseau, A. El Antably, R. Buecs, N. Fournel, R. Leupers, D. Rai, L. Thiele, L. Tosoratto, P. Vicini, and J.H. Weinstock. EURETILE Design Flow: Dynamic and Fault Tolerant Mapping of Multiple Applications onto Many-Tile Systems. Proc. Int'l Symp. on Parallel and Distributed Processing with Applications (ISPA), Aug. 2014.
[11] R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, P. S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto and P. Vicini "LO|FA|MO: Fault Detection and Systemic Awareness for the QUonG computing system", to be published, IEEE Proceedings (SRDS) International Symposium on Reliable Distributed Systems, Nara, Japan, October 6-9, 2014.
[12] R. Giorgi, R. M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. R. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Koliaï, J. Landwehr, N. Minh L, F. Li, M. Lujàn, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, M. Valero "TERAFLUX: Harnessing dataflow in next generation teradevices", Journal of Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), April 2014.
[13] Shin-Haeng Kang, Hoeseok Yang, Lars Schor, Iuliana Bacivarov, Soonhoi Ha, and Lothar Thiele. Multi-objective mapping optimization via problem decomposition for many-core systems. In Embedded Systems for Real-time Multimedia (ESTIMedia), 2012 IEEE 10th Symposium on, pp. 28-37. IEEE, 2012.
[14] L. Schor, I. Bacivarov, H. Yang, and L. Thiele. AdaPNet: Adapting Process Networks in Response to Resource Variations. Proc. Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), Oct. 2014.
[15] Singh, A.K., Shafique, M., Kumar, A., Henkel, J., "Mapping on multi/many-core systems: Survey of current and emerging trends," *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE* , vol., no., pp.1,10, May 29 2013-June 7 2013.
[16] Dong Chen, Noel A. Eisley, Philip Heidelberger, Robert M. Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L. Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J. Parker, "The IBM Blue Gene/Q interconnection network and message unit", Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA, , Article 26 , 10 pages, 2011.
[17] Yuichiro Ajima, Tomohiro Inoue, Shinya Hiramoto, Shunji Uno, Shinji Sumimoto, Kenichi Miura, Naoyuki Shida, Takahiro Kawashima, Takayuki Okamoto, Osamu Moriyama, Yoshiro Ikeda, Takekazu Tabata, Takahide Yoshikawa, Ken Seki, Toshiyuki Shimizu, "Tofu Interconnect 2: System-on-Chip Integration of High-Performance Interconnect", Supercomputing Lecture Notes in Computer Science Volume 8488, pp 498-507, 2014.
[18] Zhengbin Pang, Min Xie, Jun Zhang, Yi Zheng, Guibin Wang, Dezun Dong, Guang Suo, "The TH Express high performance interconnect networks", Frontiers of Computer Science, Volume 8, Issue 3, pp 357-366, June 2014.
[19] Bob Alverson, Edwin Froese, Larry Kaplan, Duncan Roweth, "Cray XC Series Network", http://http://www.cray.com/Assets/PDF/products/xc/CrayXC30Networking.pdf.
[20] Liming Chen; Avizienis, A., "N-VERSION PROGRAMMINC: A FAULT-TOLERANCE APPROACH TO RELIABILITY OF SOFTWARE OPERATION," *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on* , vol., no., pp.113,, 27-30 Jun 1995.
[21] Devendra Rai, Pengcheng Huang, Nikolay Stoimenov, and Lothar Thiele. An Efficient Real Time Fault Detection and Tolerance Framework Validated on the Intel SCC Processor. In Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference, pp. 1-6. ACM, 2014.
[22] Suen, Tony TY, and Johnny SK Wong. "Efficient task migration algorithm for distributed systems." Parallel and Distributed Systems, IEEE Transactions on 3, no. 4 (1992): 488-499.
[23] Lu, Chin, and Sau-Ming Lau. "A performance study on load balancing algorithms with task migration." *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*. IEEE, 1994.
[24] Chang, Hua Wu David, and William JB Oldham. "Dynamic task allocation models for large distributed computing systems." *Parallel and Distributed Systems, IEEE Transactions on* 6.12 (1995): 1301-1315.

[25] Bertozzi, Stefano et al. "Supporting task migration in multi-processor systems-on-chip: a feasibility study." Proceedings of the conference on Design, automation and test in Europe: Proceedings 6 Mar. 2006: 15-20.
[26] Mulas, Fabrizio et al. "Thermal balancing policy for multiprocessor stream computing platforms." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28.12 (2009): 1870-1882.1.
[27] Holmbacka, Simon et al. "Task Migration for Dynamic Power and Performance Characteristics on Many-Core Distributed Operating Systems." *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on* 27 Feb. 2013: 310-317.
[28] Almeida, Gabriel Marchesan, Sameer Varyani, Rémi Busseuil, Gilles Sassatelli, Pascal Benoit, Lionel Torres, Everton Alceu Carara, and Fernando Gehm Moraes. "Evaluating the impact of task migration in multi-processor systems-on-chip." In *Proceedings of the 23rd symposium on Integrated circuits and system design*, pp. 73-78. ACM, 2010.
[29] Fu, Fangfa, et al. "Low overhead task migration mechanism in NoC-based MPSoC." *ASIC (ASICON), 2013 IEEE 10th International Conference on*. IEEE, 2013.
[30] Quan, Wei, and Andy D. Pimentel. "A system-level simulation framework for evaluating task migration in MPSoCs." *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 2014.
[31] El-Antably, A, O Gruber, N Fournel and F Rousseau. "Transparent and Portable Agent Based Task Migration for Data-Flow Applications on Multi-Tiled Architectures", CODES-ISSS 2015, to be published.
[32] David Culler, Jaswinder Pal Singh, and Anoo Gupta, Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann, 1999.
[33] Lars Schor, Hoeseok Yang, Iuliana Bacivarov and Lothar Thiele. Expandable Process Networks to Efficiently Specify and Explore Task, Data, and Pipeline Parallelism. In Proc. Int'l Conf. on Compilers Architecture and Synthesis for Embedded Systems (CASES), pages. 1-10, 2013.
[34] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. Scenario-Based Design Flow for Mapping Streaming Applications onto On-Chip Many-Core Systems. In Proc. Int'l Conf. on Compilers Architecture and Synthesis for Embedded Systems (CASES), pages 71–80, 2012.
[35] Xavier Guerin and Frederic Petrot, A System Framework for the Design of Embedded Software Targeting Heterogeneous Multi-core SoCs, in Proc. Int'l Conf. on Application-specific Systems, Architectures and Processors, pages 153-160, 2009.
[36] Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In IFIP Congress, vol. 74, pages. 471–475, 1974.
[37] Engler, Dawson R. "The Exokernel operating system architecture." 18 May. 1998.
[38] Experts, ACE Associated Compiler. "The CoSy compiler development system." 2007.
[39] R. Ammendola, A. Biagioni, O. Frezza, A. Lonardo, F. Lo Cicero, P.S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto and P. Vicini, "APEnet+ 34 Gbps data transmission system and custom transmission logic," in JINST, Journal of Instrumentation, Proceedings of Topical Workshop on Electronics for Particle Physics (TWEPP) 2013, IOP Publishing, 2013.
[40] Roberto Ammendola, Massimo Bernaschi, Andrea Biagioni, Mauro Bisson, Massimiliano Fatica, Ottorino Frezza, Francesca Lo Cicero, Alessandro Lonardo, Enrico Mastrostefano, Pier Stanislao Paolucci, Davide Rossetti, Francesco Simula, Laura Tosoratto, and Piero Vicini. GPU Peer-to-Peer Techniques Applied to a Cluster Interconnect. In Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, pages 806–815, 2013.
[41] Roberto Ammendola, Andrea Biagioni, Ottorino Frezza, Francesca Lo Cicero, Alessandro Lonardo, Pier Stanislao Paolucci, Davide Rossetti, Francesco Simula, Laura Tosoratto, Piero Vicini, "Virtual-to-Physical Address Translation for an FPGA-based Interconnect with Host and GPU Remote DMA Capabilities.," in Field-Programmable Technology (FPT), 2013 International Conference on, 2013.
[42] IEEE standard SystemC language reference manual, IEEE Std. 1666-2011, 2012.
[43] http://www.synopsys.com/Systems/BlockDesign/processorDev (accessed: Nov. 2014).
[44] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr and A. Hoffmann. A universal technique for fast and flexible instruction-set architecture simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2004.
[45] D. Jones and N. Topham. High Speed CPU Simulation Using LTU Dynamic Binary Translation. In Proc. of International Conference on High Performance Embedded Architectures and Compilers (HiPEAC). Paphos, Cyprus, 2009.
[46] Accellera. "SystemC Standard Reference Implementation". http://www.accellera.org/downloads/standards/systemc. (accessed: Nov. 2014).

[47] C. Schumacher, R. Leupers, D. Petras, A. Hoffmann. parSC: synchronous parallel systemc simulation on multi-core host architectures. In Proc. of Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS). Scottsdale (AZ), USA. Oct. 2010.
[48] J. H. Weinstock, C. Schumacher, R. Leupers, G. Ascheid and L. Tosoratto. Time-Decoupled Parallel SystemC Simulation. In Proc. Design, Automation & Test in Europe (DATE). Dresden, Germany. Mar. 2014.
[49] L.G. Murillo, J. Harnath, R. Leupers and G. Ascheid. Scalable and Retargetable Debugger Architecture for Heterogeneous MPSoCs. In Proc. System, Software, SoC and Silicon Debug Conference (S4D). Vienna, Austria, Oct. 2012.
[50] L.G. Murillo, R. Buecs, D. Hincapie, R. Leupers and G. Ascheid. SWAT: Assertion-based Debugging of Concurrency Issues at System Level. In Asia South Pacific Design Automation Conference (ASP-DAC). Chiba, Japan. Jan. 2015 (accepted for publication).
[51] A. Pnueli. The temporal logic of programs. In 18th Annual Symposium on Foundations of Computer Science. IEEE, 1977.
[52] L.G. Murillo, S. Wawroshek, J. Castrillon, R. Leupers and G. Ascheid. Automatic Exploration of Software Concurrency Bugs with Event Ordering Constraints. In Proc. Design, Automation & Test in Europe (DATE). Dresden, Germany. Mar. 2014.
[53] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. Efficient Worst-Case Temperature Evaluation for Thermal-Aware Assignment of Real-Time Applications on MPSoCs. Journal of Electronic Testing 29, no. 4 (2013): 521-535.
[54] Lothar Thiele, Lars Schor, Iuliana Bacivarov, and Hoeseok Yang. Predictability for timing and temperature in multiprocessor system-on-chip platforms. ACM Transactions on Embedded Computing Systems (TECS) 12, no. 1s (2013): 48.
[55] Hopkins, A.L., Jr. A highly reliable fault-tolerant multiprocess for aircraft. Proc. IEEE, pages 1221–1239, 1978.
[56] B. D. Milburn. Apparatus and method for initializing a master/checker fault detecting microprocessor, 1998.
[57] El-Antably, A, N Fournel and F Rousseau. "Integrating task migration capability in software tool-chain for data-flow applications mapped on multi-tiled architectures.", DSD'2015, appears in August.
[58] P.S. Paolucci, I. Bacivarov, D. Rai, L. Schor, L. Thiele, H. Yang, E. Pastorelli, R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, F. Simula, L. Tosoratto, P. Vicini. EURETILE D7.3 - Dynamic DAL benchmark coding, measurements on MPI version of DPSNN-STDP (distributed plastic spiking neural net) and improvements to other DAL codes. (Aug 2014), arXiv:1408.4587 [cs.DC], http://arxiv.org/abs/1408.4587.
[59] P.S. Paolucci, R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, E. Pastorelli, F. Simula, L. Tosoratto, P. Vicini. Distributed simulation of polychronous and plastic spiking neural networks: strong and weak scaling of a representative mini-application benchmark executed on a small-scale commodity cluster. (2013) arXiv:1310.8478 [cs.DC], http://arxiv.org/abs/1310.8478.
[60] Hanwoong Jung, Chanhee Lee, Shin-Haeng Kang, Sungchan Kim, Hyunok Oh, and Soonhoi Ha. Dynamic Behavior Specification and Dynamic Mapping for Real-Time Embedded Systems: HOPES Approach. ACM Transactions on Embedded Computing Systems (TECS) 13, no. 4s (2014): 135.
[61] E. M. Izhikevich, G. M. Edelman. Large-scale model of mammalian thalamocortical systems. PNAS March 4, 2008 vol. 105 no. 93593-3598.

Alessandro Lonardo received his Master's Degree in Physics in 1997, from University ``Sapienza" in Rome, Italy. His thesis work involved, the development of a DSL optimizing compiler for the SIMD APEmille, supercomputer. He contributed to the design of the apeNEXT SPMD, parallel computer, developed its multi-node functional simulator and, ported the gcc compiler to the new architecture. He was one of the, designer of the Distributed

Network Processor, an IP enabling 2D/3D, internode communications in embedded multi-tile systems, and developed, its TLM-SystemC model. Currently he works on the design and, development of the APEnet+ 3D-torus network and NaNet real-time NIC.

Andrea Biagioni received his Master's Degree in, Physics in 2006 from University Sapienza in Rome, Italy, with a thesis regarding interconnection network implementation for the massively parallel computers designed by the APE Group at Istituto, Nazionale di Fisica Nucleare (INFN). Since 2007 he has worked in this group as hardware developer collaborating at SHAPES and EURETILE, projects involved in the VHDL coding for the development of a network, processor and its data transmission control logic and for fast I/O, mechanism with GPUs. His main research interests are High Performance, Computer, NIC design, communication protocol, FPGA, fault tolerance, and routing algorithms.

Ashraf El-Antably received his bachelor of science degree in, Communication and Electronics from Faculty of Engineering, Alexandria, University in Egypt. After working in the industry in the field of, embedded software for several years, he pursued his masters in, embedded systems design and got its master of science degree in 2011, from faculty of informatics, University of Lugano in Switzerland., Currently, he is pursuing his doctorate at University of Grenoble in, France since 2011. He does his research in TIMA laboratory. His, research interests are system level design for embedded systems, embedded software and multi-core architecture.

Clément Deschamps got a Master degree in Computer Science, from the University Joseph Fourier of Grenoble. He joined the SLS, team of TIMA lab in 2011 as a software developper. He is involved, in several projects, all of them dealing with low level software, developement or synthesis, for both simulation environment, or real hardware machines or clusters.

Davide Rossetti has a degree in Theoretical Physics from Sapienza Rome, University and is currently a senior engineer at NVIDIA Corp. His main, research activities are in the fields of design and development of, parallel computing and high-speed networking architectures optimized, for numerical simulations, while his interests spans different areas, such as HPC, computer graphics, operating systems, I/O technologies, GPGPUs, embedded systems, digital design, real-time systems, etc.

Devendra Rai is a currently pursuing a PhD from ETH Zurich. He is, interested in design, analysis, and implementation of complex, multiprocessor systems, specifically those which must meet stringent, timing and reliability expectations. Previously, Devendra Rai, graduated with a master's degree from the University of Virginia, (Computer Engineering, 2009) and worked in design and implementation, of complex computer systems in the automotive industry for three, years, and as a consultant in the USA for six months where he designed, and implemented an enterprise scale distributed computer system, (2009-2010).

Elena Pastorelli received her Master Degree in Physics from the, University of Rome Sapienza, Italy, in 1997. For fifteen years, worked in the embedded systems area for research industry, contributing to the development of two generations of floating point Digital Signal Processors. Technical experience principally related to the embedded systems and parallel custom architectures. Expertise in Digital Signal Processors software and hardware areas and in code optimisation techniques. In October 2013 joined INFN as researcher, to contribute to the development and optimization of a parallel and distributed simulator of polychronous neural network with time dependent synaptic plasticity.

Francesca Lo Cicero received his Master's Degree in Electronics Engineering in 2005 from University Tor Vergata in Rome, Italy with a thesis regarding VHDL FPGA-based design of a Calibration System for a Digital Beam Forming Network. Since 2006 she has worked for INFN as hardware developer. Her main research interest is the development of network architecture on FPGA for massively parallel processing systems, focusing on hardware accelerations and optimizations.

Francesco Simula received his Master's Degree in Theoretical Physics in 2006 from University Sapienza in Rome, Italy with a thesis regarding spin glass simulations performed on the massively parallel computers designed by the APE Group, the supercomputing initiative internal to Istituto Nazionale di Fisica Nucleare (INFN). Since 2006, he has worked at the Department of Physics of University Sapienza as temporary research fellow and then for INFN as developer of high performance numerical simulation of scientifically interesting codes.

Paralled computing is his main research interest, focusing on high performance networking and GPU acceleration on both HPC and embedded systems.

Frédéric Rousseau received the Engineer degree in computer science and electrical engineering from the University of Grenoble in 1991 and a Ph.D. in computer science in 1997 from the University of Evry - France. He has hold an assistant professor position at the University of Grenoble since October 1999 and a professor position since 2007. He is researcher in TIMA lab. His research interest concerns Systems on Chip design and architecture, prototyping of hardware/software systems and high level synthesis for embedded systems.

Iuliana Bacivarov received the electrical engineering degree in 2002 from the National Polytechnic Institute of Bucharest, Romania. In 2003, she received a master's degree in microelectronics integrated systems design from the Université Joseph Fourier in Grenoble, France. She received her Ph.D. degree in microelectronics from the National Polytechnic Institute of Grenoble in 2006. She has been a post-doctoral researcher at the Computer Engineering and Networks Laboratory of ETH Zurich since 2006. Her research interests include design, analysis, and optimization of MPSoC.

Jan Henrik Weinstock studied electrical engineering at RWTH Aachen University where he received his diploma in 2011. During the same year he joined the Institute for Communication Technologies and Embedded Systems (ICE) as a full-time research assistant where he is currently pursuing his Ph.D .-studies under supervision of Prof. Dr. rer. nat. Leupers. Since January 2015, he is the Chief Engineer of the Chair for Software for Systems on Silicon. His research interests focus on parallel SystemC simulation and virtual prototyping of multi-processor systems, but also include various other aspects of embedded system design.

Lars Schor is a researcher at the Computer Engineering and Networks Laboratory of ETH Zurich, Switzerland. His research interests include multi-processor systems and thermal analysis methods for embedded real-time systems. He received his MSc degree and his PhD degree in computer engineering from ETH Zurich in 2011 and 2014, respectively. In 2011, he received the "Willi Studer Price" and the "ETH Medal", both from ETH Zurich. In 2012, he received the "Intel Doctoral Student Honor Award".

Laura Tosoratto received a Master's Degree in Physics in 2005 from University Sapienza in Rome, Italy, with a thesis about the porting of the GNU C Compiler for the apeNEXT supercomputer architecture, developed by the APE group at Istituto Nazionale di Fisica Nucleare (INFN). Since then she has worked as researcher in this group contributing as sofware developer to different projects with EU funded grants. Her main research interests are High Performance Computing and Networking, Message Passing libraries, Fault Tolerance and Distributed Platform functional simulators.

Lothar Thiele joined ETH Zurich, Switzerland, as a full professor of computer engineering in 1994. He received his Dr.-Ing. degree in Electrical Engineering from the Technical University of Munich in 1985. His research interests include models, methods and software tools for the design of embedded systems and bioinspired optimization techniques. In 1986, he received the "Dissertation Award" of the Technical University of Munich. Since 2010, he is a member of the Academia Europaea. In 2013, he joined the National Research Council of the Swiss National Science Foundation.

Luis Gabriel Murillo received an Electronics Engineering degree from the University of Antioquia, Colombia, in 2007, and a MSc. degree in Embedded Systems Design from the University of Lugano, Switzerland, in 2009. He joined the Institute for Communication Technologies and Embedded Systems of the RWTH Aachen University in 2009, where he works as full time researcher while pursuing a Ph.D. degree in Electrical Engineering. His research intersests comprise programming and debugging of complex multicore systems, simulation technologies, system-level analysis/optimization, and hardware/software codesign.

Michele Martinelli received his Master's Degree in Computer Science in 2013 from University of Rome "La Sapienza", Italy. He's currently pursuing a PhD in Computer Science, working in the APE Group at Istituto Nazionale di Fisica Nucleare (INFN). His research activities are in the field of hardware/software co-design, device driver and software optimization, while his interests focus on HPC, operating systems, GPGPUs and networks.

Nicolas Fournel is a associate professor at University of Grenoble Alpes. Previously, he has been a research engineer at Kalray, a startup company in Grenoble, France, that designs massively parallel multicore SoCs. He completed the work described in this article while working as a researcher in the System Level Synthesis Group at TIMA Laboratory. His research focuses on high- speed simulation and low-level software for integrated multiprocessor systems. He has a PhD in computer science from Ecole Normale Supérieure de Lyon.

Ottorino Frezza received his Master's Degree in Physics in 2005 from University Sapienza in Rome, Italy with a thesis regarding VHDL FPGA-based design of a Read out board for a high energy physic experiment (NEMO). He worked from 2007 to 2009 for Eurotech s.p.a. as technical employee. Since 2009 he has worked for INFN as hardware developer. His main research interest is the development of network architecture on FPGA for massively parallel processing systems, focusing on high speed interfaces.

Piero Vicini received his Master's Degree in Physics from University Sapienza in Rome, Italy, then joined Istituto Nazionale di Fisica Nucleare (INFN) in 1993, where he is currently senior research associate. From 1993 he was one of the principal investigators of APE Group, INFN supercomputing initiative, as responsible for hardware development, VLSI design and APE Supercomputers production. Since 2005, he is the research group spokesman and coordinator. Current research interests are development of massively parallel processing systems optimized for scientific numerical simulation, in particular, floating point processor architectures, dedicated network architecture on FPGA, computational accelerators and high performance system integration.

Pier Stanislao Paolucci received his Physics M.Sc. degree from University Sapienza (Rome, Italy). He coordinates the European EURETILE project. Previously, he coordinated the European SHAPES project. He is an INFN researcher, and has been member of the INFN APE group since its foundation (1984). The APE group designed several generations of massive parallel/distributed numerical computers. He also served as CTO of ATMEL Roma, leading the design of the DIOPSIS MPSoCs and mAgic VLIW numerical processors. He patented about MPSoC and VLIW. Paolucci also invented the `Cubed-Sphere' gridding and co-invented 'Evolving Grammars'. He is a member of ACM and IEEE.

Rainer Leupers received the M.Sc. and Ph.D. degrees in Computer Science from the TU Dortmund, Germany, in 1992 and 1997. From 1997-2001 he was the chief engineer at the Embedded Systems chair at TU Dortmund. Since 2002, he has been professor at the RWTH Aachen University. His research comprises software development tools, processor architectures, and system-level electronic design automation. He has served as consultant for various companies, as an expert for the European Commission, and in the committees of research initiatives like UMIC, TETRACOM, HiPEAC, and ARTIST. He has been a co-founder of LISATek (now with Synopsys) and Silexica.

Róbert Lajos Bücs has received his BSc. degree from the Budapest University of Technology and Economics in 2010. He continued his studies on the RWTH Aachen University in Germany where he pursued his MSc. degree from the Institute of Communication Technologies and Embedded Systems (ICE) in cooperation with Synopsys Ltd. He started his doctoral studies in mid 2013. Mr. Bücs is currently actively working on the research topic "Multi-domain, multi-scale co-simulation for automotive". The project emphasizes on applying the virtual platform simulation technology and ESL-level abstractions in the automotive domain.

Roberto Ammendola received his Master's Degree in Physics in 2002 from ``Tor Vergata'' University in Rome, Italy. He is with INFN since 2003, but worked also for Centro Fermi and ``Tor Vergata'' University. His main research interests are FPGAs and high speed network interconnects, HPC systems and infrastructure design and deployment.