Next generation of Exascale-class systems

DOI:
10.1016/j.micpro.2018.05.009

Document Version
Accepted author manuscript

Next Generation of Exascale-class Systems: ExaNeSt project and the status of its interconnect and storage development

Roberto Ammendola
INFN Sezione di Tor Vergata and
Electronic Engineering Dept.,
University of Roma "Tor Vergata"
Rome, Italy
roberto.ammendola@roma2.infn.it

Giuliano Taffoni
INAF, Osservatorio Astronomico di Trieste
Trieste, Italy
taffoni@oats.inaf.it

Jose A. Pascual, Javier Navaridas,
Mikel Luján and John Goodacre
University of Manchester
United Kingdom
{name.surname}@manchester.ac.uk

Bernd Lietzow
Fraunhofer
bernd.lietzow@itwm.fraunhofer.de

Angelos Mouzakitis
Virtual Open Systems
a.mouzakitis@virtualopensystems.com

Andrea Biagioni, Paolo Cretaro,
Ottorino Frezza, Francesca Lo Cicero,
Alessandro Lonardo, Michele Martinelli*,
Pier Stanislao Paolucci, Elena Pastorelli,
Francesco Simula and Piero Vicini
INFN Sezione di Roma
Rome, Italy
{name.surname}@roma1.infn.it
*corresponding author

Nikolaos Chrysos, Manolis Marazakis and Manolis Katevenis
FORTH - Foundation For Research & Technology
Hellas
{nchrysos, maraz, kateveni}@ics.forth.gr

Paolo Gorlani and Stefano Cozzini
CNR-IOM, eXact lab s.r.l.
{name.surname}@exact-lab.it

Giuseppe Piero Brandino
eXact lab s.r.l.
brandino@exact-lab.it

Panagiotis Koutsourakis, Joeri van Ruth,
Ying Zhang and Martin Kersten
MonetDB Solutions
{name.surname}@monetdbsolutions.com

***Abstract*—The ExaNeSt project started on December 2015 and is funded by EU H2020 research framework (call H2020-FETHPC-2014, n. 671553) to study the adoption of low-cost, Linux-based power-efficient 64-bit ARM processors clusters for Exascale-class systems. The ExaNeSt consortium pools partners with industrial and academic research expertise in storage, interconnects and applications that share a vision of an European Exascale-class supercomputer. The common goal is designing and implementing a physical rack prototype together with its cooling system, the non-volatile memory (NVM) architecture and a unified low-latency interconnect able to test different options for network and storage. Furthermore, the consortium goal is to provide real HPC applications to validate the system.**

In this paper we describe the unified data and storage network architecture, reporting on the status of development of different testbeds and highlighting preliminary benchmark results obtained through the execution of scientific, engineering and data analytics scalable application kernels.

I. Introduction

The ExaNeSt European project (call H2020-FETHPC-2014, n. 671553) aims to develop both the system-level interconnect and a fully-distributed NVM (Non-Volatile Memory) storage together with the cooling infrastructure for a European Exascale-class supercomputer based on low cost, low-power many ARM-cores plus computing accelerators (FPGAs).

This approach is shared with other European projects with synergic goals: ExaNoDe [1] and ECOSCALE [2].

The Consortium brings together industrial and academic research expertise to develop all system components:

- a low latency, high throughput unified RDMA-based interconnect (UniMem [3], APEnet [4]) enabling the scalability of both storage and I/O bandwidth, together with the compute capacity. The interconnect

will be designed and validated using different topologies on the FPGA-based testbed to better exploit congestion-minimizing routing functions and network support to system resiliency;
- great emphasis was placed on providing low-latency inter-process communication, as expected of HPC networks. For this task, we have implemented virtualized packetizers, mailboxes and RDMA engines inside the network interface, which allow user processes to communicate bypassing the overhead of the operating system. Furthermore, ExaNeSt network packets are routed to a global virtual address, which is translated by the SMMU at the destination, thus avoiding needless data copying. In our first prototype, we can transfer small messages between applications over the network in just $1 \div 2$ μs;
- a novel distributed storage architecture based on local devices attached to the computing nodes enabling near-data computation and reducing the energy and latency of I/O traffic. In addition, we exploit the high bandwidth of the NVM-e devices by using a custom parallel file system (BeeGFS [5]);
- a set of relevant Exa-scalable scientific and big-data applications (DPSNN, GADGET-3, PINOCCHIO, MonetDB, LAMMPS) needed to efficiently tune the ExaNeSt architecture during the design phase and to benchmark and optimize the implementation of the final prototype platform;
- the packaging and advanced cooling mechanism: a key component of a system aiming to be dense and power efficient.

In Sections II and III we introduce the project hardware, in particular the layered architecture of the ExaNeSt interconnect and the structure of the distributed storage. The software stack, which includes an MPI library, is described in Section IV while the reference applications used to benchmark the ExaNeSt prototype are described in Section V. Two different testbeds, addressing respectively the network and processing sub-systems have been developed to assess the performances and to ease the early development of system software and reference scientific applications. A description of the testbeds together with that of the simulator used to analyze the network scalability are included in Section VI. Finally, in Section VI-B1 we show the outcome of the preliminary benchmarking and application porting activities.

II. ExaNeSt Interconnect

The ExaNeSt interconnect — ExaNet, shown in Figure 1 — is split into 3 main components: (i) the Network Interface (NI), implemented close to each end-node and described in Section II-A; (ii) the intra-rack network IP based on APEnet architecture, outlined in Section II-B; (iii) a novel microarchitecture to be employed as Top-of-Rack switches shown in Section II-C. The *Unit* of the system is the Xilinx Zynq UltraScale+ FPGA, integrating four 64-bit ARMv8 Cortex-A53 hard-cores running at 1.5 GHz. This device provides many features, the following being the most interesting: (i) a very

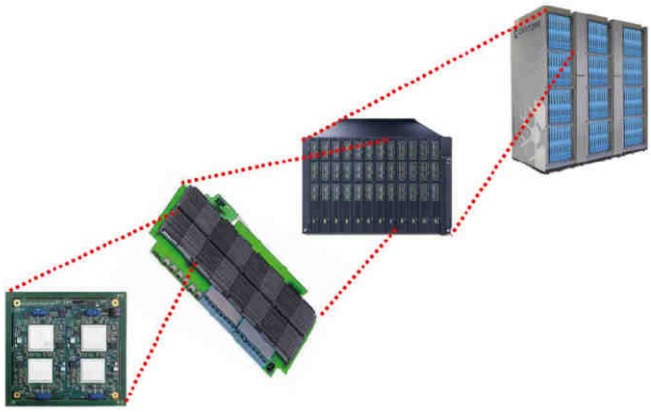

Fig. 1. The building blocks of the ExaNeSt interconnect and the different Tiers.

low latency AXI interface between the ARM subsystem and the programmable logic; (ii) cache-coherent accesses from the programmable logic and from the remote unit; (iii) a memory management unit (MMU) with two-stages translation and 40-bit physical addresses, allowing external devices to use virtual addresses and thus enabling user-level initiation of UNIMEM communication.

The *Node* is the Quad-FPGA (see Figure 2) daughterboard (QFDB), containing four Zynq Ultrascale+ FPGAs, 64 GB of DRAM and 512 GB SSD storage connected through the ExaNeSt Tier 0 network.

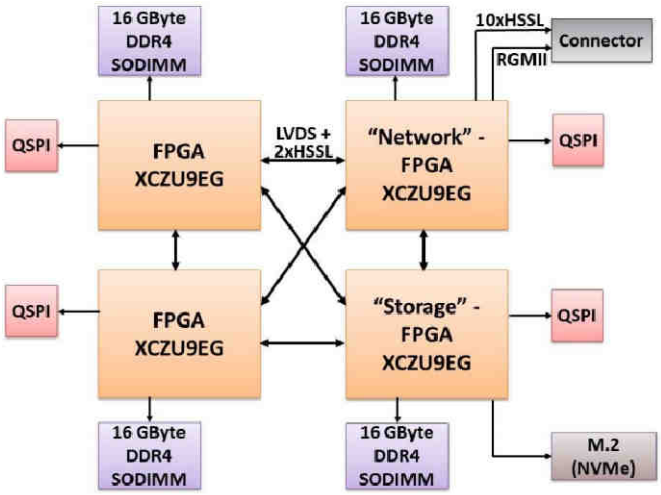

Fig. 2. The Quad FPGA daughterboard.

The inter-FPGA communication bandwidth and latency affect the overall performance of the system. As a consequence, at QFDB level, ExaNeSt provides two different networks: the first is dedicated to low-latency exchanges and is based on LVDS channels and the AXI protocol; the second is for high-throughput transmissions and relies on the High Speed Serial links (HSS) and the APEnet protocol.

For inter-node communication, the QFDB provides a connector with ten bidirectional HSS links for a peak aggregated bandwidth of 20 GB/s. Four out of ten links connect neighbouring QFDBs hosted on the *Mezzanine* (also known as Blade) (Tier 1). The first Mezzanine prototype (Track-1) enables the mechanical housing of 4 QFDBs hardwired in a 2D cube topology (a square) with two HSS links (2×16 Gb/s) per edge and per direction. The remaining six HSS links, routed through SFP+ connectors, are mainly used to interconnect mezzanines within the same Chassis (Tier 2). Furthermore, they can also be exploited to modify the Intra-Mezzanine topology.

ExaNeSt will develop two liquid-cooled prototypes — Track-1 and Track-2. Track-1 will be used to test the interconnects, storage and system software technologies developed in the project. Track-2 will allow denser racks benefiting from the new liquid cooling that will be developed by Iceotope.

Track-1 enables the safe mechanical housing of four QFDBs in a custom-made blade. Nine such blades will fit within an 11U (approximate height, the blade are hosted vertically) chassis. Thus, each chassis hosts 36 QFDBs, meaning 576 ARM cores and 2.3 TB of DDR4 memory — approximately 43 cores and 210 GB of memory per 1U of cabinet height. Finally, each Track-1 rack will host 3 chassis.

Track-2 will enable a mezzanine made of 16 QFDBs, with 6 blades fitting into a shorter approximately 8U height half-depth chassis. Thus, a "full depth" system can host 12 blades (6 blades on each side) or 192 QFDBs in 8U of cabinet height, — *i.e.* approximately 24 QFDBs, 384 cores and 1.5 TB per 1U of cabinet height. This translates to a compute density of 384 cores plus 96 FPGAs and 1.5 TB of DDR4 memory per 1U of cabinet height. Table I summarizes the Track-1 and Track-2 setup.

The Inter-Chassis (Tier 3) and Inter-Rack (Tier 4) interconnects round up the multi-tiered network. ExaNeSt adopts custom-made FPGA-based switches to support inter-chassis and inter-cabinet communications.

TABLE I
THE EXANEST TRACK-1 AND TRACK-2 OVERVIEW

	Track-1	**Track-2**
cores per blade	64	256
memory per blade [GB]	256	1024
FPGAs per blade	16	64
cores per chassis	576	1536
memory per chassis [GB]	2304	6144
FPGAs per blade	144	384
core per rack	1728	15360
memory per rack [GB]	6912	61440
FPGAs per blade	432	3840
core per equivalent 1u	∼43	384
memory per equivalent 1u [GB]	∼173	1536
FPGAs per equivalent 1u	∼11	96

A. ExaNeSt Network Interface

The network interface bridges the processes that run on the ARM cores with the communication layer provided by the network. As networks become faster and faster, there is an increasing need to offload many of the tasks traditionally performed by the network stack to smart hardware in order to reduce the software stack overheads weighing on communication latency. A smart network interface must therefore undertake a number of tasks, like transferring data to and from process memory, translating processes virtual addresses to physical memory locations, scheduling and shaping traffic, just to name a few. In addition, the NI can implement a number of primitives that help processes to communicate in efficient ways.

The network interface developed in ExaNeSt consists mainly of hardware blocks, designed for optimized inter-process communication. In addition, a number of software drivers were also developed so as to make the complex hardware mechanisms work properly for application purposes.

a) Virtualized ExaNet Packetizer: A virtualized packetizer hardware block has been developed for the network interface that allows application processes to generate and send small, latency-sensitive ExaNet packets. An application process can write the packet payload and its destination virtual address inside a configuration space in the local Programmable Logic (PL), and the hardware block creates an ExaNet packet and forwards it to an ExaNet receiver, such as a Mailbox or a switch.

b) Virtualized ExaNet Mailbox: The Virtualized Mailbox implements complementary functionalities to the virtualized packetizer. The first goal is to provide a large number of mailboxes (currently 1024) with a small FPGA area overhead. The IP hosts the mailbox buffers in coherent main memory (DRAM) and uses BRAMs only for the small metadata information needed for keeping state (base addresses, pointers and control flags). Moreover, having the mailbox buffers in main memory allows for software-configurable mailbox sizes; our implementation allows for a maximum size of 1 MByte (256 pages) per mailbox. The second goal of this IP block is to reduce the software latency of mailbox accesses. The Virtualized Mailbox leverages the AXI Accelerator Coherency Port (ACP) to deliver the packets destined to mailboxes directly to the shared L2 cache of the ARM $4 \times A53$ multicore processor. The mailbox access latency is similar to an L2-cache access ($\sim$ 20 processor clock cycles) and offers the opportunity to prefetch mailbox data in the L1-cache and even hide this small latency completely.

c) Axi-to-ExaNet Adapter: Integrating the low-power domain zDMA engine offered by the Xilinx Zynq UltraScale+ MPSoC into the ExaNet Trenz prototype is achieved through two adapter blocks developed at FORTH: one for converting the AXI transactions generated by the DMA into ExaNet packets that can be transmitted through the ExaNet network, and a second one that transforms the ExaNet packets into AXI transactions at the destination. The hardware IPs for the conversions are relatively simple and introduce a small latency — *i.e.* a couple of clock cycles. The payload of each AXI burst is conveyed over a single ExaNet packet, and each ExaNet packet is converted into a single AXI burst transaction.

B. ExaNeSt intra-rack Network IP

The ExaNet intra-rack Network IP provides switching and routing features and manages communications over High Speed Serial (HSS) links through different levels of the interconnect hierarchy: (i) the high-throughput intra-QFDB level (Tier 0) for data transmission among the four FPGAs of an ExaNeSt node; (ii) the intra-Mezzanine level (Tier 1) directly connecting the network FPGAs of different nodes within the same mezzanine; and (iii) inter-Mezzanine communication level (Tier 2) managing the connectivity of the Mezzanine based on SFP+ connectors and allowing for the implementation of a direct network among QFDBs within a chassis.

The Intra-rack ExaNet architecture is based on a layered model including physical, data-link and network layers of the OSI model.

The physical layer — **APEphy** — defines the data encoding scheme for the serialization of the messages over the cable and shapes the network topology. APEphy provides each node with point-to-point bidirectional, full-duplex communication channels to its neighbors along the available directions (*i.e.* the connectors composing the IO interface). APEphy is strictly dependent on the embedded transceiver system provided by the available FPGA. It is normally based on a customization of tools provided by the FPGA vendor — *i.e.* DC-balance encoding scheme, deskewing, alignment mechanism, byte ordering, equalization, channel bonding.

The data-link layer — **APElink** — establishes the logical link between nodes and guarantees reliable communication, performing error detection and correction. APElink [6] is the INFN proprietary high-throughput, low-latency data transmission protocol for direct network interconnect based on word-stuffing technique, meaning that the data transmission needs submission of a magic word every time a control frame is dispatched to distinguish it from data frames. The APElink manages the frame flow by encapsulating the packets into a light, low-level protocol. Furthermore, it manages the flow of control messages for the upper layers describing the status of the node (*i.e.* health status and buffer occupancy) and transmitted through the APElink protocol.

The network layer — **APErouter** — defines the switching technique and routing algorithm. The Routing and Arbitration Logic manages dynamic links between blocks connected to the switch. The APErouter applies a dimension order routing [7] (DOR) policy: it consists in reducing to zero the offset between current and destination node coordinates along one dimension before considering the offset in the next dimension. The employed switching technique — *i.e.* when and how messages are transferred along the paths established by the routing algorithm, *de facto* managing the data flow — is Virtual Cut-Through [8] (VCT): the router starts forwarding the packet as soon as the algorithm has picked a direction and the buffer used to store the packet has enough space. The deadlock-avoidance of DOR routing is guaranteed by the implementation of two virtual channels [9] for each physical channel.

C. ExaNeSt inter-rack TOR

We have designed a custom-made FPGA-based switch to support inter-chassis and inter-cabinet communications. The custom-made FPGA-based switch will route packets within Tier 3 and Tier 4 and, typically, its level of granularity will be a Chassis; this means it will rely on lower tiers of the interconnect to route packets to the specific computing element. It uses wormhole switching in which network packets are divided into multiple flow control units – called flits – whose length is equal to the channel width. In our current implementation 70 bits: 64 bits for data, 4 bits used in the MAC layer and 2 bits to control the end and the beginning of the packet. To reduce the utilization of FPGA resources and, in turn, the required power we implemented a table-free architecture, as content-addressable memories (CAMs) are known to be very costly to implement within FPGAs. In practice, this means that our custom-made switch works by performing routing decisions arithmetically based on the local and destination addresses. Furthermore, we implemented our switch with virtual output queues to reduce Head-of-line blocking during transient phases of saturation. At the moment, we are able to use our switch directly from the processing system (PS) of our Zynq UltraScale+ FPGAs by means of a simple packetizer which interfaces between AXI and our custom protocol. In our first prototype, we successfully managed to run applications and benchmarks remotely, both in a remote PS and/or FPGA.

III. ExaNeSt storage system

In a typical traditional storage subsystem (see Figure 3(a)), compute nodes usually only have main memory and maybe a boot-device; they execute application code and co-ordinate via a dedicated interconnect. Persistent data is kept separately in a parallel file system and served by a set of dedicated storage server nodes in Tier 1. In this architecture, the distance between compute nodes and data is relatively long, and even a dedicated interconnect is used for the storage traffic, it can still quickly become a bottleneck for data access.

Nowadays, the decreasing cost of low-power fast NVMs (e.g. flash-based) is dramatically changing the computing landscape. These devices promise to narrow the storage-processor performance gap with low latency (tens of microseconds vs. tens of milliseconds) and high I/O operations per second (IOPS) at affordable prices, and at capacities of several hundreds of gigabytes versus the few gigabytes of in-node DRAMs. ExaNeSt will place NVM devices directly on the compute nodes (see Figure 3(b)), rather than in a centralized location. It works as follows. Data is still stored in the protected space provided by Tier 1 file servers, but the file system is extended to consider Tier 0 devices as large persistent caches. In this way, we can shorten the common I/O path and reduce data transfers from Tier 1 servers. This architecture would be able to not only improve I/O performance, but also reduce the energy consumption associated with frequent data

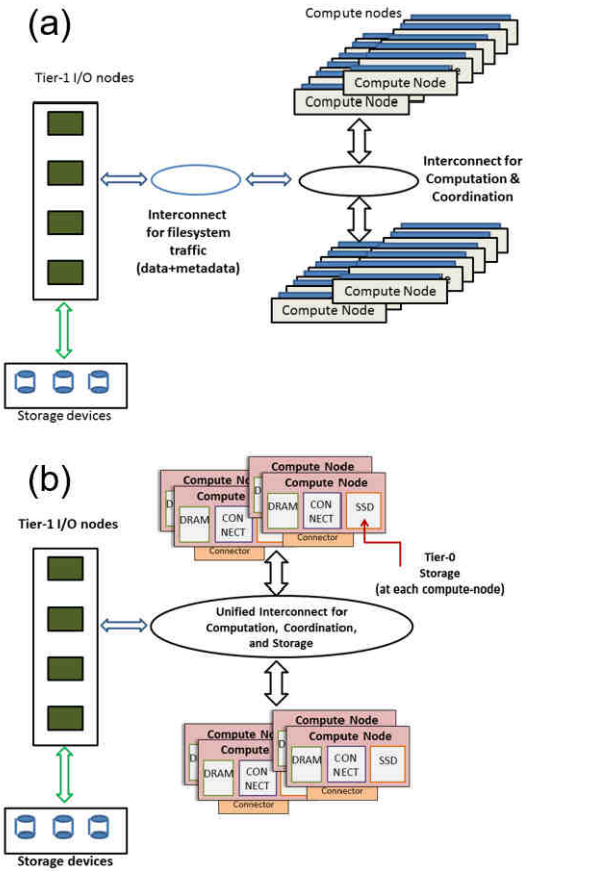

Fig. 3. (a) Traditional storage subsystem integrated through a separate interconnect with the compute nodes; (b) The ExaNeSt storage subsystem with distributed in-node NVMs (SSDs), introduced as a new caching layer, and unified interconnect.

movements to/from the storage devices. In ExaNeSt, we use the distributed parallel file system BeeGFS [5] to manage the storage devices. In addition, we have designed extensions to it to take advantage of the NVM devices as a cache layer (see Figure 4).

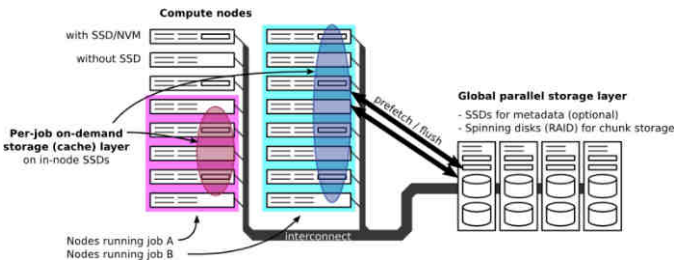

Fig. 4. BeeGFS extended into a two-tier architecture.

In this two-tier storage approach, the global parallel storage layer contains one or several big BeeGFS storage server(s) with RAIDs of spinning disks outside of an ExaNeSt rack, which serves as the permanent backing store. On the compute nodes, there is an extra storage layer which uses the integrated SSDs as a per-job cache layer and temporary/ephemeral scratch space. Each time a number of nodes with SSDs are assigned to a (computing) job, we can start a light-weight BeeGFS instance to serve those SSDs as storage for just this job. This storage can then be used either for temporary data (snapshots, intermediary results, etc.), or as a cache layer by copying data from the global file system to this "instance store" (prefetch), so that this job henceforth can access its data there, directly where the calculation is going to happen — this reduces network load, latency and the load on the global file system server. At the end of the job, the data can be copied back to the global store if needed (flush). A further optimization might be to share an on-demand BeeGFS instance with applications that only need node-local storage.

IV. ExaNeSt Software Stack

A. Low-level Software Stack

In ExaNeSt, we build a Global Virtual Address Space (GVAS), in order to allow processes to store or load data directly from (remote) process variables and data structures, without needing to copy them in intermediate system variables. The ExaNeSt global virtual address space is addressed by 80 bits. A valid address identifies a global (variable) location of data. The following fields of the global address space are used as follows:

- The Protection Domain ID (PDID) identifies the system-level process group to which the data belong. At each node, there is at most one process per PDID.
- The Destination Coordinates ID (DCID) identifies the node at which the data reside.
- The Destination Memory Address (DMAdd) identifies a virtual or physical address of the process belonging to the group PDID and running at node DCID.

With UNIMEM, in order to enable a global address space, each "coherence island" (the four A53 processors inside an FPGA) has to allocate disjoint regions of physical memory to local memory and peripherals, and to the external world. This exposed memory is in the form of an address "window" which enables direct memory and I/O accesses to remote islands.

Various communication protocols (AXI, Ethernet, APElink, etc.) can be used to address this remote world, while a translation mechanism provides a mapping (either static or dynamic) between the island physical address space and the global address space. In the ExaNeSt prototype, the Zynq Ultrascale+ MPSoC will support a 448 GB memory window, i.e. more than 16 GB of DRAM memory for 16 "coherence islands".

In order to perform zero-copy user-level RDMA transfers between different boards using Global Virtual Address Space, we have to configure the SMMU on the sender side (initiator of DMA) as well as on the receiver side. Therefore, we have implemented an SMMU Virtualization driver, which enables the processes that initiate RDMA transfers to perform the operations needed for the translation of virtual addresses to local physical addresses. RDMA functionality is implemented using the low-power domain zDMA engine provided by the Xilinx Zynq UltraScale+ MPSoC.

User-level initiation of remote operations eliminates the need of context switch to kernel mode and thus reduces dramatically the operating system overhead. Regarding the use of RDMA, instead of using the Linux RDMA API, we set up the chosen channel's registers and then trigger the transaction. In order to perform user-level initiation, we have to memory map the registers to the process that will perform the remote operations using the mmap() system call. As arguments to the mmap we need to pass the physical page that contains the registers of the chosen channel and the file descriptor of our SMMU driver that will remap the physical base address to some virtual address at process page table using the kernel's remap_pfn_range call. After that, we can access the channel's registers as standard process memory, *i.e.* setting registers and polling for the end of the transfer.

As already mentioned in Section II-A, a new virtualized packetizer was developed in the ExaNet-based prototype along with its pairing block, *i.e.* virtualized mailbox. The functionality of these two hardware blocks was integrated into the kernel through a driver-module developed at FORTH. The functionality of the driver is used only for the allocation and the deallocation of the hardware resources, which are infrequent operations. For the frequent operations, such as sending and receiving messages, the kernel is completely bypassed.

To allow user-space code on the AXI Trenz-based prototype to initialize zDMA-based RDMA operations, a user-space library has been developed at FORTH. A core notion within this library is the communication channel which consists of two main elements: (i) a zDMA channel, (ii) a protection domain in the SMMU. The protection domain guarantees that remote virtual addresses are translated according to the correct page table of the protection domain process. For different processes to communicate they must allocate the same communication channel.

B. MPI Library

The ExaNeSt MPI library is a partial implementation of the MPI standard. It currently implements only the point-to-point primitives, while the rest of primitives are delegated to a slightly modified MPICH implementation that relies on TCP sockets. The MPICH modification required by our MPI library is just exporting the "context id" related to a communicator, which is 16 bits wide and is thus more suitable to fit in an atomic message of 256 bits — this is the current atomic message size supported by the packetizer hardware. In order for each MPI process to be able to participate in point-to-point based message exchanges, it needs to allocate the following:

- Virtual mailbox
- Virtual packetizer
- Communication channel (DMA)

All of them are allocated through the ExaNeSt user space library. The packetizer is used to pack and issue an atomic message of up to 256 bits. The virtual mailbox is the block that receives atomic messages. The protection domain that is part of the communication channel guarantees that remote virtual addresses are translated, according to the correct page table, to the protection domain process. Two or more processes of the same application that run on different nodes share a Global Virtual Address Space and belong to the same protection domain. In order for these processes to communicate with RDMA transfers in the GVAS and use the packetizer and mailbox, they must allocate the same communication channel and thus participate in the associated protection domain. The current point-to-point protocol used is rendezvous-based. Support for an eager protocol, where no synchronization is required between the sender and the receiver for a message exchange, is part of ongoing work. Message exchanges are initiated by the sender and are matched at the receiver. Virtualized packetizers and virtualized mailboxes are used to send and receive control data, respectively. For the current implementation, control data refers to the MPI message envelope and the acknowledgment that will be mentioned in the rest of this section. The actual RDMA data transfer is performed through the ZDMA engine. The sender packs the MPI message envelope using the virtual packetizer acquired and sends it to the mailbox of the peer process. Note that the peer process can either be running on the same node or on a remote one. After matching has taken place at the receiver, it performs an RDMA read operation to fetch the data from the virtual address provided by the sender (part of the message envelope received in the virtual mailbox). Using the zDMA-related functionality provided by the ExaNeSt user-space library, the receiver can determine the completion of the transfer and use its virtual packetizer to send an acknowledgment back to the sender. Finally, it is worth pointing out that our MPI library is thread-safe and thus, it can support an MPI_THREAD_MULTIPLE thread level.

V. ExaNeSt Reference Appplications

A. DPSNN

The Distributed and Polychronous Spiking Neural Network (DPSNN) simulation engine is a natively distributed mini-application benchmark representative of plastic spiking neural network simulators, coded as a network of C/C++ processes interfaced using a pure MPI implementation [10] [11].

Each process describes the synapses in input to a cluster of neurons with an irregular interconnect topology, with complex inter-process traffic patterns broadly varying in time and per process. It has been designed to be natively distributed and parallel, with full parallelism exploited also during the creation and initialization of the network.

DPSNN is based on a mixed time- and event-driven engine, dedicated to the simulation of the dynamics of networks of point-like neurons and synapses, optionally including synaptic spike-timing dependent plasticity (STDP). During each millisecond of simulation, the network evolution is the result of the dynamic of each single neuron, computed at sub-millisecond time resolution. Such a simulated neural activity generates spikes with target synapses distributed both in the same process and in other processes. The exchange of spike messages (*i.e.* the "axonal-spikes" composed of the identity of

the source neuron and the original time of emission), are the inter-process traffic payload.

Speed-up measures and strong and weak scaling analysis have been performed to demonstrate the scalability of simulations run over MPI processes ranging from 1 to 1024. To address this issue, we ran a number of simulations of spiking neuron networks organized in bi-dimensional grids of modules, each module aiming at modeling a cortical column, including up to 50 G synapses connecting 46 M point-like neurons (Leaky Integrate and Fire with Spike Frequency Adaptation) distributed over a large set of MPI processes. The execution platform was a server composed of up to 64 dual-socket nodes, each socket was equipped with Intel Xeon Haswell E5-2630 v3 processors (8 cores @ 2.40 GHz clock).

B. GADGET-3 and PINOCCHIO

In Astrophysics, the scientific problem under investigation often involves complex physics, a very large dynamical range, or both. This kind of computations requires high resolution, that translates in a very large number of computational elements — usually particles.

In the ExaNeSt project we use the TreePM+SPH code GADGET-3, an evolution of the public code GADGET-2 (GAlaxies with Dark matter and Gas intEracT) [12], and PINOCCHIO semi-analytical code [13]. In simulations used in ExaNeSt, besides gravity and hydrodynamics, the following processes are modeled: radiative cooling of the gas, star formation, chemical evolution, energy feedback from exploding stars, uniform time-dependent UV background, evolution of black holes and their energy outputs. These numerical computation are dynamical in space and time: the computation evolves from an initially homogeneous, easy to balance configuration, to an extremely non-homogeneous one. Furthermore, galaxies move with respect to each other, possibly colliding and clustering together. This behaviour makes load-balancing a severe issue and those codes an excellent and challenging candidate to design and optimize the interconnect of a supercomputer.

To test the environment and to verify our porting, we are using two different cosmological simulations. The first one is a portion of the Universe, a cube having a side of 25 Mpc, with relatively low resolution (details in [14]). This simulation does not resolve the internal structure of galaxies and is thus relatively easy to balance in all of its phases. The second one follows the birth and the evolution of a single galaxy. Here, the resolution is higher and the dishomogeneity towards the end of the simulation is larger (details in [15]). In neither case we run the full simulation, which would be very time-consuming.

C. LAMMPS and miniMD

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical molecular dynamics simulation package, capable of simulating large number of atoms and molecules, with a large variety of potential and post-processing tools. Parallelization is supported in both shared memory (OpenMP) and distributed memory (MPI) paradigms as well as hybrid. GPU acceleration is supported via both CUDA and OpenCL. LAMMPS has been successfully ported to ARM CPUS and run on ExaNeSt prototype using the RDMA engine. Benchmark simulations were executed up to several millions of atoms, using both Lennard-Jones, Coulomb and CHARMM potentials. These preliminary benchmarks show that the RDMA available through the UNIMEM logic is capable of reducing communication time up to 40%, which corresponds to a total wall time reduction of above 7%.

Given the availability of FPGA on the QFDB, we also explored the feasibility of porting the OpenCL kernels of LAMMPS to FPGA using high level synthesis tools. To this end, we decided to start from a mini-app extracted from LAMMPS, called miniMD. miniMD shares the same computational algorithms of LAMMPS but with a limited collection of potentials and minimal postproccessing. This allows an easier porting of the key components, in order focus on the optimizations needed to properly take advantage of the architectural specificity of the Zync Ultrascale+ MPSoC.

In particular, we initially focused on the most complex OpenCL, namely the kernel used to calculate the forces between the atoms. The kernel had to be radically modified, and the most performing strategy was to manually replicate the block which is the result of the high level synthesis of the OpenCL kernel. We then schedule the execution of the working groups on the available blocks, which are then executed concurrently. Even though the available logic on the FPGA allowed for 16 or more blocks, we saw that the speedup would rapidly saturate after 8 blocks were used, showing once more the well-known fact that main limitation of FPGA in general, and of this kind of SoC in particular, is the available bandwidth between the FPGA and the main memory. Nevertheless, the use of FPGA allows a significant benefit in performance, compared to using the ARM cores. Executing the force kernel on the 4 ARM cores (using OpenMP parallelization) requires 1.3 s, while running of FPGA requires 0.56 s, which is more than a factor 2 in speedup.

D. MonetDB

MonetDB [16] is an open-source column-store database system. It targets at supporting statistical analytical data warehouse applications. Such applications have the characteristics that data are generally write-once-read-many, updates are rare, and new data arrive at low frequencies (hours, days, weeks, etc.) in big chunks (usually in the order of GBs or even larger). Compared to HPC applications, the scope of use cases MonetDB supports is very different. For instance, MonetDB must be able to process any query that can be expressed in SQL on many data types (from numerical to strings to *e.g.* JSON). In addition, MonetDB must be able to handle extremely dynamic workloads (in terms of *e.g.* query complexity, data size and level of concurrency), while working with heterogeneous underlying hardware/software systems. MonetDB is optimised for in-memory processing, *i.e.* it performs best when both the hot data (*i.e.*, persistent data required by a query) and all intermediates (*i.e.*, data

generated for a query execution) fit into the main memory. Therefore, the best practice of getting optimal performance out of MonetDB is to i) keep I/O at an absolute minimal, and ii) have as large as possible I/O bandwidth. Ideally, one should immediately load the whole database into the main memory at the start of a MonetDB server session, and provide sufficient main memory throughout the whole session for the intermediates. In this way, MonetDB would not need any I/O's during the session, unless there are updates. Since updates usually come in batches, large I/O bandwidth and low disk latency enables lower transaction commit time. The challenges imposed by a database system like MonetDB on the ExaNeSt platform include i) performance: how can the abundantly available hardware in the system be maximally utilized, and ii) elasticity: to what extend does the platform support scalability in all directions (i.e. scale-up/down and scale-out/in), so as to capture the sudden changes in the application workloads, while minimizing energy consumption.

VI. ExaNeSt testbed description

This section describes the prototypes that are in use for the integration activities within the ExaNeSt project. Both the presented prototypes are based on TEBF0808 Trenz board featuring the same Xilinx Ultrascale+ MPSoC FPGA family chosen for the final prototype and 2 Gbytes of DDR4 memory. The differentiation in prototype architectures reflects the fact that integration efforts have been conducted incrementally at different levels of the hw/sw stack. The development of user-space libraries that expose hardware functionality (virtual packetizer, virtual mailbox, zDMA) to applications is prototype independent.

A working AXI-based prototype — described in section VI-B thus allows for progress towards this direction and for testing the SMMU and user-level, zero-copy RDMA. Further we used this prototype to port an MPI library.

In parallel, we developed the ExaNet network primitives using a Trenz-based system — described in section VI-A — which will later be ported to the QFDB prototype.

Finally we described the results obtained with a lightweight packet-level functional simulator — see Section VI-C — able to scale up to tens of thousands of nodes while still modeling accurately the different components of the network.

A. ExaNet Trenz-based prototype

The ExaNet Trenz-based prototype has the Network Interface and Intra-rack Network logic integrated into PL's design (see 5). The prototype presented in this section has been used to integrate and test proper functionality of hardware blocks.

Preliminary tests were performed to validate the network interconnect, shaping a 2×2 mesh topology to connect four boards through the two SFP+ ports available on each system. The testbed allows for validation of the ExaNet architecture at both Tier 0 and Tier 1 levels. The QFDB, composed by four FPGAs, matches perfectly with the testing platform. Furthermore, the development platform emulates the communication

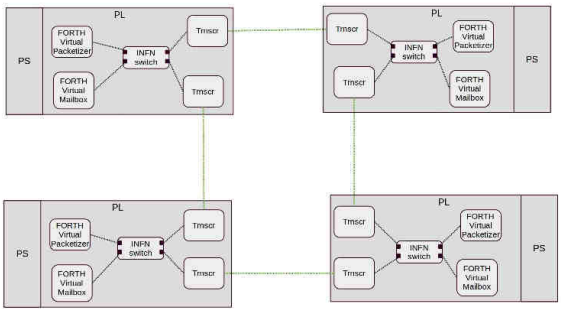

Fig. 5. ExaNet trenz-based prototype.

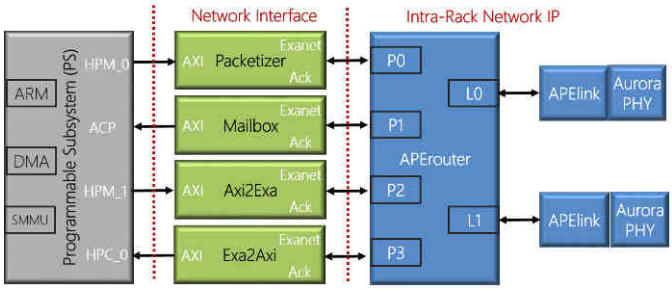

Fig. 6. Integration SoC-level design, binding the ExaNet switch with the NI primitives.

among the four network FPGAs of the QFDBs hosted within the track-1 mezzanine.

The IPs compiled inside the FPGA of the Trenz boards for the Integrated ExaNet network are listed below (see Figure 6:

- The AXI-to-ExaNet IP will take AXI traffic from master devices inside the PS through the HPM_0 PS-master port and convert it to ExaNet packets. The main source of traffic will be the DMA engines inside the PS. In this first version, the IP will be connected to an intra-tile port of the ExaNet Switch.
- The Packetizer is connected to the HPM_1, receiving AXI store commands, and outputting ExaNet packets to a second intra-tile port of the ExaNet switch.
- The Mailbox will receive ExaNet packets by the output of an intra-tile Exanet Switch port, and will output AXI transactions on the ACP port, which is connected directly to the L2 cache of the processor, without going through the CCI interconnect (and the SMMU).
- The ExaNet-to-AXI will receive the ExaNet packets by the output of an intra-tile ExaNet Switch port and it will output AXI transactions on the HPC_0 port. This PL-master AXI port goes through the SMMU and will be the last hop of DMA traffic before reaching the SMMU. Therefore, it has to pass the protection domain on the AXI ID interface.
- The ExaNet Switch provides three inter-tile and two intra-tile bidirectional ports. The inter-tile ports implement the off-chip direct connectivity, driving the FPGA embedded high-speed transceivers via a transmission control logic

block (APElink). The intra-tile ports realize the ExaNet Switch interface with the NI and PS. The intra-tile port consists of two independent FIFO-based physical queues, one for packets header and footer and one for payload. Multiple intra-tile ports will be used to setup a priority scheme.

In this first version of the integration, the AXI acknowledgements generated at the destination will be overlaid through ExaNet Ack packets and will be sent to the source.

We have implemented simple bare-metal tests that send packets with varying size from one node to another, in a ping-pong fashion, and then measure the latency. In the figure, we report the half of this latency as a function of the packet size. As can be seen, the one-hop latency is approximately 1.2 μs for small packets (smaller than 16 bytes), and increases linearly with the packet size, approaching 2 μs for 256-byte packets.

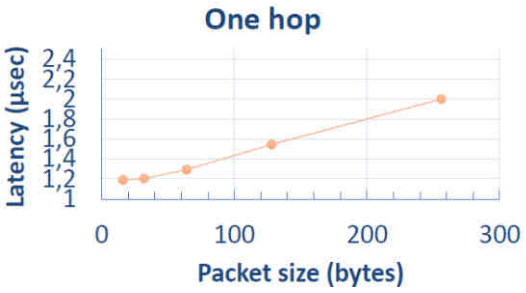

Fig. 7. One-way packet latency versus packet size for a single hop configuration.

The same test is repeated in Figure 24, but now the two communicated boards are two hops away. Effectively, packets cross a third FPGA in their journey from source to destination. As can be seen, the latency has now increased to 1.6 from 1.2 μs, manifesting that we spend approximately 400 ns per extra hop. This latency overhead is mostly due to the AURORA transceiver.

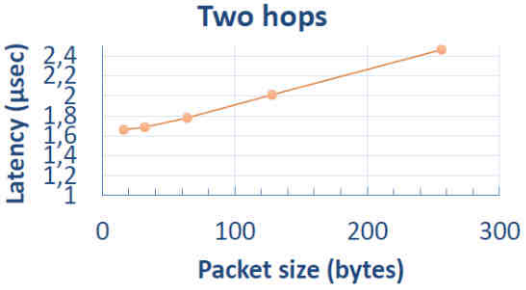

Fig. 8. One-way packet latency versus packet size for a double hop configuration.

B. AXI Trenz-based prototype

The AXI Trenz-based prototype, depicted in figure 9, consists of four nodes connected through a custom 10 Gbps switch. Each node is a TEBF0808 Trenz board, equipped with a Trenz TE0808 UltraSOM+ module.

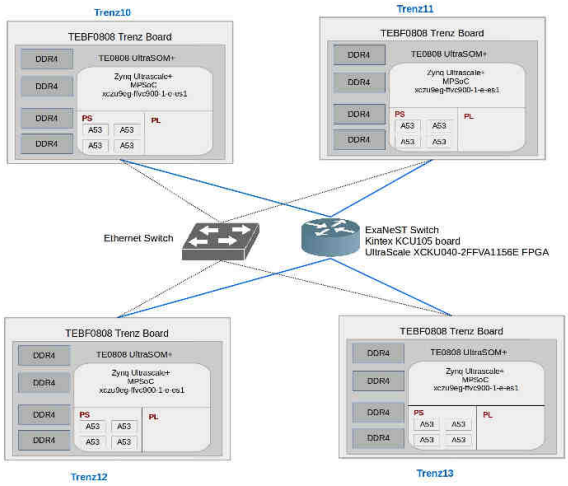

Fig. 9. AXI Trenz-based prototype.

The Processing System (PS) of the MPSoC mainly consists of 4x A53 ARMv8 cores operating at 1.2 GHz. The Programmable Logic (PL) that is tightly integrated with the MPSoC uses a clock of 150 MHz. Several hardware blocks that implement part of the functionality offered by the ExaNeSt platform are deployed into it. Our MPI library uses the services of the virtualized mailbox, the virtualized packetizer (a small message engine), and the Xilinx DMA engine (zDMA) IPs. All nodes are connected through a 1 Gbps management Ethernet network. Furthermore, each node is connected to a switch through a 10 Gbps SFP+ serial link with switching logic implemented on a Kintex UltraScale FPGA.

1) Porting and Benchmarking on the Applications testbed:

a) DPSNN: Direct porting the DPSNN application to run on the Trenz-based prototype has not required a significant effort besides sidestepping a couple of minor quirks in order to make either MPICH and OpenMPI work on an ARM boards cluster.

Of course, the data-set must be sized accordingly to fit the available memory; we note that at the moment inter-node communication has no better channel than 1 Gb Ethernet. In this regard, some reshuffling of allocations relieving the pressure the application put on the memory subsystem of the Trenz boards was necessary in order to make it run; moreover, message passing between processes was changed from variable payloads to a tunable fixed length exchange plus eventual remainder, which was found to work better for the range of sizes that form the bulk of the exchanges in a DPSNN run.

We are investigating another, more complex optimization: we envision a two-level hierarchy where, during collective communications, each process distinguishes between “local”

and "distal" peers and accordingly uses different communicators. In this way, exchanges between local peers — processes belonging to the same node — can be made to employ a different channel compared to communications between distal ones — processes belonging to different nodes — which makes available different, possibly concurrent strategies for optimization. For example, inter-node collective communication can be made to exploit the broadcast/multicast capabilities of a dedicated IP on the FPGA, just like those under investigation in the APEnet+ project [17].

In Table II we report the simulation run-times of a reference configuration (5000 simulated milliseconds of an 8×8 grid of cortical columns, 1250 neurons per column) for different layouts of cores and cluster nodes compared against those of a standard HPC platform (nodes are dual-socket servers populated with Intel CPUs — Xeon E5-2630 v2 clocked at 2.6 GHz) interconnected with an InfiniBand network interface.

TABLE II
DPSNN RUNTIMES.

# Nodes ÷ # Cores	Time (ARM)	Time (Intel)
1 ÷ 1	3044.4s	632.9s
2 ÷ 2	1530.5s	336.0s
2 ÷ 4	777.3s	181.6s
4 ÷ 8	437.0s	83.2s
4 ÷ 16	254.8s	40.7s

b) GADGET-3 and PINOCCHIO: The porting efforts for GADGET and PINOCCHIO are aimed at redesigning the codes to profit of Exascale platforms and in particular the one designed by the project where a large number of ARM cores coupled with FPGA accelerators. Actually, we ported the code as it is on the available testbed together with all the required libraries. Minor tests on scalability were performed together with some integration tests to verify that both libraries and codes were producing the same results as on Intel based clusters.

Moreover, to study the detailed network traffic produced by the code through MPI library calls, we instrument the code and run it for small temporal intervals at the beginning of the simulation, at intermediate times and towards the end, in order to sample the different dynamical situations, that can turn into different communication patterns and workload balances. In this way we collect statistics on network traffic and API usage.

Even if GADGET and PINOCCHIO are full-fledged high-performance codes for cosmological simulations, some architectural concerns arise at the edge of Exascale era that ponder on the necessity of redesigning the code to fully profit of the ExaNeSt platform. On such a system, it is crucial for codes to take advantage of spatial and temporal locality of data, which also requires NUMA-awareness. This concept translates into the "affinity" that a given memory segment has with a particular computing core. The GADGET memory model is not NUMA-aware and tries to exploit memory locality only with some basic stratagems. Hence, the redesign of the code must start from the redesign of the memory model in order to take into account the different affinities of different memory regions. Due to the extreme diversity of physical processes being modeled and algorithms implemented, this is a difficult task that does not have a unique solution and may require memory layout transformations in some points. Secondly, the code has been conceived as the parallel generalization of a serial code, in a pre-multithread era. Hence, the work-flow is rigidly procedural, with this meaning that all tasks perform the same operations — possibly individually using more threads in local loops — with frequent synchronization via MPI messages. In view of this fact, it is compulsory to adopt a different code design, *i.e.* to decompose the work-flow in as-small-as-possible single tasks with clear dependencies on, and conflicts between, each other from both the point of view of operations to be performed and data to be processed. In such a way, the work-flow would be translated in a queue system where idling threads perform the first available tasks on not-under-use data. Synchronization of operations should pass as much as possible through RDMA operations and the queue system itself.

The porting of GADGET code to the Trenz boards has been preceded by the installation and tests of some HPC libraries: FFTW, HDF5, BLAS, and GSL. Those libraries are optimized on the HW (in particular to maximally benefit the CPU and accelerators capacity, e.g. cache size and type, processors pipelines, SIMD) and has been tested on Trenz and results compared with Intel CPUs.

Given the constraints in terms of network, memory and CPUs of the current prototype, we identify a standard set of cosmological initial condition to use as a reference in any experiments that we have done and will do in the future on the other release of the platform. This experimental setup has been used (and will be used) also to test any HW and SW improvements.

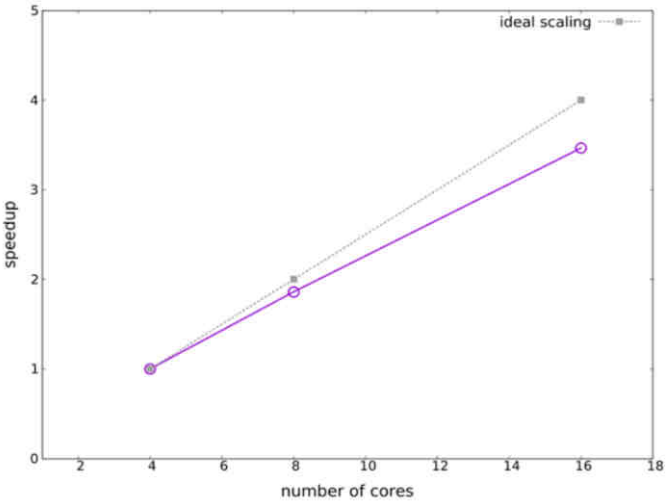

Fig. 10. weak scaling for GADGET on Trenz board.

PINOCCHIO code has been re-engineered to match the new platform of ExaNeSt. The tests presented have been performed on Trenz boards using an FFTW3 and PFFT 1.8 compiled with

the MPICH library available on the boards. Some tests have been done also to compare the FFTW and PFFT in order to verify the stability and efficiency of PFFT. We verified that actually the PFFT library is at least as much efficient, if not better, than the original FFTW library.

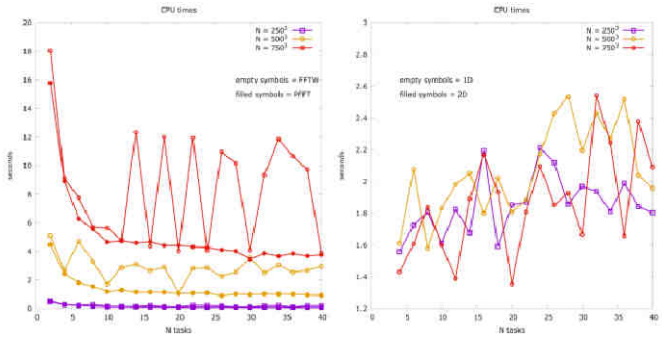

Fig. 11. The measured execution times for a FFT of an NNN cube of complex values, with N equal to 250 (violet lines), 500 (yellow) and 750 (red), with 2 to 40 MPI tasks (x axis), for both FFTW (empty symbols) and PFFT (filled symbols) for a 1D decomposition (the only possible one with FFTW): the reader can easily appreciate the fact that the PFFT library is in average even better than the original FFTW.

c) MonetDB: After having ported MonetDB to the ARM64 architecture, we have conducted preliminary experiments with the Air Traffic benchmark on Trenz. Air Traffic contains 22 SQL queries on 30+ years of departure and arrival information of the US commercial flights that computes various statistics to give insights to *e.g.* airports, airlines and passengers, and to produce predictions for future flights. The main goal of these experiments is to gain basic understanding of possible effect of on-board NVMs for distributed query executions. Figure 12 shows the setting and the flow of control of our experiment. On each Trenz node, we run one MonetDB database server, so we have 1 master and 3 workers. On the master node, we also run a MonetDB client, who sends queries to the master to be executed. The master is responsible for generating distributed execution plans, sending subqueries to the workers and gathering partial results to generate final answers. The persistent data was stored in the big BeeGFS partition available to all Trenz nodes. Each Trenz node is also directly connected to a small SSD, which was used in our experiments to mimic the on-board NVMs of ExaNeSt.

Figure 13 shows the execution times of the benchmark queries. We conducted two sets of experiments: "w/o NVM" means only the RAM and HDD were used; while "with NVM" means that the SSDs were used by each MonetDB instance as an extra cache layer for transient data, so as to reduce time consuming I/O to the BeeGFS storage. These results clearly confirm our expectation that an extra fast cache layer can help improving query performance. For all queries (including q02, 07, 11 and 12), their execution times are shorter with the NVM. For some queries, the improvement is significant, *e.g.* q08, which generates a lot of intermediates. On the actual ExaNeSt platform, we expect to obtain even bigger performance gain, because we will run benchmark using larger

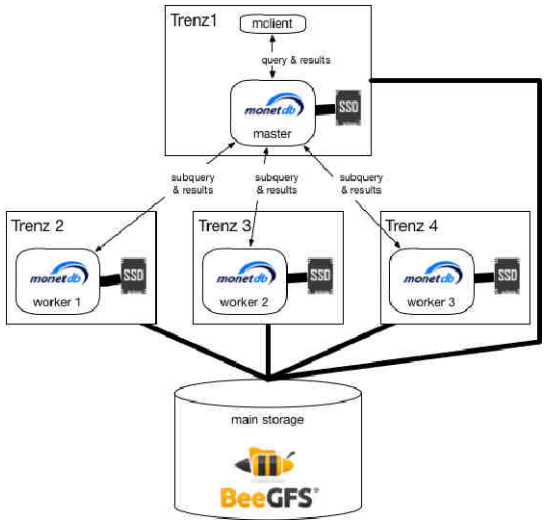

Fig. 12. The set up and flow of control of the distributed Air Traffic benchmark experiments on Trenz.

data sets and the distance to the main BeeGFS storage system is much longer.

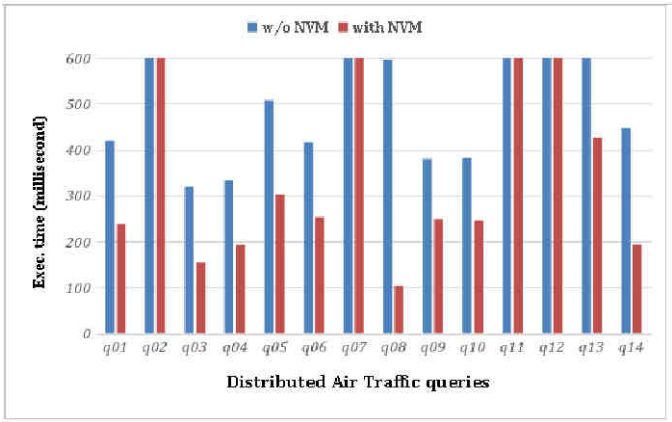

Fig. 13. execution times of Air Traffic benchmark queries distributed over 4 Trenz nodes, with vs. without using an extra cache layer. For reasons of readability, we omit execution times beyond 600 milliseconds.

C. Large-scale Network Modelling

One of the main objectives of ExaNeSt is to analyze and understand the scalability of interconnected systems connecting up to millions of endpoints. Obviously performing such an analysis with real hardware would be impossible with the resources available in ExaNeSt, so we need to rely on simulation instead.

For this reason, we are developing two different simulators: INSEE[1] [18] and INRFlow [2]. The strengths of each simulator are different: while INSEE is a flit-level simulator that uses a very detailed model of the interconnect, INRFlow is a flow-level simulator, less detailed than INSEE. This difference

[1]Available at: https://gitlab.com/ExaNeSt/insee
[2]Available at: https://gitlab.com/ExaNeSt/inrflow

determines the size of the networks that they are able to simulate. INSEE can deal with networks in the order of thousands of nodes whilst INRFlow can scale the simulations up to millions of nodes. Both simulators implement several types of networks, including trees and tori, as well as some hybridizations.

Both simulators are capable of evaluating with realistic traffic by maintaining traffic causality, either at packet- or flow-level, including traces from the ExaNeSt applications. In addition we have implemented a range of custom-made traffic generators modeling HPC applications, such as stencil-based, collectives or multi-step, as well as specific Data-driven patterns, such as MapReduce or graph-based applications.

These simulators have been used for analyzing a large number of network-related characteristics such as topological arrangement of the network elements [19], routing algorithms, the microarchitecture of the router [20], several network mechanisms (deadlock avoidance [21], prioritization, VC management, analysis of application traffic and their effect on the interconnect, analysis of storage traffic and their impact on interprocessor communications [22].

As an example of the typical utilization we show in Figure 14 an scalability analysis in which we compare a number of topologies as they contain from a few thousands up to millions of endpoints. In the plot we can see how the fattree and drangonfly seem to be the topologies that are able to scale better in terms of aggregated throughput, with very little difference among them. Then the random, Jellyfish topology seems to have the better scalability, but not be very efficient for small networks. Finally the torus topologies are found to be the ones that achieve the lowest results, with the 3D torus lacking severly in terms of performance as the system scales up.

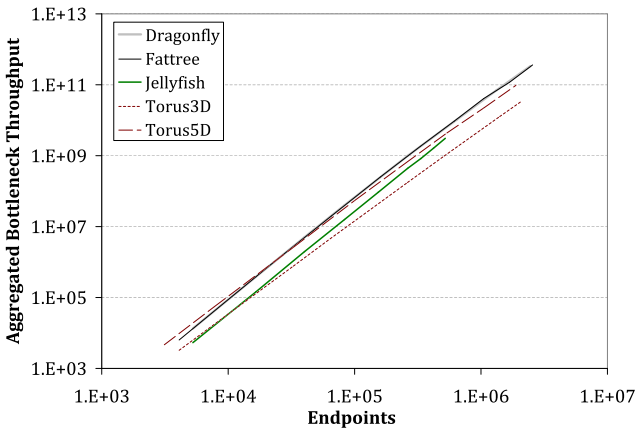

Fig. 14. Scalability analysis of state-of-the-art topologies.

Finally, Table III shows the simulated execution time of LAMMPS, one of our applications, when run over a number of networks. There we can see a large difference in terms of execution time and how that can be explained by the huge difference in terms of distance. LAMMPS is a scientific application that relies on near neighbour communication, which is very well suited for a torus architecture in which most communications are done to destinations at distance one, so there is little contention in the network. The Fattree is the next better architecture as it ensures lower contention than other networks due to the large path diversity it provides. Dragonfly seems to be the one that is the worst-suited for this specific application as it more than 3 times slower than the torus. We believe that the reason for this is that it may enforce an adversarial communication pattern which exacerbates congestion; Note that Jellyfish, which has a similar average distance as Dragonfly, is noticeably faster. In this case the random arrangement of the network reduces the likelyhood of some areas of the network to become permanently congested.

TABLE III
APPLICATION SIMULATION RESULTS - INSEE

	Exec. Time (ms)	Avg. Distance (hops)	Avg. Delay (cyles)
Dragonfly	1202.06	3.53	287.23
Fattree	611.97	3.06	199.13
Jellyfish	958.21	3.48	222.67
Torus	372.12	1.80	93.48

VII. Future work and conclusions

In this paper we discussed the general overview and preliminary results of the ExaNeSt project in terms of hardware architecture and software development. System testbeds are fully working enabling us to evaluate current network architecture and design enhanced IPs. Current results are very promising: the QFDB sub-module has been produced and it is currently under test with no major fixes required; the network testbed shows peer-to-peer small packet latency of the order of μS and new optimized hardware release is under development. Network scaling for extreme size ExaNeSt based system has been demonstrated through simulations and hardware benchmarking and applications porting activities are in progress. Next project phases foreseen the design of the enhanced Network FPGA firmware, supporting multiple independent inter-node network channels and implementing new adaptive routing functions and a sub-set of optimized collective functions. Final ExaNeSt prototype is expected at the end of 2018 and it will integrate a mesh of $\sim$ 48 QFDB modules distributed on several Track-1 mezzanines and interconnected according Torus and experimental Dragonfly topology. A complete software stack, including low level system software and MPI libraries, will be also released allowing to benchmark the system through the execution of real and optimized scalable applications.

Acknowledgment

This work was carried out with support from the ExaNeSt project, funded by the European Union Horizon 2020 Research and Innovation Programme under Grant Agreement No. 671553, and from the Human Brain Project (HBP), Grant Agreement No. 720270 (HBP SGA1).

REFERENCES

[1] "ExaNoDe," http://exanode.eu/, accessed: 2017-04-24.

[2] "ECOSCALE," http://ecoscale.eu/, accessed: 2017-04-24.

[3] M. Marazakis *et al.*, "Euroserver: Share-anything scale-out micro-server design," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 678–683.

[4] R. Ammendola, A. Biagioni, O. Frezza, A. Lonardo, F. Lo Cicero, M. Martinelli, P. Paolucci, E. Pastorelli, D. Rossetti, F. Simula, L. Tosoratto, and P. Vicini, "Architectural Improvements and Technological Enhancements for the APEnet+ Interconnect System," *Journal of Instrumentation*, vol. 10, no. 02, p. C02005, 2015. [Online]. Available: http://stacks.iop.org/1748-0221/10/i=02/a=C02005

[5] "BeeGFS," https://www.beegfs.com, accessed: 2017-04-24.

[6] R. Ammendola, A. Biagioni, O. Frezza, A. Lonardo, F. Lo Cicero, P. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, and P. Vicini, "APEnet+ 34 Gbps Data Transmission System and Custom Transmission Logic," *Journal of Instrumentation*, vol. 8, no. 12, p. C12022, 2013. [Online]. Available: http://stacks.iop.org/1748-0221/8/i=12/a=C12022

[7] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. C-36, no. 5, pp. 547–553, 1987.

[8] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[9] W. J. Dally, "Virtual-channel flow control," *SIGARCH Comput. Archit. News*, vol. 18, no. 2SI, pp. 60–68, May 1990. [Online]. Available: http://doi.acm.org/10.1145/325096.325115

[10] P. S. Paolucci, R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, M. Martinelli, E. Pastorelli, F. Simula, and P. Vicini, "Power, Energy and Speed of Embedded and Server Multi-Cores applied to Distributed Simulation of Spiking Neural Networks: ARM in NVIDIA Tegra vs Intel Xeon quad-cores," *arXiv:1505.03015*, June 2015.

[11] P. S. Paolucci, A. Biagioni, L. G. Murillo, F. Rousseau, L. Schor, L. Tosoratto, I. Bacivarov, R. L. Buecs, C. Deschamps, A. El-Antably, R. Ammendola, N. Fournel, O. Frezza, R. Leupers, F. L. Cicero, A. Lonardo, M. Martinelli, E. Pastorelli, D. Rai, D. Rossetti, F. Simula, L. Thiele, P. Vicini, and J. 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762115001423

[12] V. Springel, "The cosmological simulation code gadget-2," *Monthly Notices of the Royal Astronomical Society*, vol. 364, pp. 1105–1134, Dec. 2005.

[13] P. Monaco, T. Theuns, and G. Taffoni, "The pinocchio algorithm: pinpointing orbit-crossing collapsed hierarchical objects in a linear density field," *Monthly Notices of the Royal Astronomical Society*, vol. 331, pp. 587–608, Apr. 2002.

[14] P. Barai, P. Monaco, G. Murante, A. Ragagnin, and M. Viel, "Galactic outflow and diffuse gas properties at z $\geq$ 1 using different baryonic feedback models," *Monthly Notices of the Royal Astronomical Society*, vol. 447, pp. 266–286, Feb. 2015.

[15] G. Murante, P. Monaco, S. Borgani, L. Tornatore, K. Dolag, and D. Goz, "Simulating realistic disc galaxies with a novel sub-resolution ISM model," *Monthly Notices of the Royal Astronomical Society*, vol. 447, pp. 178–201, Feb. 2015.

[16] S. Manegold, M. L. Kersten, and P. Boncz, "Database architecture evolution: Mammals flourished long before dinosaurs became extinct," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1648–1653, Aug. 2009. [Online]. Available: https://doi.org/10.14778/1687553.1687618

[17] R. Ammendola, A. Biagioni, O. Frezza, F. L. Cicero, A. Lonardo, M. Martinelli, P. S. Paolucci, E. Pastorelli, D. Rossetti, F. Simula, L. Tosoratto, and P. Vicini, "Hardware and Software Design of FPGA-based PCIe Gen3 interface for APEnet+ network interconnect system," *Journal of Physics: Conference Series*, vol. 664, no. 9, p. 092017, 2015.

[18] J. Navaridas, J. Miguel-Alonso, J. A. Pascual, and F. J. Ridruejo, "Simulating and evaluating interconnection networks with insee," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 494 – 515, 2011, modeling and Performance Analysis of Networking and Collaborative Systems.

[19] J. A. Pascual, J. Lant, A. Attwood, C. Concatto, J. Navaridas, M. Luján, and J. Goodacre, "Designing an exascale interconnect using multi-objective optimization," ser. 2017 IEEE Congress on Evolutionary Computation, CEC 2017, Donostia, San Sebastián, Spain, June 5-8, 2017, 2017, pp. 2209–2216.

[20] C. Concatto, J. A. Pascual, J. Navaridas, J. Lant, A. Attwood, M. Lujan, and J. Goodacre, "A table-free exascalable hpc router," ser. Architecture of Computing Systems (ARCS), 2018, p. to appear.

[21] J. A. Pascual and J. Navaridas, "High-performance low-complexity deadlock avoidance for arbitrary topologies/routings," ser. The 32nd ACM International Conference on Supercomputing, 2018, p. to be submitted.

[22] J. Lant, C. Concatto, J. A. Pascual, J. Navaridas, A. Attwood, M. Lujan, and J. Goodacre, "Shared memory communication in networks of mpsocs," *Concurrency and Computation: Practice and Experience*, vol. 2012, p. Under review, 2018.