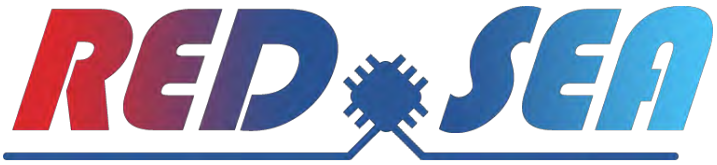
Network Solution for Exascale Architectures



D1.4: Report on holistic evaluation of RED-SEA network technologies

Document Properties

Contract Number	955776
Contractual Deadline	M36 (31/03/2024)
Dissemination Level	Public
Nature	Report
Edited by:	N. Chrysos, V. Mageiropoulos (FORTH)
Authors	All RED-SEA partners
Reviewers	Giuseppe Brandino (Exact-lab) Piero Vicini (INFN)
Date	31/03/2024
Keywords	RED-SEA Interconnect Technologies
Status	Final
Release	1.0

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955776. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Greece, Germany, Spain, Italy, Switzerland.

History of Changes

Release	Date	Author, Organization	Description of Changes
0.1	20/01/2024	N. Chrysos, FORTH	Skeleton Ready
0.9	22/03/2024	V. Mageiropoulos, FORTH	All partners completed their inputs
1.0	31/03/2024	V. Mageiropoulos, FORTH	Document ready for delivery

Table of Contents

List of Figures

List of Tables

Executive Summary

In this deliverable, we summarize the key findings of technical and scientific results in the area of interconnection networks performed inside the RED-SEA project. As we approach the end of the project, we outline the status and the most prominent results from the RED-SEA projects.

1 Introduction

As set out in the RED-SEA DoA, RED-SEA paved the way to the next generation of European Exascale interconnects, including the next generation of BXI, as follows:

- specified the new architecture using hardware-software co-design and a set of applications representatives of the new terrain of converging HPC, HPDA, and AI;
- tested, evaluated, and/or implemented the new architectural features at multiple levels, according to the nature of each of them, ranging from mathematical analysis and modelling, to simulation, emulation or implementation on FPGA testbeds;
- enabled seamless communication within and between resource clusters, and therefore the development of a high-performance low latency gateway, bridging seamlessly with Ethernet;
- contributed new efficient network resource management schemes thus improving congestion resiliency, virtualization, adaptive routing and collective operations;
- opened the interconnect to new kinds of applications and hardware, with enhancements for end-to-end network services – from programming models to reliability, security, low- latency, and new processors;
- leveraged open standards and compatible APIs to develop innovative reusable libraries and Fabrics management solutions.

Our work in RED-SEA produced innovative hardware and software Intellectual Property (IP) items tailored for BXIv3 and beyond that have been tested in BXI testbeds or with hardware platforms connecting low-power RISC-V or ARM-based cores and even NVIDIA GPUs, to BXI interconnects, via custom low-latency RDMA engines that eliminate the communication overhead from the slim processor cores.

In addition, the SAURON network simulator has been enhanced to model the BXIv3 fabric with the ability to collect and replay traces from state-of-the-art HPC applications and test them in 100K-node interconnects. The VEF tracing framework has also been used to collect traces from a plethora of HPC applications in BXI or other HPC platforms, bringing in very useful insight regarding the traffic characteristics when running state-of-the-art HPC applications. In fact, the VEF framework has been instrumental in our collaboration with other European projects that focus on software aspects and with applications, such as IO-SEA and DEEP-SEA. We also used it to evaluate congestion control schemes and optimizations for collectives using traces from actual applications. Inside RED-SEA we also simulated actual (cycle accurate) RISC-V cores through GEM5 using the COSSIM parallel simulator framework, something missing from the research community. In our tests, we simulated up to 64 RISC-V cores using and producing traces that were compared with that of ARM processors.

Network resource management is a research topic of great interest for interconnection networks. At the same time, hardware congestion control becomes indispensable in modern HPC interconnects. As this topic is relatively new for HPC clusters, more research is needed to better understand the problems and to evaluate solutions. Inside the RED-SEA, the architecture defined for BXIv3 has laid the ground for effective congestion control using both flow injection throttling, virtual channels and multipath routing. In addition, the RED-SEA academic partners delivered and evaluated novel network resource management schemes, including Weighted Round Robin (WRR) link scheduling solutions that can be used in Ethernet links (e.g., Priority Flow Control, PFC) and link power management. Accurate Congestion Control has also been evaluated on an actual hardware platform with advanced monitoring (telemetry) and workload replaying capabilities, demonstrating up to 8x lower runtime of LAMMPS in the presence of hotspots. Regarding collectives, as they are a primary contributor to congestion and HPC application runtime, novel software and hardware solutions have been proposed and evaluated in computer simulations and HPC clusters showing that they can boost performance by a factor between 2-4x.

Innovative in-network compute solutions based on sPIN that further offload the processor have been proposed and evaluated in simulation platforms. We developed a Verilog implementation of sPIN on the basis of PULP Ultra-low power processing cores which we named PsPIN and evaluated this implementation using cycle-accurate simulations for multiple workloads showing that we can process

most workloads at 200 Gbit/s line rate. Newer tests in hardware testbeds demonstrated >94% computation-communication overlap for all message size (>98% for large messages).

In parallel with significant research findings, our work in RED-SEA served as a useful vehicle for the specifications of BXIv3, by serving as a forum where ideas on architecture are being discussed as they are designed. The BXIv3 hardware engines have evolved to support per-packet end-to-end timeouts and retransmissions, off-chip and on-chip caches that allow applications to post thousands of descriptors, and to leverage a co-processor for non-critical processing tasks inside the Portals pipeline. The BXIv3 NIC adopted the Ethernet link layer to connect seamlessly with modern datacenter clusters and to benefit from the Ethernet roadmap towards Terabit/s link rates and beyond. BXIv3 further supports the Portals interface and accelerated TCP/IP paths that are crucial in several commercial deployments. ATOS has also joined the Ultra-fast Ethernet consortium to monitor and influence the evolution of next-generation Ethernet specifications.

In terms of NIC design, the BXIv3 NIC shares some design options with the EU-funded efforts for low-power NICs. Message segmentation, per-flow multi-path routing and congestion management per-flow, as well as dynamic hardware contexts of different flavors are examined in BXIv3 NIC and in caRVnet. When connected to ARM or RISC-V cores in RED-SEA testbeds, caRVnet avoids the use of the PCIe, which is studied further with BXI and APEnetX. APEnetX also proposes several optimizations for issuing small messages that greatly reduce the descriptor and data latency over PCIe. On the other hand, all NI designs support BXI as a link layer, support user-level communication, exploit a processor IO-MMU for address translations. In the case of RISC-V processors, the RISC-V IO-MMU that is developed by FORTH in the context of the eProcessor EuroHPC project has been integrated and tested inside the RED-SEA project. This activity also shows the collaboration efforts and activities between the EuroHPC-01-2019 project based on the associated collaboration agreement.

Inside RED-SEA we also designed and implemented a 100 Gb/s ETH MAC with FEC (Reed Solomon), PFC and CRC options. Many projects studying 100 Gb/s connectivity rely on MAC hardcore IPs which are available for testing in FPGAs, but these IPs cannot be implemented in ASIC and cannot be used in actual products without a fee.

Regarding the BXIv3 ecosystem, in parallel with the hardware developments, the software for next-generation BXI interconnects has benefitted from the research performed in RED-SEA. There has been extensive work both on HPC applications and on runtimes for BXI (e.g., MPI). Next, we summarize the main contributions.

- new HPC (DIAPASOM) applications and benchmarks (LinkTest) have been ported and optimized for BXIv2,
- several HPC applications from RED-SEA and other EuroHPC projects have been run to generate (VEF) trace files that were later used in computer network simulations (SAURON, DQN_SIM and COSSIM) inside RED-SEA,
- BXIv3 Ethernet driver that exposes an Ethernet interface to the system, supports standard Linux interfaces and efficiently manages IP transfer, an reduces copy and locking overheads,
- improvement of the Linux kernel module (ptlnet) that implements the Internet Protocol (IP) over BXIv2, allowing to reach 80 Gb/s in TCP/IP traffic,
- the APEnetX system library has been enhanced with user-level communication optimizations and with the use of virtual addresses; the caRVnet system library was ported and evaluated on RISC-V cores,
- The MPC MPI runtime has evolved to support *multiple* BXI NICs to further scale throughput and performance evaluation has been conducted on actual BXI platforms,
- the MPC MPI has been improved leveraging BXI features to help multi-threading,
- the Parastation MPI and PGAS library has been ported for the integration with BXI, and tested on BXIv2 HPC clusters,
- the Parastation transparent bridging capabilities have been extended to embrace BXI, allowing to bridge different clusters adopting InfiniBand and BXI interconnects along the Modular Supercomputer Architecture (MSA).

In the remainder of this deliverable, we summarize the key findings of different studies in the RED-SEA project, outlining their limitations and perspectives for future BXI interconnects and related European activities. Every partner presents one or two main contributions. To maintain brevity, each partner tried to fit contributions on a limited number of pages.

2 BXIv3 Network Interface

Throughout the RED-SEA Work Packages and tasks, ATOS has been working on several aspects of the interconnect network: Ethernet interoperability, Internet Protocol (IP) communications, flow control and congestion management, adaptive routing, transport reliability and security. We have studied architecture and design solutions, defined functional specifications, and provided prototype implementation and functional evaluation of certain Intellectual Properties (IPs).
Among these activities, we have selected two key contributions to the RED-SEA project for inclusion in this deliverable. For each of them, we will summarize and evaluate it in terms of impact, technology readiness, complexity, cost implications, assumptions. These two contributions are:

- IP (Internet Protocol) / Ethernet communications
- End-to-end reliability of the transport

2.1 IP / Ethernet communications

Our goal in RED-SEA was to develop a high-performance low latency bridging solution between the HPC network fabric and the Ethernet backbone network (see Figure 1).

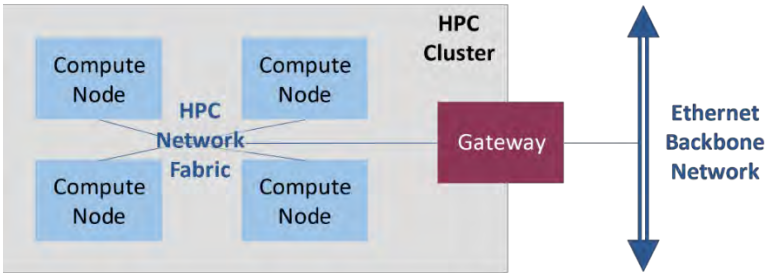


Figure 1: Ethernet bridging in a data centre.

This has been achieved with the following key architecture choices:

- **Use of standard Ethernet frames** at the physical and data link layer of the Open Systems Interconnection (OSI) model. It means the NIC is able to build standard IEEE 802.3 Ethernet frames and the Switch can route them to their destination based on their destination MAC address.
- **Definition of an Ethernet hardware device (BXIv3 NIC named NICIA)** that allows efficient transmission and reception of Ethernet frames over the network and exposes resources and interfaces to the operating system. The hardware design targets a high level of performance: 400 Gb/s bandwidth and 220 MegaPackets/second (MP/s) rate.
- **Design of an IP / Ethernet driver** that exposes an Ethernet interface to the system, supports standard Linux interfaces and efficiently manages IP transfers.
- **Use L3 Ethernet switches as gateways** to allow connectivity and interoperability with existing Ethernet devices.

In terms of realisation, a HAS (High level Architecture Specification) of the Ethernet hardware device) has been defined and reported in deliverable D2.1 and a prototype implementation of the Ethernet device has been developed and presented in deliverable D2.7. It consists of Register Transfer Level (RTL) code for an Agilex I-series FPGA, along with its verification environment. As shown in Figure 2, the main hardware components are: the IP Offload Transmission Engine, the IP Offload Reception Engine, the network crossbar and the Ethernet network interface.

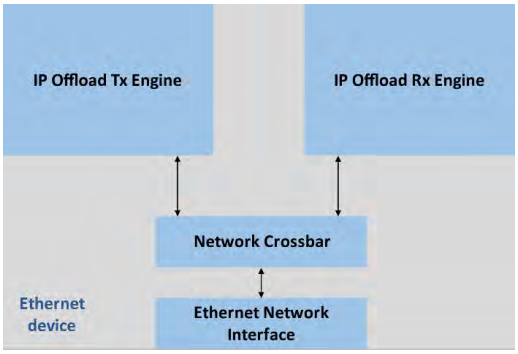


Figure 2: Ethernet device hardware components.

At the time of writing this report, the RTL code development is completed, and verification is in progress. The synthesis reaches the target of 400 MHz and hardware bring up has started.

In RED-SEA, an IP / Ethernet software driver has been designed and developed. This work has been reported in deliverable D2.6. The driver has been designed to ensure a high level of parallelism with independent resources and zero-copy mechanism in order to reach targeted performances. It supports both x86 and ARM architectures and is fully integrated in the Linux network stack and Linux administration tools. The driver tests have been conducted using an emulated device in a virtualisation environment. The driver is fully functional.

Additionally, the Ethernet device and the IP / Ethernet software have been integrated in an Ethernet bridging prototype whose goal is to demonstrate the Ethernet gateway functionality. As shown in Figure 3, the prototype is made of: a) a compute node machine with the RED-SEA Ethernet device and IP / Ethernet driver; b) a L3 Ethernet switch used as the gateway, c) a service node machine with an off-the-shelf Ethernet device. In addition, two traffic spy FPGAs have been added to capture Ethernet frames that comes in and out of the switch.

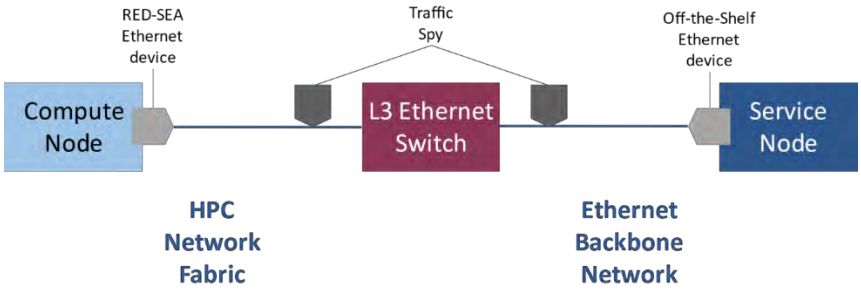


Figure 3: Ethernet bridging prototype.

The demonstrator has been used to test and validate the connectivity between the compute node in the HPC network fabric and the service node in the Ethernet backbone network. At the time of writing this report, the demonstrator uses an emulated Ethernet device in a virtualisation environment on the compute node. It will be replaced by the RED-SEA Ethernet hardware device (BXIv3 NIC with codename NICIA) when bring up activity is completed. Simple IP connectivity checks (ICMP echo request / echo response) are successfully tested.

The Ethernet bridging solution that has been designed and developed provides a significant advantage compared to the bridging solutions that are being used today in HPC data centres. The bridging functionality is no more implemented with a gateway server that uses network adapters from both protocols and that performs the translation in software, copying data in memory. The new approach offers a higher bandwidth and a lower latency for the communications across the two networks. Additionally, having switches rather than servers allows higher performance scaling and reduces investment and exploitation costs.

This solution will facilitate the interoperability of HPC network fabric with other supercomputers, storage servers, hybrid clouds or edge servers. This is an increasing requirement with Exascale data centres that are not monolithic and closed systems anymore but tends to be modular systems. The solution opens the door to inter-cluster communications, as it provides a first step toward HPC over Ethernet transfers between compute nodes of supercomputers interconnected by the backbone network.

The technical readiness level reached by this RED-SEA contribution is between TRL5 and TRL6. Some features of the technology can be demonstrated but there are still ongoing tests and potential additional features to develop (like TCP offload features).

The technology is based on a commonly used standard (Ethernet). This will allow an easier integration with other technologies and tends towards a simplification compared to proprietary solutions.

The solution is going to be integrated in the BXI (Bull Exascale Interconnect) product developed by Atos. It will be one of the two communication flows supported by the BXIv3 NIC and Switches, and it will provide a key feature for the BXI technology in term of interoperability.

2.2 End-to-end reliability of the transport

One of the main objectives of the RED-SEA is to develop the end-to-end reliability of the transport layer of the interconnect at scale. It means that with several thousands of servers connected to the network fabric, the reliability of the HPC communications between each pair of servers must be guaranteed.

This objective has been achieved by designing the transport layer of the hardware device with the following choices:

- **handle reliability at packet level**. Each message is fragmented into packets whose size is limited to a maximum of 9 Kilobytes.
- **have a separate management for each flow**, a flow being a sequence of packets targeting the same target peer and having the same attributes (request or response, order and unorder, compute and service). The design allows to manage a large number of flows in parallel.
- **assign sequence numbers to packets**. For ordered flows, packets are assigned a sequence number (SN) at transmission. This number is compared at reception with the expected sequence number.
- **track received packets with acknowledgments**. When a packet is received, a control packet is returned to the emitter to acknowledge good reception by the peer.
- **retransmit packets upon timeout**. If no acknowledgement for a packet is received after some time, this packet is retransmitted.
- **detect packet corruption with Cyclic Redundancy Check** (CRC). Packets transmitted over the wire include a CRC field for its payload, and a Frame Check Sequence (FCS) field for the entire Ethernet frame. These two values are re-computed and compared at reception.

The reliability functions have been implemented in several blocks of the transport layer of the NIC (see Figure 4 and Figure 5).

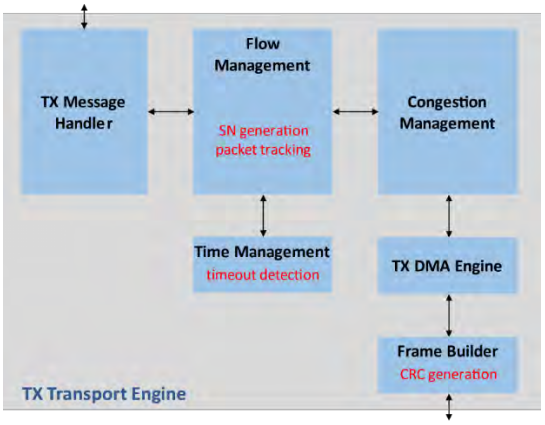


Figure 4: Reliability functions in the TX Transport Engine.

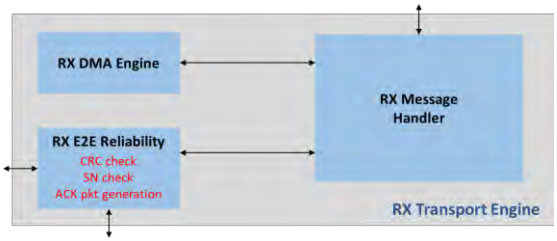


Figure 5: Reliability functions in the RX Transport Engine.

In the Transmission (TX) Transport Engine, the flow management block handles sequence number generation and packet tracking for each independent flow. The time management block takes care of timeout detection and triggers packet retransmission when needed. The frame builder block computes the CRC value and include it into the frame.

In the Reception (RX) Transport Engine, the end-to-end (E2E) reliability block handles the CRC check and the sequence number check. In case of packet data corruption or packet duplication or out-of-order packet, it drops the packet. Else it generates an acknowledgement packet that is returned to the transmission initiator.

This design has first been described in the deliverable D4.1 “End-to-end protocols and methods for reliability and protection: designs and functional specifications”. Then, it has been implemented and reported in deliverable D4.5 “End-to-end protocols and methods for reliability and protection: final designs and evaluations”. At the time of writing this report, the RTL code development and its verification are in progress.

The technical readiness level reached by this RED-SEA contribution is TRL5, since technology is still under development. It will be possible to test and evaluate the solution within an interconnect network fabric in the coming months.

This solution developed within the RED-SEA project is going to be integrated in the BXIv3 NIC. It offers a full hardware reliability of the transport layer for the Portals communications. Compared to previous BXI NIC generations, several aspects have been enhanced: fine-grained management of the transmission tracking, reduced timeout for retransmission and improved scalability within flow management.

These features will benefit to applications and data centre services that will perform their communications on a more reliable interconnect network, able to scale up to several thousands of servers. Since reliability feature is fully managed by the hardware, the software layers can focus on computing and communication, providing an optimized environment for HPC simulation and AI model training.

3 Multi-Rail MPI for BXI

A number of contributions through Multi-Processor Computing (MPC), CEA's MPI implementation has been provided by CEA during the RED-SEA project, especially focusing on the optimization of point-to-point communications for high-performance computing (HPC) applications:

1. Improved software architecture and Portals BXI driver performance improvements.
2. Multirail feature to improve performance on nodes equipped with multiple NICs.

We presented a detailed account of the strategies employed to adapt MPC for better multirail support, significantly enhancing communication bandwidth and efficiency within MPI (Message Passing Interface) environments, particularly over BXI networks. These developments act as general Proofs of Concept for later production-ready software stacks.

These enhancements are rooted in substantial improvements in MPC's software architecture, designed to manage multirail configurations through different algorithms across multiple NICs (Network Interface Cards). Within RED-SEA, we provide support for data-striping and multiplexing multirail algorithms for different configurations. In particular, we leveraged the offloading capabilities of BXI NICs to distribute message fragments. This approach aims to reduce communication overhead and optimize bandwidth usage. These technological advancements are substantiated through rigorous benchmarking tests, which confirm significant improvements in MPI communication performance, especially for systems integrated with BXI networks.

We have tested our new implementation in BXIv2 platforms using nodes with up to 4 NICs per node. Note that our implementation is based on "Portals" interface of BXI and will thus be portable to newer versions of BXI. Figures below show the maximum bandwidth (in MB/sec) as a function of message size obtained by sending a message back-to-back capabilities of the BXI NIC, and finally "*ompi*" is the reference corresponding to OpenMPI implementation.

Figure 6 showcases significant improvements in terms of bandwidth for small messages compared to previous versions of MPC when a single NIC is used which demonstrates the scaling of throughput as the number of NICs increases. In addition, both regular and offloading versions show promising performance compared to the state-of-the-art OpenMPI implementation.
Next, comparing Figure 6 with Figure 7 for large messages (> 8kB), we demonstrate **x4** bandwidth improvement when using 4 BXI NICs. thanks to the use of the new multirail feature, for both regular and offloading configurations, which demonstrates the thread optimization.

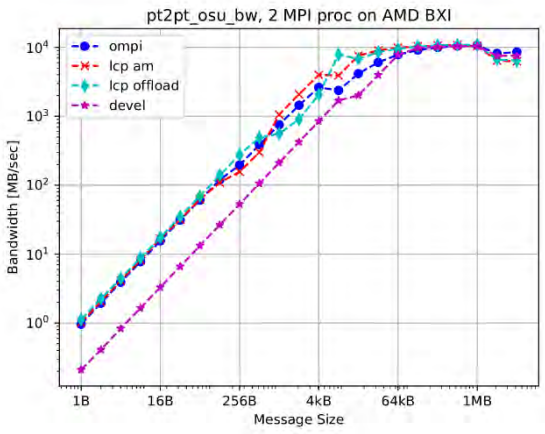


Figure 6: OSU bandwidth with one BXI NIC.

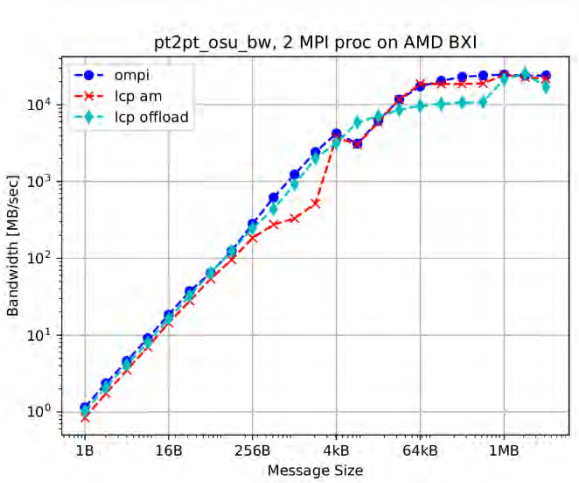


Figure 7: OSU bandwidth with four BXI NICs.

Furthermore, the adaptation of thread-based communications within the MPC framework has been performed, showcasing efforts to ensure efficient data transmission and reception through specific multithreaded communication strategies.

These efforts also underscore the potential for more developments to enhance the robustness, performance and efficiency of MPI through specific BXI offloading network features. It sets the stage for the creation of fully offloaded collective algorithms and other sophisticated features, promising a significant impact on the scalability and efficiency of HPC systems.

4 PsPIN In-network Compute in Xilinx FPGA

Throughout the RED-SEA project, ETH's focus was to produce top quality research in the area of network offload, particularly in developing an open-source implementation of a smart Network Interface Card (NIC) which follows the sPIN [1] abstract machine model. We have published 17 papers so far in the context of RED-SEA, many of them in high-impact conferences such as ISCA, SC and CCS.
During the project we changed from a sPIN interface which heavily borrowed from the Portals 4 API to an interface that is specific to sPIN and submitted it as a RED-SEA deliverable.
We developed a Verilog implementation of sPIN on the basis of PULP Ultra-low power processing cores which we named PsPIN [2]. We evaluated this implementation using cycle-accurate simulations for multiple workloads. As shown in Figure 8, we can process most workloads at 200 Gbit/s line rate.

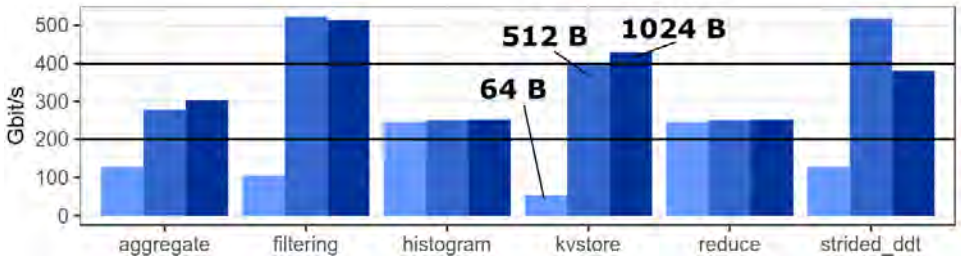


Figure 8: Throughput of different workloads on PsPIN, 22nm ASIC simulation.

At this stage of the project (we published these results in June 2021) only the sPIN specific part of the smart NIC was simulated and we relied on inputs from other RED-SEA partners to learn which workloads and communication patterns are relevant. While this technically fulfilled our minimal goals for the RED-SEA project, we saw the need to go further and build a full-system prototype. We combined the PsPIN implementation with the open-source Corundum [3] NIC implementation. Figure 9 shows a block diagram of this implementation.

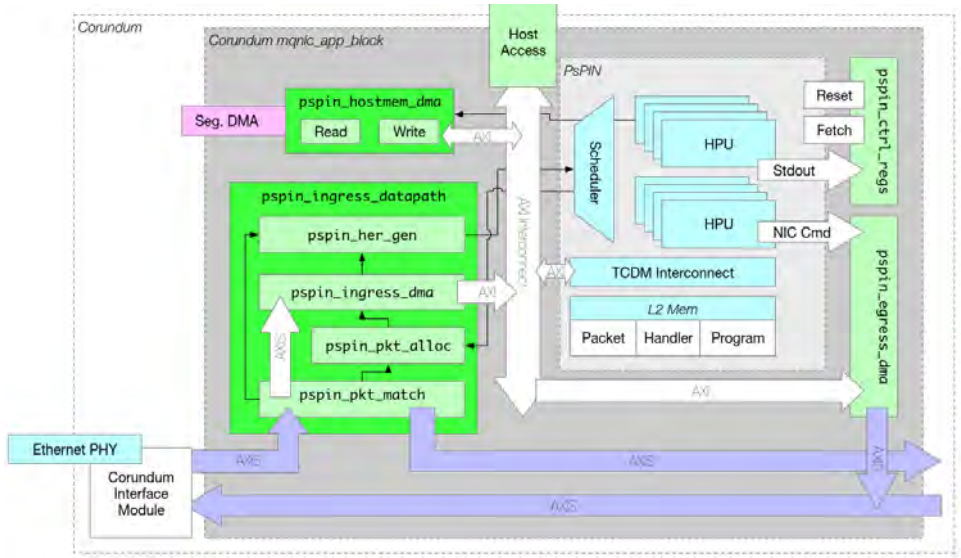


Figure 9: Block diagram of our FPsPIN Corundum Module.

A big drawback of the prototype shown in Figure 9 is the relatively low clock speed of the PULP cores when synthesizing for a Xilinx Virtex 7 FPGA of 40 MHz (compared to 1 GHz in 22nm ASIC simulation) and a relatively small number of eight cores (compared to 32 in 22nm ASIC simulation). This means we are not able to match the bandwidth numbers obtained in simulation using our FPGA based prototype. However, the full-system FPGA based prototype, which we named FPsPIN [4], allows us to evaluate the achievable overlap: the purpose of smart NICs is to lower latency and to free the host CPU of network related data processing tasks, thus we devised a benchmark where we simultaneously perform a matrix multiplication on the host CPU and unpack an MPI derived datatype on the FPsPIN NIC. We define the

overlap ratio as the ratio between the time taken for the matrix multiplication itself, divided by the matrix multiplication time combined with any polling performed by the host CPU in order to progress the MPI operation. As shown in Figure 10, we achieve 94% overlap even for small messages. For large enough messages the achieved overlap is 99%.

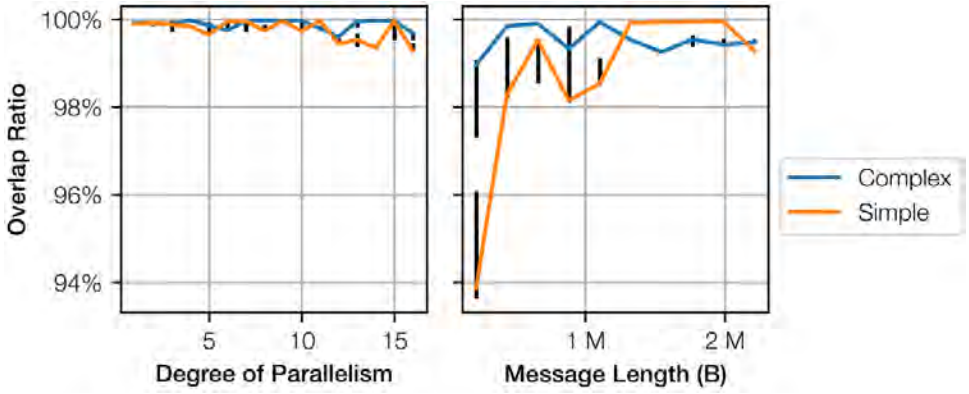


Figure 10: Overlap between matrix multiplication and two different MPI datatype receives.

The FPsPIN prototype has brought valuable insights also towards refining the sPIN specification. For example, for certain workloads it would be beneficial if the user could pin the processing of a stream of packets to a specific core, in order to exploit cache locality and avoid synchronization, while for other workloads allowing a maximal degree of parallelism is desired.

References:

[1] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, R. Brightwell: sPIN: High-performance streaming Processing in the Network in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)
[2] S Di Girolamo, A Kurth, A Calotoiu, T Benz, T Schneider, J Beranek, L Benini, T Hoefler A RISC-V in-network accelerator for flexible high-performance low-power packet processing in 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)
[3] A. Forencich, A. C. Snoeren, G. Porter, G. Papen, *Corundum: An Open-Source 100-Gbps NIC,* in FCCM'20
[4] P. Xu: Full-System Evaluation of the sPIN In-Network-Compute Architecture, in ETH Zurich Research Collection

5 RISC-V Network Simulations

Nowadays, there is a rapid advancement in the capabilities of Highly Parallel and Distributed computing systems, commonly referred to as HPC systems. These systems encompass a wide range of processing units ranging from well-known X86/ARM to modern RISC-V architectures interconnected through multiple networks. A significant challenge encountered by designers of heterogeneous systems is the lack of simulation tools capable of providing comprehensive insights beyond basic functional testing. Such insights include the actual performance of the nodes, accurate overall system timing and network deployment issues. However, in an era of complex networked HPC heterogeneous systems, simulating independently only parts, components, or attributes of a system-under-design is a cumbersome, inaccurate, and inefficient approach.

5.1 COSSIM Framework and our approach

In the context of RED-SEA we extend the open-source COSSIM framework which is designed to overcome the aforementioned constraints. The COSSIM framework effectively incorporates a collection of sub-tools that simulate the computing devices of the processing nodes, as well as the network(s) of the parallel systems. It provides cycle-accurate results by simulating the actual HPC application and system software executed on each node together with the actual networks employed. Specifically, COSSIM is built upon several established open-source packages. GEM5 is utilized to simulate the processing components of each node in the system, while OMNET++ is employed to simulate the real networking infrastructure between those nodes. In order to unify the entire framework and establish a common notion of time, COSSIM employs the IEEE1516 HLA through the open-source CERTI package.

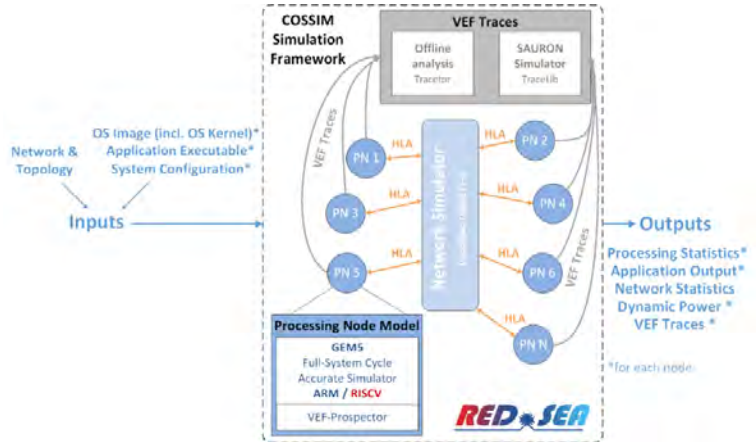


Figure 11: COSSIM Framework Extensions.

Figure 11 illustrates the COSSIM simulator with all its components, interfaces and modifications. Multiple instances of a node simulator module (i.e., a GEM5-based module) are required for the efficient simulation of the numerous processing nodes of a parallel HPC. The network that binds together the different nodes is simulated by the network simulation module (i.e., an OMNET++ based module). Initially, COSSIM supported only ARM multicore processors, while during the RED-SEA project we extended the COSSIM functionality to support multicore RISC-V processors with RISC-V PCI interconnection. Although the only network interface card that has been implemented, tested, and verified in the publicly accessible repositories of GEM5 is the gigabit Ethernet adapter based on the Intel 8254x, it does not enforce a bandwidth limit at all, and it can achieve more than 1 Gbps (up to 5.63 Gbps for ARM and 1.61 Gbps for RISC-V using 4 GHz as measured from the *iperf* network benchmark); the bottleneck is mainly the CPU on the sender side.
In order to efficiently support real HPC applications, COSSIM is extended in the context of RED-SEA to simulate the workload of the parallel program in combination with the Operating System which reflects the interaction between the parallel program and the software environment (i.e., MPI library, etc.) and real network simulator. For this reason, Ubuntu 22.04 LTS GEM5 compatible images with the modern Linux Kernels v6.5.5 and bootloaders have been created and configured in both ARM and RISC-V

architectures. Specifically, we configure an OpenSBI bootloader[1] binary and a modern Linux kernel binary that works with gem5 full system simulations in order to provide a more stable RISC-V environment for gem5 simulator. To be able to run a parallel computation on a network of computers via MPI, an MPI Cluster is set up. For this reason, the *rsh* server has been successfully imported into all gem5 nodes to be able to launch commands on remote nodes. Furthermore, MPICH v4.0 has been imported and evaluated inside both ARM & RISC-V simulated Operating Systems using both MPI simple and collective communications routines.
In addition, COSSIM is extended to communicate with other network simulators (i.e., SAURON) through VEF traces to exploit more complex network topologies and cycle-accurate processing simulations with different ISA multicore processors (ARM & RISC-V). Specifically, VEF Prospector successfully ported in both ARM & RISC-V simulated OS in order to produce VEF traces. The VEF traces can be analyzed through Tracetor application showing the number of messages recorded in the VEF trace files (providing an extensive number of plots and PDF reports, with specific information about message generation, types of collective operations, etc.). Finally, the VEF traces can be analyzed from any other simulator through TraceLib in order to model the whole interconnection network providing a set of functions for trace reading, task mapping to end nodes, and trace execution management.

The contribution of our work can be summarized in the following points:

- Development of an open-source integrated simulation framework which can simulate complete heterogeneous HPC Systems supporting Full System ARM and RISC-V architectures.
- An innovative flow to enable the designers to simulate the complete aspects of HPC Systems (i.e., CPU and Network Environment) through real MPI applications within one simulation framework.
- The integration of VEF traces Framework to analyze communication traffic of MPI-based applications and generate traces that can be used to feed other network simulator tools (SAURON simulator).
- A thorough evaluation of the end system based on real-world HPCG & LAMMPS benchmarks using both ARM & RISC-V architectures.

5.2 Evaluation Results

This section presents the experimental results which were obtained in order to validate the accuracy and scalability of COSSIM features which developed in the context of RED-SEA using the widely-used HPCG and LAMMPS benchmarks. In all experiments we use a server with AMD Ryzen 9 7950X @ 4.50 GHz with 128 GB of RAM. In our study, we create a star network topology in OMNET++, characterized by a central gem5 node (Node0) that is connected to all other gem5 nodes through an ethernet switch, thereby ensuring the centralized management of the whole simulation environment from Node0.

5.2.1 COSSIM Accuracy

In order to verify the correct behaviour and accuracy of COSSIM simulator the widely used HPCG v3.1 benchmark is ported successfully in simulated OS environment on both ARM and RISC-V architectures and compare the results with real systems; Dibona cluster for ARM and HiFive Unmatched board[2] with SiFive Freedom U740 SoC for RISC-V.

First of all, we configure gem5 to simulate from 2 up to 64 ARM cores per gem5 using Armv8 processor @2 GHz & DDR4 memory (similar to Dibona Cluster) to verify accuracy as well as compare the performance which is produced. In Figure 12 we can see the HPCG performance comparison (in GFLOPS) between the COSSIM simulator and Dibona Cluster using 2-64 ARMv8 cores @2 GHz (in COSSIM simulator). In the left y-axis we can see the GFLOPS, while in the right y-axis we can see the performance deviation and accuracy error from HPCG execution; the performance error between the COSSIM simulator and Dibona Cluster is below 5%, while the accuracy error is zero.

[1] https://github.com/riscv-software-src/opensbi
[2] https://www.sifive.com/boards/hifive-unmatched

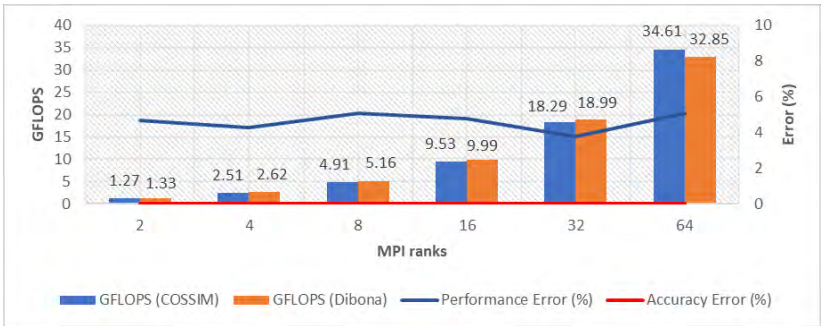


Figure 12: HPCG COSSIM vs Dibona Comparison using 2-64 ARMv8 cores.

In addition, we configure gem5 to simulate 4 RISC-V cores per gem5 with the same architectural characteristics as SiFive U740 in order to compare the performance and accuracy which are produced. Figure 13 illustrates the Performance comparison (in GFLOPS) between the COSSIM simulator and SiFive U740. As we can see the performance error between the simulated U740 in COSSIM simulator and real U740 is below 10% (8.3%) for 4 MPI ranks (U740 is equipped with a 4 core RISC-V processor), while the accuracy error is zero. In both cases (1st and 2nd column), unoptimized version of HPCG is used; the main reason of the performance error is that the HPCG has been executed for 761 seconds in real system[3], while in simulated OS only 1 second due to the huge required simulation time. In addition, we achieve 2.7x better performance efficiency (0.504 GFLOPS) on HPCG applying compiler optimizations.

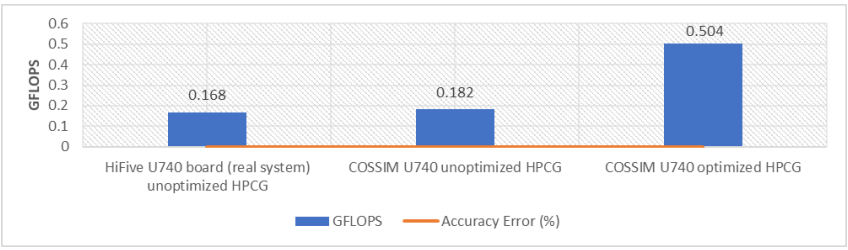


Figure 13: HPCG COSSIM vs HiFive Board comparison using 4 SiFive U740 cores.

5.2.2 COSSIM Scalability

Apart from HPCG benchmark, we port LAMMPS benchmark, which is developed and updated in the context of RED-SEA project, on COSSIM simulator on both ARM & RISC-V architectures to measure and verify the COSSIM scalability on multiple nodes. We configure each gem5 node to simulate from 2 up to 8 ARM & RISC-V cores with same architectural characteristics to compare the performance and VEF traces which are produced. Specifically, we configure each gem5 node processor on both architectures @1.4 GHz with 16 GB DDR4 memory, 32 kB L1 instruction and data caches and 2MB L2 cache size (similar to SiFive U740 SoC). Finally, in all LAMMPS experiments 10 steps with 32000 atoms have been used.

Figure 14 and Figure 15 present the Performance comparison (in timesteps/s) executing LAMMPS benchmark on RISC-V and ARM architectures with the same CPU specifications using up to 16 gem5 nodes. As we can see the ARM processor outperforms approximately 140% the RISC-V processor in all MPI ranks using the same CPU specifications in 1 gem5 node, while the accuracy error is zero. Moving to multiple gem5 nodes (Figure 15), the performance gap for ARM architecture increases from 195% to 262%. Finally, the scalability of LAMMPS benchmark ranging from 1.51x up to 1.95x doubling the MPI ranks for ARM architecture and from 1.31x up to 1.95x doubling the MPI ranks for RISC-V architecture. The main reason for the performance difference among the architectures is the bottleneck which is caused from the simulated CPU.

[3] https://www.gaborsamu.com/blog/riscv_benchmarking/

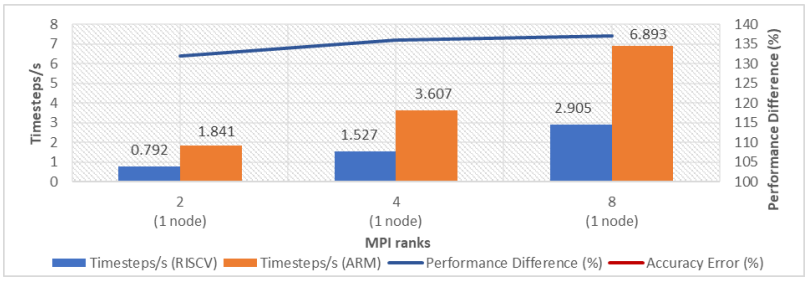


Figure 14: RISC-V vs ARM Performance on LAMMPS using 2-8 MPI ranks for 32K atoms.

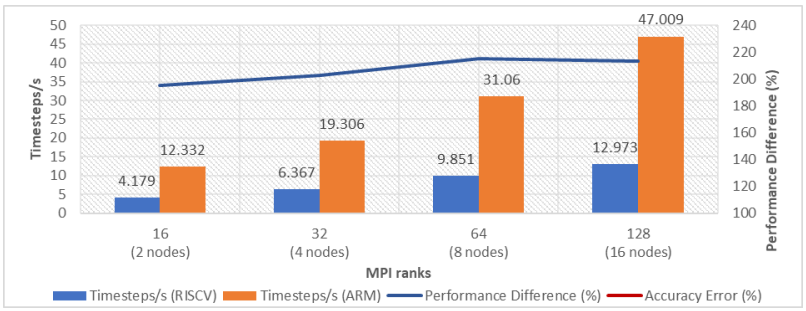
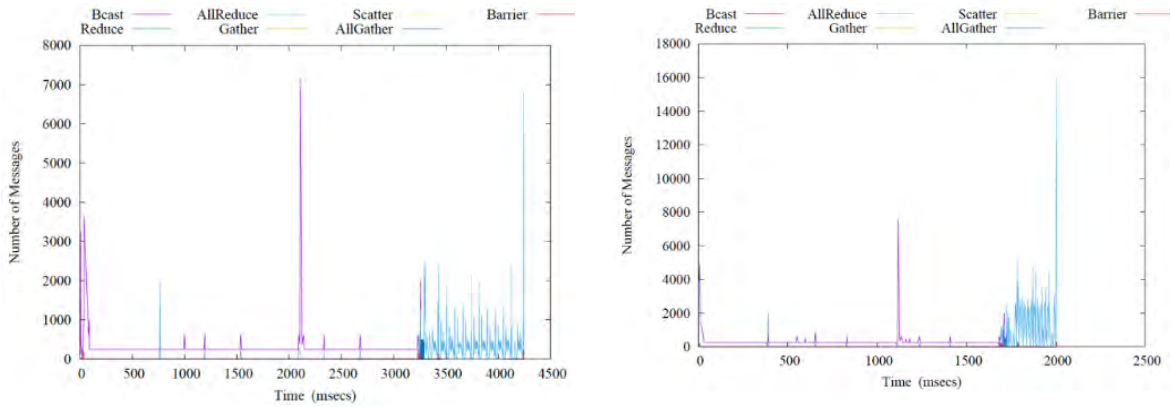


Figure 15: RISC-V vs ARM Performance on LAMMPS using 16-128 MPI ranks for 32K atoms.

In order to further analyze the communication patterns and overheads, we collect traces through the VEF traces tool to analyze the MPI communication of LAMMPS (VEF Prospector is successfully ported in COSSIM simulator). Figure 16 illustrates the evolution of collective operations over time (in ms) using 128 MPI ranks (16 gem5 nodes) on RISC-V and ARM architectures. As demonstrated from this Figure, the number of messages on ARM & RISC-V differ in their dispersion over time, while the total messages and MBs are identical. This is because the ARM processor has better performance than RISC-V (Figure 15) and it can execute more CPU instructions and MPI messages per simulated second.



(a) RISC-V *(b) ARM*

Figure 16: LAMMPS Evolution of Operation over time (ms) using 128 RISC-V & ARM cores.

6 General-purpose 100 Gb/s Ethernet MAC&PCS

Within the context of the RED-SEA project EXTOLL developed a general-purpose 100 Gbit/s Ethernet MAC & PCS logical IP block.
The IP block implements the 100GBASE-SR4 protocol as defined in the IEEE802.3 specification, which is intended for short-range optical connections. It uses four lanes, each operating at 25.78125 Gbit/s for a combined bandwidth of 103.125 Gbit/s.

The block implements the following Ethernet sublayers and IEEE clauses:

- Priority Flow Control (PFC) [IEEE 802.1Q clause 36]
- Partial MAC Control Layer (for PFC) [IEEE802.3 clause 31]
- Media Access Control (MAC) sublayer [IEEE802.3 clause 2 & 3]
- Reconciliation sublayer (RS) [IEEE802.3 clause 81]
- Physical Coding Sublayer (PCS) [IEEE802.3 clause 82]
- Reed-Solomon Forward Error Correction (RS-FEC) RS (528,514) [IEEE802.3 clause 91]
- Physical Media Attachment (PMA) [IEEE802.3 clause 83]
- Physical Media Dependent (PMD) [IEEE802.3 clause 95 - SR4]

The IP development focused on an ASIC implementation utilizing the EXTOLL serializer IP for GlobalFoundries 22nm (GF22). It will also be compatible with EXTOLL's GlobalFoundries 12nm (GF12) serializer IP, once available. The ASIC version utilizes a 128/160-bit data path at an operating frequency of 805.67 MHz, see Figure 17.
For the initially planned use within the RED-SEA project, an FPGA version targeting Xilinx FPGAs was also developed in parallel. It uses a 512/640-bit data path at an operating frequency of 201.42 MHz.

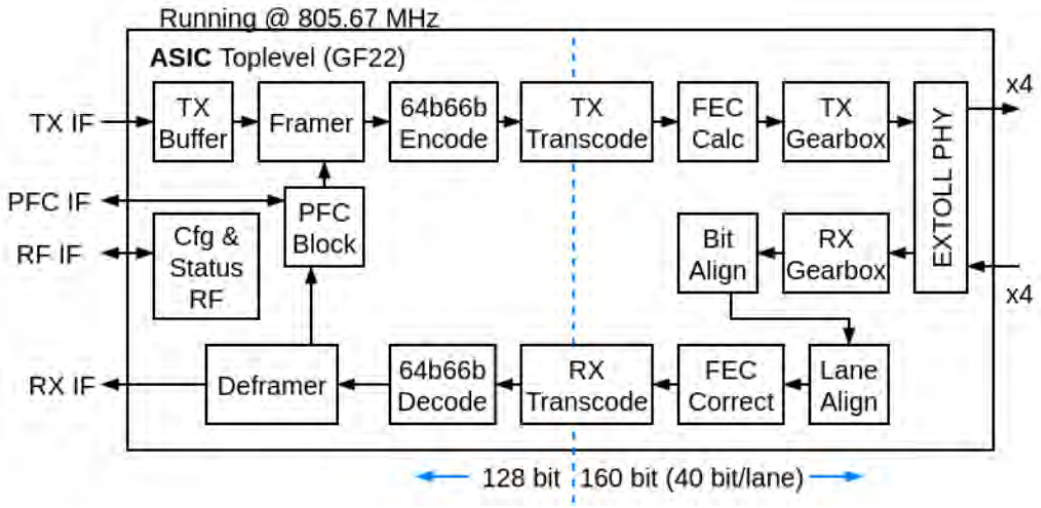


Figure 17: 100GBASE-SR4 Ethernet MAC block level diagram.

Both versions of the IP block, except for the serializer block, are written in fully synthesizable SystemVerilog code that can easily be ported to other technologies.
The IP block was verified using a testbench written in the Specman e language using the constrained random Universal Verification Methodology (UVM) framework. As golden reference the Cadence Ethernet XT Verification IP was used to verify protocol compliance. Additionally, the FPGA version of the IP block was successfully tested in-system with a Xilinx FPGA against an Ethernet endpoint.

The major functionality implemented within the IP block consists of the following:

- Configuration and Status interface accessible via a proprietary interface or an AXI4 interface.
- Transmit path functionality:

  - FIFO transmit interface with optional clock domain synchronization.
  - Preamble insertion.
  - Frame padding to 64-byte minimum frame size.
  - Cyclic Redundancy Check (CRC) checksum calculation and insertion.
  - Inter-packet gap (IPG) insertion supporting deficit idle count (DIC) approach.
  - RS-FEC codeword formatting with parity symbol calculation and insertion.
  - Insertion of PFC MAC Control frames controlled via dedicated interface.
- Receive path functionality:
  - Stripping of framing information (Preamble, CRC, FEC).
  - Bypassable RS-FEC symbol correction and removal.
  - CRC check and error indication (at end of frame due to streaming mode).
  - Configurable destination address check.
  - Extraction of MAC Control frames and handling of PFC frames.
  - PFC status information through dedicated interface.
  - Streaming-only receive interface (no store-and-forward).

The technical readiness level reached by this RED-SEA contribution is between TRL4 and TRL5, as it is still undergoing verification and testing. The FPGA implementation was tested in a lab environment against a commercially available device.

In the future this IP block can also be expanded to support backplane and copper-based standards (100GBASE-KR4 and 100GBASE-CR4), which, for full specification compliance, will require the addition of the Auto-Negotiation/Link Training (AN/LT) sublayers.

The 100 Gbit/s MAC is part of EXTOLL's IP product portfolio and aids in expanding the European IP eco system for interconnect technologies.

7 HPC Applications

7.1 LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)

LAMMPS is a classic molecular dynamic engine with a focus on material modelling. It is used widely in several branches of science: solid state physics, computational chemistry, biophysics and many others. The fact that it is used to simulate dynamics for atomic (atomic gases), meso- (large molecules such as proteins) and continuum scale (metals), makes it a perfect codesign reference tool as it is the perfect benchmark for congestion analysis and network design in real-world scenarios.
In the RED-SEA project, we ported LAMMPS to the Dibona cluster, the TGCC KNL testbed and the ExaNeSt-based Platform, where it was used to evaluate Accurate Congestion Control developed in RED-SEA. We also collected a large number of communication traces related to different configurations on both the Dibona cluster and the TGCC KNL cluster. The traces collected on the TGCC KNL are characterized by a very high number of MPI ranks, with the biggest one amounting to 32768 ranks and being the biggest trace ever collected with the tracing tool provided by UCLM.
LAMMPS has also been ported to the COSSIM network simulator and the corresponding results are reported in Section 5 and has also been used to evaluate the traffic congestion management mechanism developed by FORTH (see Section 8).
To this end, LAMMPS has been tuned to increasingly stress the network, while retaining scientifically relevant workload.

For the collection of the traces, the physical system has been sized and modelled in LAMMPS, in order to have a significant computational load and the two cases which are the most significant from an applicative point of view have been considered, i.e., strong scaling and weak scaling. The static analysis of the collected traces evidenced the behaviours described in the following paragraphs.

In the strong scaling setup trends can be observed both for the message size and the number of messages. The expected message sizes are near linearly decreasing with the number of MPI ranks since the size of data to be communicated is proportional to the volume of the domain. In fact, increasing the number of MPI tasks corresponds to linearly decreasing the volume of the domains plus some second order corrections proportional to the ratio between the skin size of the domain and its volume.
The collected results are compatible with this interpretation and the change in shape of the distribution is due to the aforementioned higher order corrections and communication optimizations.

For the number of messages, the scaling behaves differently for the communication of particle data and Fourier transform. Increasing the number of domains increases the number of messages for particle data linearly, since each domain has to communicate only with neighbouring ones.
On the other hand, for the Fourier transform each domain must communicate with all the domains on at the same x, y, and z, so, for a cubic setup with N domains per side, this amounts to $3n^3(n-1) \approx 3n^4$ point-to-point communications. This implies that going from K processes to αK MPI processes, the size of the domains is scaled down by a factor $\alpha^{1/3}$, which implies that the expected number of messages increases by a factor $\alpha^{4/3}$.
In conclusion, in the strong scaling setup, the messages are typically medium-sized and the size scales down linearly, while the number of messages scales up super linearly.
In the weak scaling setup, the range in message sizes does not change significantly with number of ranks since the size of the domains is fixed.
On the other hand, the number of messages is expected to show the same behaviour as in the aforementioned strong scaling case, since it depends on the number of domains and not on their size.

In conclusion, the static code and traces analyses suggest that the *network should be able to handle well small messages* and in a number that scales super linearly with the number of MPI processes. The FFT, needed for systems characterized by long-range interactions, represents the main responsible for the observed scaling of the message size and therefore the major bottleneck for the parallel scaling of LAMMPS as a whole.

In Figure 18 are presented the typical distributions of messages (in terms of size) for collective communications (rhodopsin benchmark). The results have been computed on the Dibona cluster and analysed with tracing library provided by UCLM.

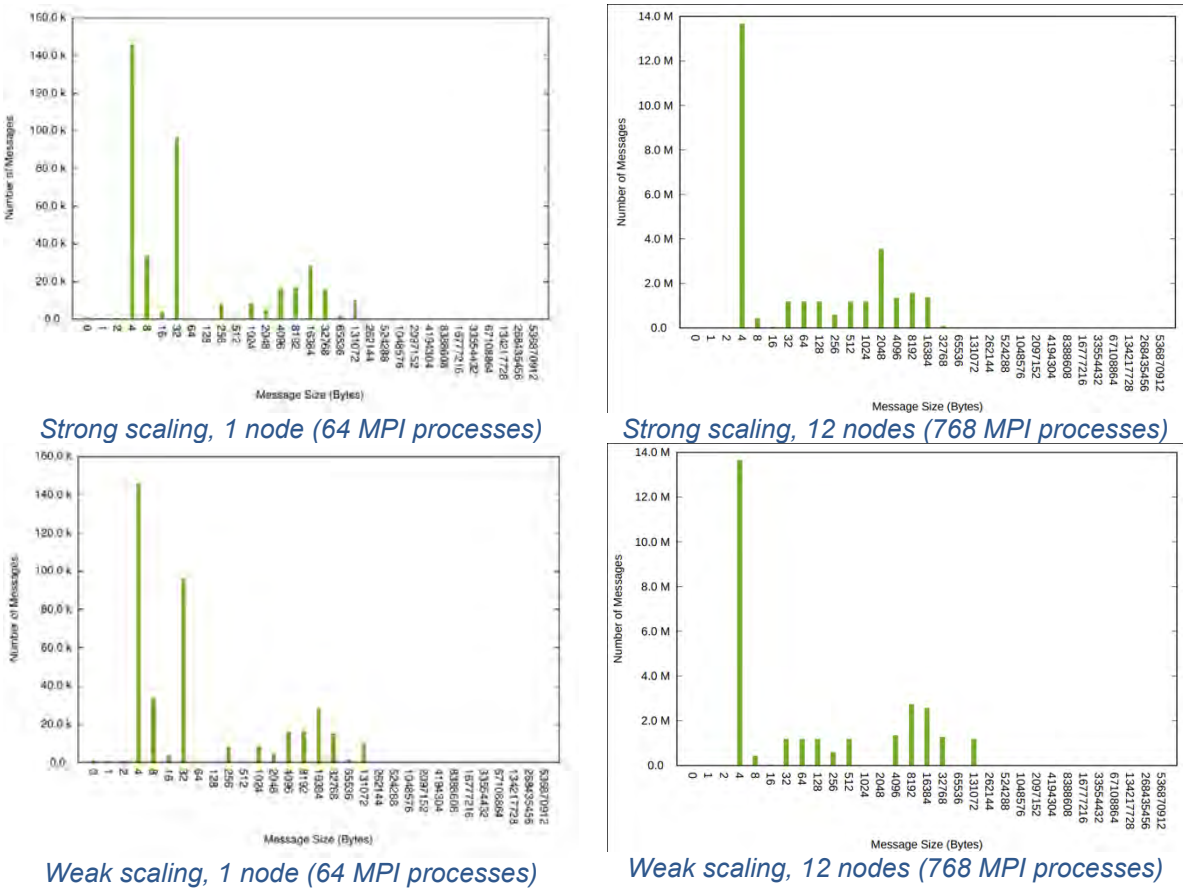


Figure 18: Collective message distribution sizes for the rhodopsin benchmark. Strong scaling setup: 2.048.000 atoms. Weak scaling setup: 32000 atoms per MPI process.

7.2 DIAPASOM HPC Application

DIAPSOM is an in-house implementation of the parallel Self-Organizing map algorithm (SOM) developed by eXact-lab and ported to the Dibona and TGCC KNL testbeds.
The application, developed from scratch during RED-SEA, has been released as an open-source package under the BSD 4-clauses license and is available at https://github.com/exactlab/diapasom.

The application (written in C++17) can be used both as a standalone executable and as a library. Four different implementations have been developed: a reference serial implementation and three parallel implementations.

The first parallel implementation is an MPI-based implementation, while the remaining two are OPENSHMEM-based, employing respectively collective ("classic") and one-sided communication. One of the reasons to develop both MPI and an OPENSHMEM versions was also to analyse the trade-off between development time and complexity of the implementation versus performance achieved: in principle, OPENSHMEM should allow faster development and design, since the management of the memory distribution is transparent to the developer.
On the other hand, MPI is the de facto standard for distributed computation, and the programming and optimization know-how is widely spread.

The development of DIAPASOM showed however how the allocation of buffers, the synchronizations and other details bring the complexity of the code to the level of the MPI implementation.

For this reason, a second OPENSHMEM implementation has been developed in which one-side communications are employed instead of collective ("more classic") communications.
In this way it was possible, at the same development complexity as the MPI implementation, to attain somewhat better performances that the ones observed with the MPI implementation.
This gain in performance with respect to MPI is related to the fact that, employing one-side communications, it is possible to overlap communication and computation (at least at the receiving end), thus reducing the overall latency.
As it can be seen in Figure 19, the gain with respect to MPI is not huge (around 5%) but it can still be considered a very good result considering the great level of maturity and optimization of the MPI libraries.

The algorithm implemented in DIAPASOM is a variation on the original "data-partitioned batch algorithm" SOM algorithm (e.g., Lawrence et al., 1999) with the aim to achieve a finer control of the training process. This was done by introducing the concept of batch training in the SOM algorithm, and thus allowing an arbitrary number of updates per epoch. performing and update after each batch has been processed. This allows, in several situations, to reach a higher quality or the results in terms of validations and, at the same time, to better control the trade-off between execution time and achieved accuracy.
An extended description of the DIAPSOM implementation has been reported in D1.2.

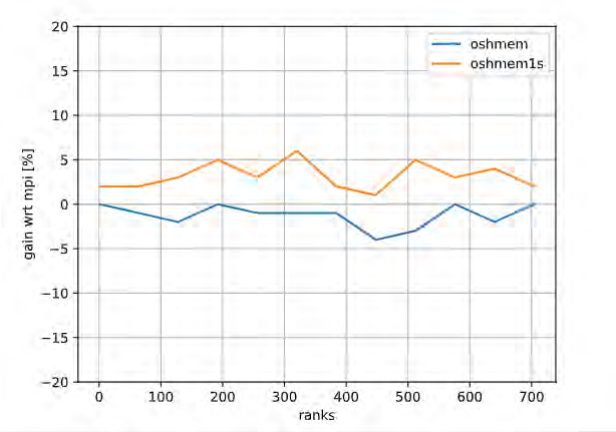


Figure 19: Gain (%) with respect to MPI implementation.

A wide range of runs, spanning many different configurations, has been performed on both the Dibona cluster and the TGCC KNL cluster. The runs, in the range 2-512 ranks on the Dibona and 2-32768 ranks TGCC KNL, showed good scalability MPI and OPENSHMEM implementations show minor differences, with no clear winner. This can be seen in Figure 19 and Figure 20.
It is worth mentioning that the speedup in Figure 20 (second subfigure: DIAPSOM speedup up to 32768 ranks on TGCC) is less optimal (linear) that the one related to smaller runs in big part because, in that situation, the DIAPASOM application had to run in a very suboptimal setup due to the fact that it was not possible to feed the application with database big enough to engage it from a computational point of view (i.e., the application had to do more communication than computation because the dataset was too small).
It was not possible to eliminate this problem since the datasets needed to properly engage the application were too big to be created and stored on the TGCC cause of the limitations on file size of the cluster.

The runs have been analysed using the VEF traces tools and the analysis software provided by UCLM and the resulting traces have been contributed to the traces repository created by UCLM and have been used to feed the simulator of traffic congestion.

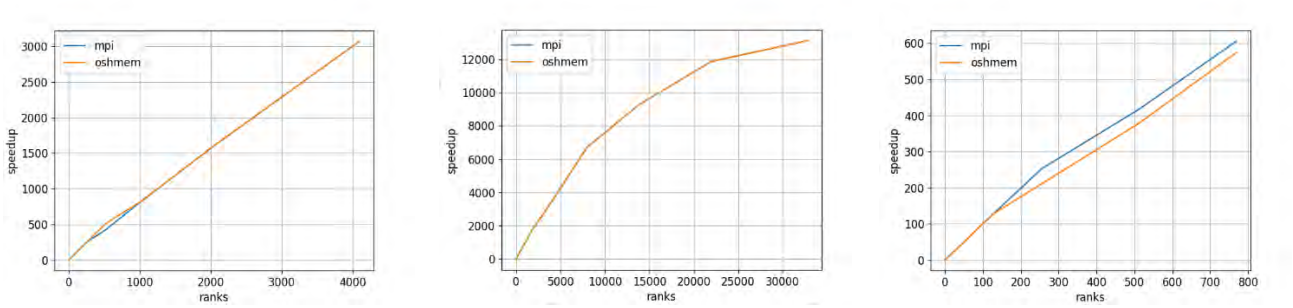


Figure 20: Speedup of sample runs of DIAPASOM on different machines: TGCC KNL (1st and 2nd from the left) and Dibona (3rd from the left). The runs were performed using 1e9 records and 100 batches on TGCC, and 1e8 records and 100 batches on Dibona.

8 caRVnet NI & Hardware Congestion Control

High Performance Computing deals with many aspects of our everyday life, including universe and climate simulations, galaxy formation, physics theory validation and others. Low-latency, high-throughput communication is a precious element of modern HPC clusters. However, efficient communication does not come for free. A common rule-of-thumb states that one 1 GHz core is needed in order to run the host software for 1 Gb/s (unidirectional) Ethernet-based traffic using sockets.

caRVnet is a lean packet interconnect that we develop and advance in RED-SEA[4], to verify and evaluate novel network technologies in actual hardware testbeds. The Network Interface of caRVnet supports RDMA transfers from user memory (at source) to user memory (at destination) completely bypassing the kernel. The caRVnet NI also offers protected virtualization through multiple descriptor channels that are implemented in on-chip SRAMs and can be allocated to different processes. With a custom user-level runtime library, processes can safely initiate concurrent transfers bypassing the kernel while the caRVnet hardware engines dynamically allocate and release contexts for RDMA transfer descriptors at source and destination. The caRVnet Network Interface breaks transfers into blocks for multipath routing, chains blocks in flows for congestion management purposes, and multiplexes the payload of transfers (possibly initiated by different users) on a per network packet granularity. A first implementation of the caRVnet NI architecture is currently installed in the full-scale ExaNeSt prototype interconnecting 12 blades, where every blade consists of four (4) Quad FPGA Daughter Boards (QFDBs), or 16 Xilinx Zynq MultiProcessor System on Chip (MPSoC). The full testbed includes 192 ARMv8 cores and runs real HPC applications using a custom MPI library.

Inside the RED-SEA project, our work on caRVnet evolved along two parallel branches. Along one branch, we implemented and optimized hardware congestion control (Accurate Congestion Control - ACC) for RDMA networks, which we tested using real HPC applications and Datacenter-inspired workloads (DAW) in a cluster of 16 FPGAs with advanced workload-replaying, monitoring and telemetry tools. Our results show that Accurate Congestion Control can minimize the flow completion time and protect the LAMMPS application runtime from other workloads that congest network links.

In the other branch, we advanced the RDMA engine with low latency and advanced QoS features, implementing also in hardware tasks previously assigned to a co-processor (such as transfer segmentation and acknowledgements). We also tightly coupled this design with RISC-V processors and with 100 Gb/s BXIv2 links, and ported optimized caRVnet libraries for RISC-V. Using microbenchmarks, we validated the hardware capabilities of the network interface, with respect to latency, bandwidth and throughput (packets per second) that can be exercised in small-packet workloads, such as key-value stores.

Until recently, running real user-level RDMA was not possible in our RISC-V caRVnet testbeds because there was no IO-MMU available for RISC-V cores. In the last months of the RED-SEA project, an IO-MMU for RISC-V processors became available from the work of FORTH in the eProcessor EuroHPC project, which we integrated and tested with caRVnet, by performing user-level RDMA with virtual addresses between two RISC-V nodes.

8.1 Hardware Congestion Control for High-Performance RDMA Interconnects

The next generation of high-performance interconnects need congestion control to deal with traffic bursts and congestive episodes that are present in HPC and data center workloads. At the same time, congestion control needs to reduce the network tail latency that can negatively affect application runtime by eliminating needless head-of-line blocking introduced once buffers fill up. Achieving these requirements in interconnects with fat links and shallow buffers requires fast and accurate congestion control methods.

[4] before that in ExaNeSt and EuroEXA, European project

Accurate Congestion Control (ACC) computes and assigns exact max-min fair rates to network flows, without relying on costly per-flow states inside the network. Instead, the mechanism relies on a simple hardware block in front of network links, with minimal latency and hardware cost overheads. ACC was first proposed in the ExaNEST EU project, where it was first evaluated using computer simulation. Then, in EuroEXA, a first hardware implementation of ACC using hardware rate limiters was reported. Preliminary results also validated that the algorithm convergences to max-min fair allocation in 5-1 incast scenarios with bandwidth-sensitive victim flow.

Inside RED-SEA, we implemented, evaluated and advanced Accurate Congestion Control in cluster of 16 Ultrascale+ Xilinx FPGAs with 64 ARMv8 cores using a mix of HPC and data center inspired workloads. We have built a long list of hardware and software tools to further evaluate and optimize ACC when running real HPC applications and data center inspired workload:
• Global Clock Mechanism, for fabric-wide telemetry. We used it to generate fabric-wide Rate Re-evaluation Pulses (20 μs the default width) needed by ACC, but also with other monitoring tools (e.g., we can align timeseries captured at different links of the fabric), which helped us identify problems and debug our implementation,
• Packet timestampers, capturing and registering at endpoints the fabric latency of individual packets,
• Traffic Generator, for flexible emulation of Data center-Inspired workloads and congestive scenarios, by launching RDMA flows with controllable start time, size, inter-transfer gap (long term rate),
• Flow Measurer, for per-flow monitoring of flow rates at μs scale on fabric internal and endpoint links,
• Flow Rate Packet (FRP) Sampler, for monitoring the dynamic fair shares of flows on fabric links according to ACC's emulation of max-min fairness,
• Multiple-Priority Crossbar, in order to isolate and prioritize control traffic (e.g., Flow Rate Packets of ACC) as well as replies from request RDMA messages,
• Transceiver FIFOs with Configurable Depth, in order to experiment with different network buffer sizes.

Furthermore, we implemented missing critical functions inside the hardware of Contention Points (Flow-init messages, Short-circuit mechanism) and identified and corrected a number of flaws in our first implementation and its integration with the caRVnet RDMA Send Engine at fabric sources.

Algorithmically, we have proposed and implemented several protocol enhancements for ACC that reduce the flow completion time of latency critical flows and their tail latency by further controlling the in-fabric backlogs during transients:

- The first one is "Wait-for-feedback", which can be applied proactively to bandwidth-sensitive or best-effort flows and constrain their source while waiting for a valid FRP-response.
- The second method, "Mind-the-gap", detects "bottlenecked-here" flows that are misclassified as "bottlenecked elsewhere" during transients, reducing transient time and buffer backlogs.
- Last, "Wait-to-rise" adopts exponential increase of bandwidth-increments after "suspicious" positive feedback, while waiting for it to be validated in the next RRPs; this method contributes to lower flow completion times by preventing backlogs, without seriously impacting background flows, as it converges to the target rate in a few RRPs.

Effectively, the combined work in RED-SEA enables us to run LAMMPS HPC application (proposed by EXACTLAB in RED-SEA), on the 1 GHz, ARMv8 (A53) 4-core processors of the ExaNeSt-based RED-SEA prototype and mix it with synthetic traffic emulating Datacenter workloads using the Traffic Generator. We have run extensive experiments in a cluster of 16 Zynq Ultrascale+ FPGAs (1 mezzanine), where we compare ACC with PAUSE-only fabrics, while studying the impact of:

- Traffic conditions: injection load, message size, transient and persistent hotspots.
- Network buffer sizes: small 4KB and large 64KB input and output buffers.
- Accurate tuning knobs: Link margin, Rate Re-evaluation Period (RRP).
- Alternative congestion control policies: PAUSE-only, FRP-only, FRP-init, Wait-for-feedback, Mind-the-gap, Wait-to-rise.

Hardware testbed results show that Accurate Congestion Control can keep the excessive traffic outside the fabric, by throttling flows to their network max-min fair share. By reserving a small headroom bandwidth, ACC frees buffers and can protect the latency and the bandwidth of victim flows. ACC allows

new flows to inject at full speed and finds the new fair share rates after short (< 60 μs) transients when flows become active or terminate. Additionally, the Mind-the-gap and Wait-to-rise protocol enhancements are particularly tailored to minimize buffer overshoots during such short-lasting transients, thus further reducing the tail latency. In experiment results with victim, latency-sensitive flows, ACC provides up to 16x lower tail in-fabric latency under low loads and 12x lower average value under medium to high loads. Also, in the presence of congestive background traffic, the LAMMPS HPC applications run up to 3.0x faster when using Accurate Congestion Control. These results realize KPI-4 "*10x reduction in tail latency under congestive episodes relative to BXIv2. Protection of latency critical traffic 2x better than in BXIv2.*"

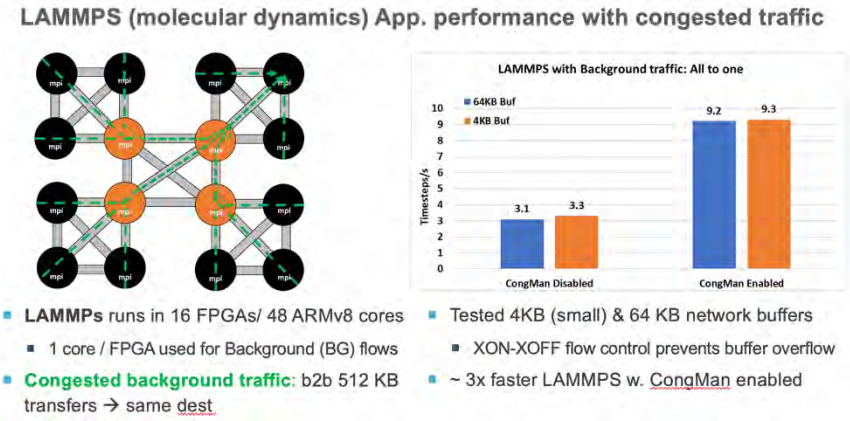


Figure 21: LAMMPS application (from EXACTLAB) running in the ExaNeSt-based cluster over caRVnet RDMA interconnect with Accurate Congestion Control and background congestive traffic.

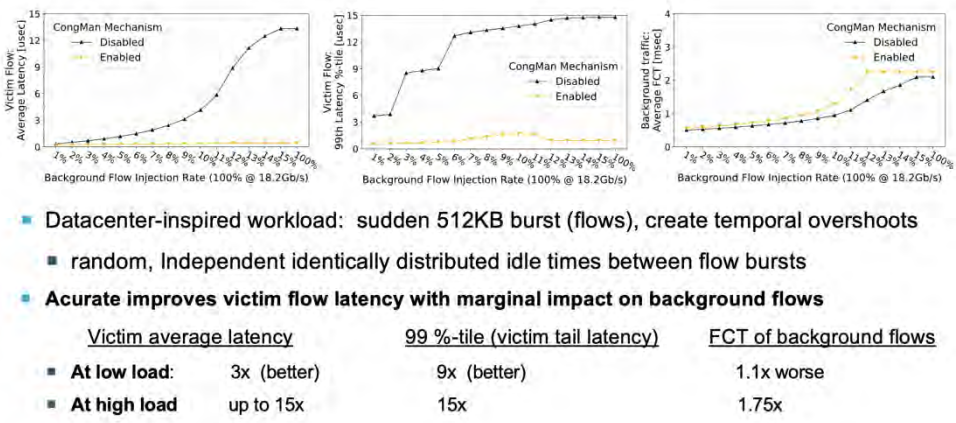


Figure 22: Experiments evaluating victim flow average and tail latency in the ExaNeSt-based cluster over caRVnet RDMA interconnect with Accurate Congestion Control.

8.2 Lean network interface tightly-coupled w. RISC-V, IO-MMU and BXIv2 links

In RED-SEA, we advanced the RDMA hardware engines for significantly better base latency and higher link rates. We also tightly coupled it with RISC-V cores, targeting low descriptor latency, and we also integrated it with the newly developed RISC-V IO-MMU from FORTH in eProcessor for virtual address translation. Finally, we made it compatible with BXIv2 Links using the BXI-Link IP from ATOS. The following list summarizes the major work items reported in deliverables D4.2 and D4.6:

1. We designed and implemented a new hardware IP responsible for RDMA transfer segmentation and endpoint acknowledgements and we used it to replace a co-processor previously responsible for these tasks. We also advanced all caRVnet NI blocks for higher throughput and lower latency.
2. We tightly coupled caRVnet a low-power RISC-V softcore achieving a throughput between the core and the NI (for descriptors) of 1 store instruction per processor cycle. We also validate that the RISC-

V core can issue a transfer descriptor to the caRVnet network interface in less than 12 processor cycles. This achieves KPI-6 "*A few (<12) cycles one-way latency from processor to the Network Interface*."

3. We ported caRVnet systems software (drivers and user-space library) from ARM-based systems to RISC-V- based ones and we optimized the library for ultra-low latency and low CPU overheads.
4. We ran ping-pong tests with the Packetizer-Mailbox user-level library between two nodes containing Ariane RISC-V soft cores running at just 100 MHz, achieving a user-level latency of 720 ns (just 72 processor cycles, 20 of which are spent in SERDES) and present a detailed breakdown of user-level communication across two RISC-V cores.
5. We evaluated the RDMA performance in a testbed of Xilinx VCU FPGA nodes with RISC-V softcore processors running at 100 MHz. The RDMA latency is now below 0.5 μs from 4 μs that was measured before using a faster 1 GHz ARM core, and the link throughput from 20 Gb/s to 100 Gb/s. We also evaluated the Packets per second throughput performance of the caRVnet RDMA finding that the hardware can reach up to 1 packet per 4 processor cycles (25 MP/s with the 100 MHz RISC-V in our prototype) using well-tuned user-level programs. These results contribute to KPI-10 *"Improve by 2x the network transactions per second in Key-Value store Benchmark*."
6. Scaling the throughput performance in workloads with small messages is a challenging task. In RED-SEA caRVnet moved to a full hardware implementation, including pipelining to process transactions at hardware clock speed, as required for high-throughput links.

Inside the **RISER** project, we continue to enhance the caRVnet RDMA-capable Network Interface. So far, we have designed a hardware engine to handle timeouts and negative acknowledgements in RDMA. We are currently working on 100 Gb/s Ethernet link interfaces and on a cache-coherent interface with a RISC-V ASIC that will become available later in the project.

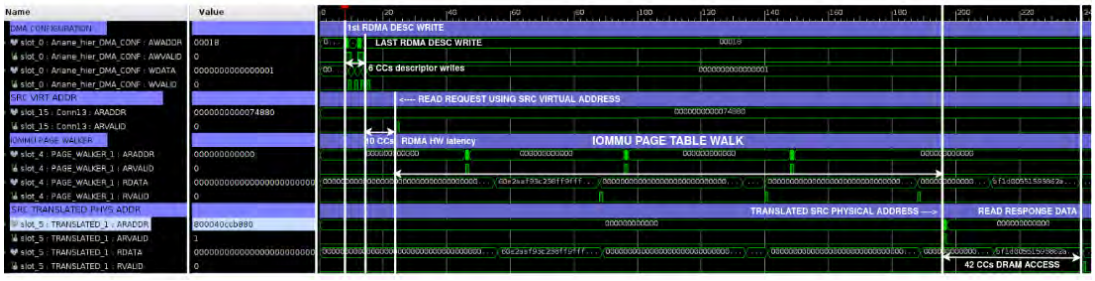


Figure 23: Ariane RISC-V core initiating an RDMA transfer to caRVnet Network Interface using four (4), back-to-back 64-bit store instructions. The HW engine generates the first read to DRAM 10 cycles later. The IO-MMU from FORTH in eProcessor is leveraged to translate the virtual address to physical, performing a page-table walk as this is the first access to a virtual page from the network interface.

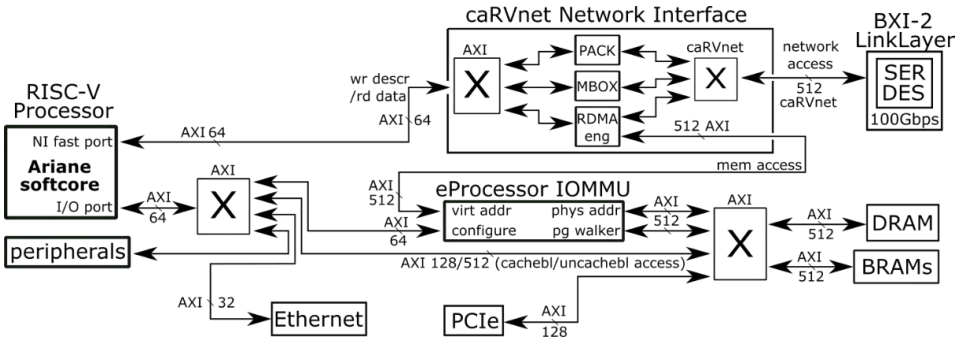


Figure 24: Xilinx VCU testbed integrating Ariane RISC-V core, caRVnet Network Interface, IO-MMU from FORTH in eProcessor and BXIv2 link. This testbed was used to produce the result shown in the figure above.

9 APENetX Network Interface and Simulations

INFN activity in the RED-SEA framework unfolds along two lines:

- adopting a co-design approach in support of the network design activities (WP1) by contributing with the NEST application. In the first phase of the project, the co-design loop is started employing applications to extract requirements and specifications of the network architecture while in the second phase the co-design iteration is closed porting the applications onto the project evaluation platforms.
- designing and implementing the PCIe-based network interface card APEnetX – prototyped on Xilinx Alveo FPGAs (WP4) to allow for low latency communication between CPU, interconnect device and GPU accelerators - and sporting a BXIv2 link layer hardware block for seamless integration in BXI environment.

In summary, INFN activities in RED-SEA focused on the development of a network interface card and related network IPs, mainly targeting the communication generated by spiking neural network applications.

9.1 Co-Design through application

We performed an extended analysis of the NEST application running on the DIBONA testbed and exploiting the VEF traces toolset to provide recommendations for the network architecture. The sampled NEST simulation describes the dynamics – called Slow Waves Activity – that the cortex of one brain hemisphere of a mouse undergoes when in a deep sleep state and whose connectome was obtained with Wide-Field Calcium high resolution imaging techniques. The parallelization scheme followed by NEST simply assigns neurons to different 'virtual processes' in a round-robin fashion, whereas a virtual process is one core of the platform the simulation is run on; if the code is launched asking for *x* MPI processes – however they may be spread across the interconnected nodes of the system – and for *y* OpenMP threads – however many may be available on the CPU architecture of the node and allocated by the user according to the scheduling policies of the system – the virtual processes are *x*×*y*, each of which computes the evolution of a corresponding fraction of the grand total of neurons.

The analysis of the simulation time and network traces highlights two main behaviors:

1. The total amount of OpenMP threads should not exceed the grand total of 32.
2. The NEST simulation could be distributed on a maximum of 64 MPI processes. Beyond this upper limit the huge number of generated messages and the reduced size significantly affect the performance of the adopted interconnect.

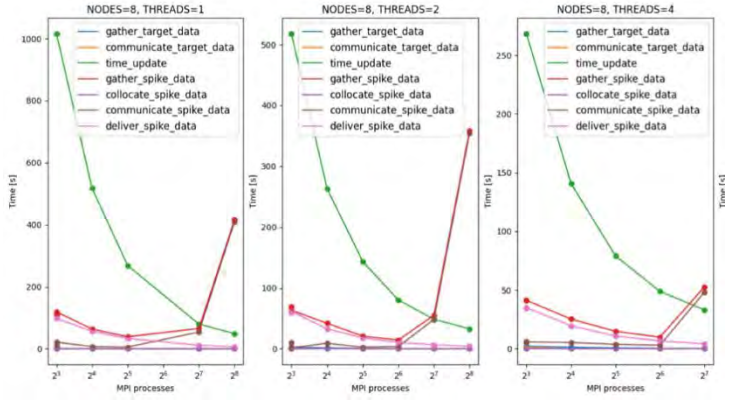


Figure 25: NEST built-in timers analysed with VEF-TraceLIB with respect to the number of processes for a fixed number of nodes and threads.

The limitation running over 64 MPI processes is analyzed in depth exploiting information contained in the built-in accurate timers of the latest version of NEST, to improve the results about the communication generated by the simulation and to confirm the results obtained using the VEF traces tool.

Focusing on the 3 cases reported in Figure 25 it is possible to notice that time gather spike data (time for complete spike exchange) decays up to 64 MPI processes and increases with the number of processes, disregarding the number of threads and nodes, becoming a bottleneck in the speedup of the system as supposed during the analysis based on the VEF-traces results.

Finally, the details of the NEST strong scaling results are reported in Table 1. As expected, distributing the application on the largest number of available nodes achieves the best performance. The simulation scales properly up to 16 cores (2 cores and 1 MPI process per node). The not ideal scaling obtained with 8 MPI process (and even worst with 32) suggests optimizations also for messages in the range of 512B – 16kB as reported in Table 1.

Number of nodes	MPI processes	OpenMP threads	ARM cores	Simulation time[s]	Ideal [s]	deviation	Speed up
1	1	1	1	2390.71	2390.71	0.00	1.00
2	2	1	2	1180.21	1195.35	-1.27	2.03
4	4	1	4	586.45	597.68	-1.88	4.08
8	8	1	8	292.39	298.84	-2.16	8.18
8	8	2	16	152.50	149.42	2,06	15.68
8	8	4	32	80.45	74.71	7.68	29.72
8	8	8	64	44.13	37.35	18.12	54.18
8	8	16	128	25.35	18.68	35.71	94.32
8	32	8	256	17.11	9.34	83.19	139.75
8	32	16	512	12.54	4.67	168.56	190.65
11	44	16	704	11.27	3.40	231.87	212.13

Table 1: NEST strong scaling results.

From a co-design perspective, in addition to the use of application traces for the network design, the described analysis, implemented by INFN during the project (scaling analysis, implementation of counters and pointers in the code), represents a useful methodology to guide the effective optimization of the code on the platform.

9.2 APEnetX: the INFN network interface card

INFN APEnetX is a low-latency and high-throughput NIC based on a PCIe Gen3/Gen4 interface designed to increase the capability of the network and compliant with off-the-shelf clusters.

9.2.1 Architecture

The NIC was prototyped on Xilinx Alveo boards. In the first half of the project, we used Alveo U200 boards – these offer a PCIe Gen3 interface – but we procured a few PC servers sporting PCIe Gen4 slots to validate the architecture on devices equipped with a x16 Gen4 PCIe module like the U280 and therefore exploit an enhanced interface towards the host system.

As shown in Figure 26, APEnetX architecture leverages on the combination of (i) the network IPs (APErouter and APElink) developed during the ExaNest and EuroEXA projects, (ii) the embedded DMA engine and transceivers of the Xilinx FPGA board, (iii) the APEni providing Host-to-Card (H2C) and Card-to-Host modules to manage the data transmission towards the PCIe interface, and (iv) the register interface to manage configuration and status registers.

In RED-SEA we focused on the development and deployment of a proprietary Network Interface (APEni) compatible with the embedded DMA engine of the Xilinx. The APEni interfaces with the DMA engines available in the Xilinx Device to encapsulate data in the INFN-proprietary format feeding the network. APEni supports RDMA semantics to manage user-level, zero-copy RDMA data transmission to/from the user memory offloading the kernel system. In the current implementation,the virtual memory is managed through the IOMMU of the Intel processor for the implementation of the direct I/O. To support the DMA read/write process INFN provided a complete software stack customizing the Xilinx one, adding as small a set of modifications as possible to be compliant with future releases by the vendor. We provided a user-space library (LIBQCM) between user-space application and the Linux device driver; this

component is used to prepare the data structures used in the IOCTL *syscall* and to implement the optimization we made to bypass the user-space to kernel-space context switch.

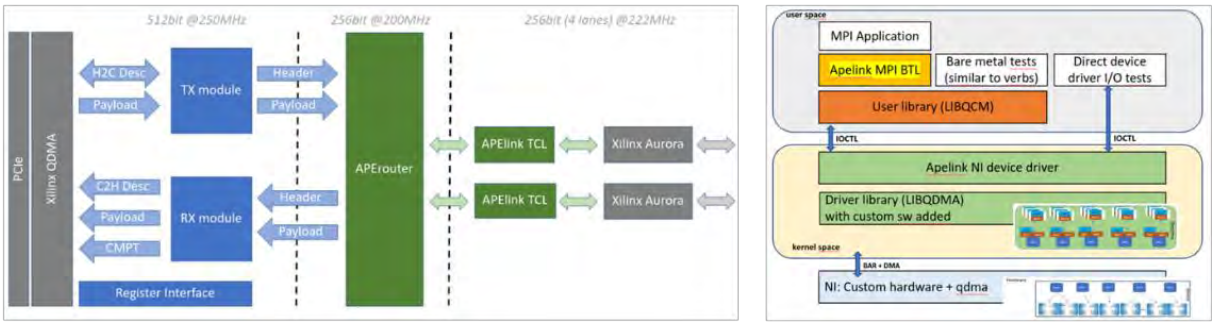


Figure 26: APEnetX architecture and software stack.

Several optimizations and advanced custom mechanisms were implemented to enhance the performance and comply with the requirement of optimizing the transmission of low-size messages:

1. **Direct Completion**: having user-space completion management, the nodes involved in the communication exchange information on the completion virtual addresses during the initial handshake. This allows bypassing the device driver kernel module in the receiving phase, while the receiver user process polls on its completion address.
2. **User-Space Completion queue remap**: the completion queue (integrated in the C2H queue) memory area is remapped into user-space, to make it accessible directly from the user application.
3. **Payload in completion**: we can exploit up to four reserved 64-bit words (32B, small packet optimization) of the “completion” data structure to transmit a payload small enough to fit instead of pointing to data written elsewhere in memory.
4. **Fast Send**: The Linux device driver is completely bypassed by the user-level library, which uses the AVX512 instruction set extension to do an atomic write directly to the hardware shared memory (PCI BAR4).

9.2.2 OpenMPI on APEnetX

Furthermore, INFN developed a Byte Transfer Layer (BTL) component that can be instantiated by the Modular Component Archictecture (MCA) framework of OpenMPI to use the APEnetX NIC for MPI data transmission. For APEnetX we choose OpenMPI, being open source, portable and widely deployed in the HPC world. The support of GPU memory transfers requires the so-called NVIDIA GPUDirect RDMA feature. This is available from Kepler-class GPUs onwards and allows us to enable a direct path for data exchange between the GPU and ApenetX device over PCI Express.
In Figure 27, we report results for (i) the INFN proprietary synthetic latency and bandwidth tests, (ii) the de-facto standard tests of the OSU Micro Benchmarks suite, (iii) two synthetic tests to move data from GPU memory to CPU memory and back from CPU memory to GPU memory on the same host (local-loop). The testbed is composed by two Supermicro X13SEI-F servers equipped with one Tesla P100 GPU and one NVIDIA A100. As can be seen, all the operations involving GPU pinning and locking (when the source of the data is the GPU memory) incur more overhead in terms of latency.

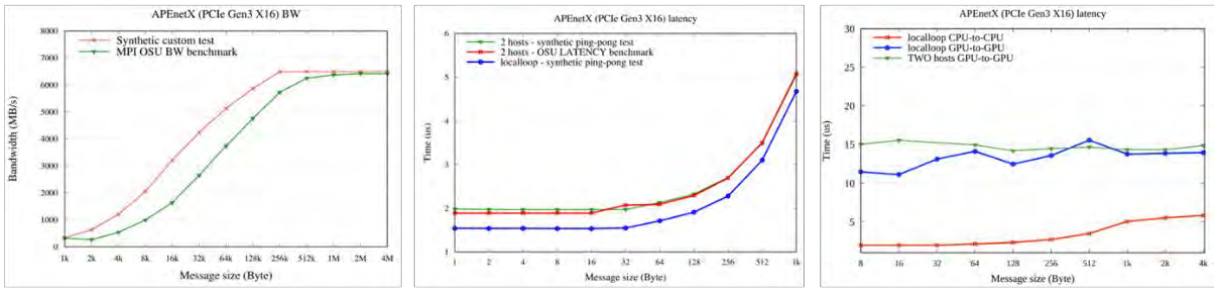


Figure 27: Test were carried out between two Supermicro X13SEI-F servers equipped with Intel Xeon Silver 4410T processors.

9.2.3 APEnetX in BXI environment

The integration of APEnetX in the BXI environment required the integration of BXI link layer. The BXI link layer firmware provided by ATOS as an encrypted module was first ported onto the the APEnetX prototype (Xilinx Alveo U200) and, subsequently, an adapter between the BXI and APEnetX communication protocols was designed and implemented. We measured the latency between two boards by performing a pingpong_test (Figure 28) to assess the performance of the present BXI link integration into the APEnetX design.

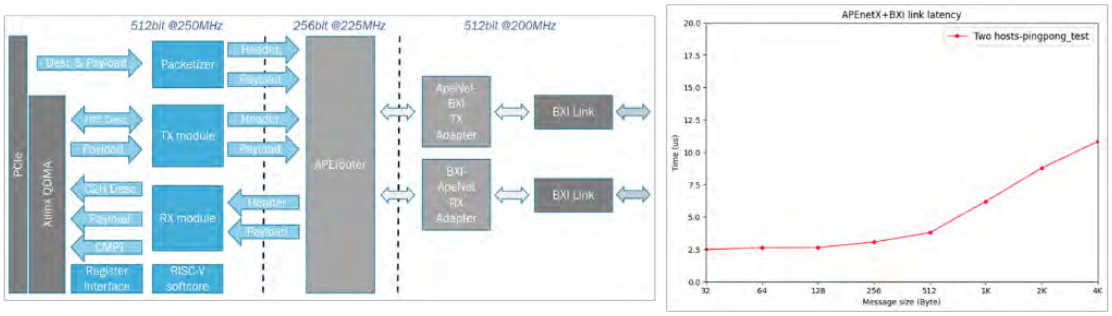


Figure 28: APEnetX+BXI integrated Platform design and its host-to-host link latency.

9.2.4 APEnetX simulator

We implemented a simulator based on OMNet++ to evaluate APEnetX performance at large scale. The INFN simulation library is composed of several modules which can be combined to build a network node:

- The **Buffer** provides the FIFO functionality needed to implement transmission channels.
- The **Channel** logic implements the functionality of the APElink.
- The **Consumer** interface to the receiving (RX) FIFO which pops all messages in the RX queue as soon as they are available.
- The **Producer** interface to inject packets to the transmitting (TX) FIFO.
- The **Router** has all the required gates to connect with an arbitrary number of external ports using Channel logic modules and an arbitrary number of internal ports using Consumer or Producer and Buffer modules.

An example of a network node built using the modules provided by the INFN simulation library is in Figure 29 (left) that has two external channels with two virtual channels each and one internal channel. The APEnet simulator (DQN_SIM) proved to be a useful porting platform for MPI applications. In Figure 29 (right), the throughput of the NEST application within the simulator is plotted. The NEST application is composed of two phases, a building phase, and a simulation phase, during which most of the traffic is produced. Since the INFN interest is focused on the traffic pattern of the application, Figure 29 (right) depicts only the working phase. As expected, the working phase shortens as the number of MPI ranks increases; in the ideal case the simulation halves its runtime when doubling the MPI ranks, provided they are assigned to different cores. Such behavior is confirmed by the APEnetX simulator.

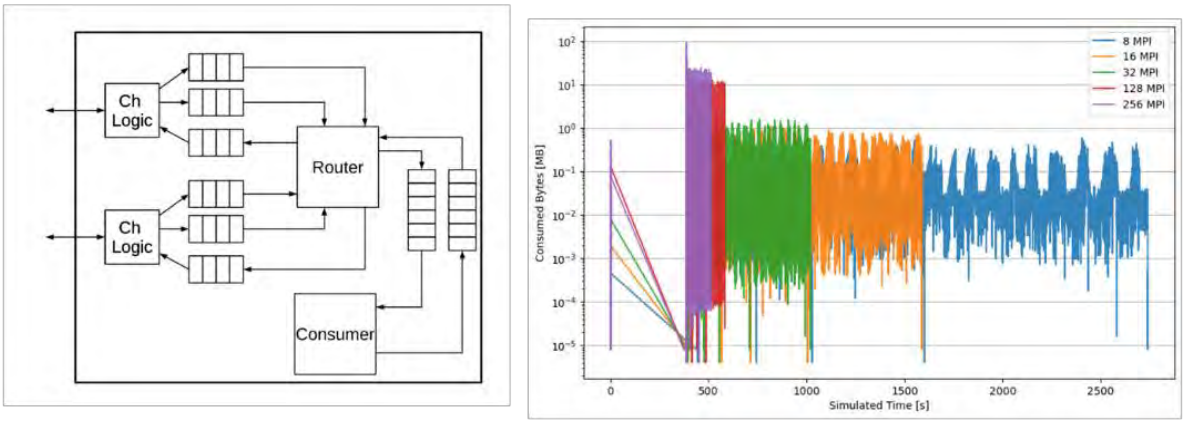


Figure 29: Example network node and simulated time of the NEST traces injected in the APEnetX simulator (DQN_SIM).

10 Parastation MPI and Gateway for BXI

In the RED-SEA project, the main goal in Task 4.5 for ParTec is to extend ParaStation MPI to enable the utilization of BXI networks within computing clusters following the MSA approach and to optimize one-sided communication semantics in ParaStation MPI on top of BXI.
The ParaStation MPI communication stack is a central pillar of ParaStation Modulo, a comprehensive software suite especially designed for MSA systems. ParaStation MPI is an MPICH derivate integrating its low-level communication layer pscom at the ADI3 layer (see Figure 30). The pscom library enables point-to-point communication among the MPI processes and abstracts the hardware with a variety of plugins supporting different interconnects and interfaces relevant to the HPC domain, e. g., InfiniBand, UCX, Extoll, and OmniPath. The natural way for adding support for BXI to the ParaStation MPI communication stack is to add an appropriate plugin to the pscom.

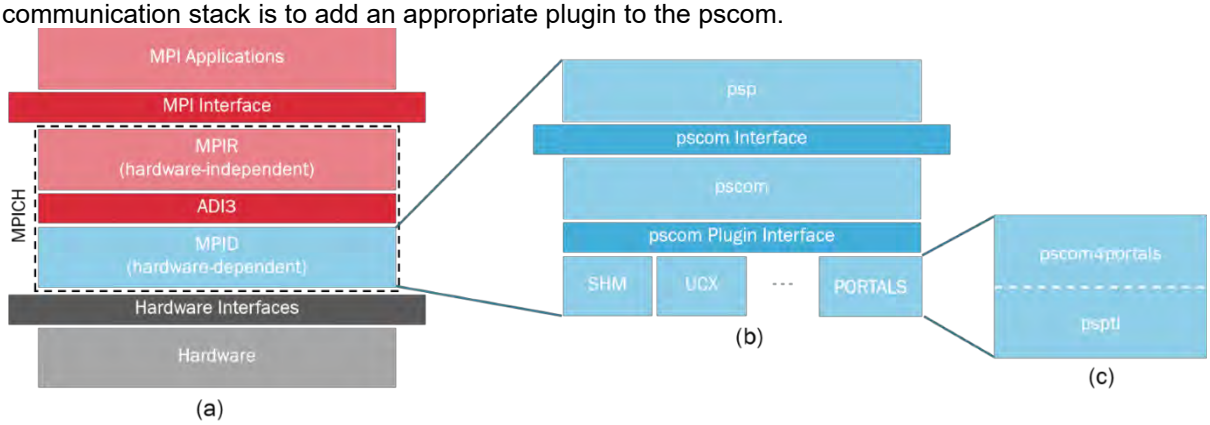


Figure 30: The architecture of ParaStation MPI: The pscom is used to implement the psp layer (b) for an integration into the general MPICH software stack (a) as an ADI3 device. BXI support is implemented by adding a pscom plugin with a layered architecture (c).

Our work mainly focused on the following tasks:

- The low-level communication layer pscom is extended by adding a plugin for BXI support. This enables the utilization of BXI networks on high-performance systems following the MSA approach.
- One-sided RMA communication is extended and optimized in pscom on top of BXI. This enables MPI and PGAS programming models the efficient use of BXI-based RMA operations and one-sided communications through the use of pscom.
- The transparent bridging capabilities offered by pscom are further extended by the support for BXI interconnects. This allows MPI applications running on top of a heterogeneous network landscape using BXI among other high-speed interconnects.

10.1 Design and Implementation of the pscom4portals Plugin

The pscom4portals plugin is implemented with the Portals 4 API to leverage the BXI hardware. Similar to other pscom plugins, this plugin is logically divided into two layers (Figure 30(c)): the upper layer implementing the interface to the hardware-independent part of the pscom and the lower `psptl` layer implementing a point-to-point communication channel by leveraging the Portals 4 API.
The main purpose of the upper layer is to implement the handshake mechanism of the pscom. During the handshake procedure, the pscom4portals negotiates and exchanges between two peer processes the required endpoint information for communication, i.e., the Node ID (NID) and the Process ID (PID) provided by the Portals 4 API. Additionally, this layer provides the necessary callbacks for sending and receiving data over pscom4portals connections.
In the lower `psptl` layer, both the Eager and the Rendezvous communication protocol are implemented separately by allocating dedicated Portals Table Entries (PTEs). The rendezvous communication builds upon an RPUT protocol utilizing RMA write operations. Depending on the message size, the plugin

decides which communication protocol is used. Users can change the threshold of message size to switch the communication protocols by setting the `PSP_PORTALS_RENDEZVOUS` environment variable.
The BXI support in ParaStation MPI has been preliminarily evaluated on two hardware testbeds: (1) the Dibona system and (2) the DEEP system. Figure 31 presents the communication throughput obtained with both ParaStation MPI and Open MPI on the DEEP system. The preliminary evaluation results show that the performance of ParaStation MPI is aligned well with Open MPI for large message sizes. There is still room for optimization and improvement for the latency of small-sized messages.

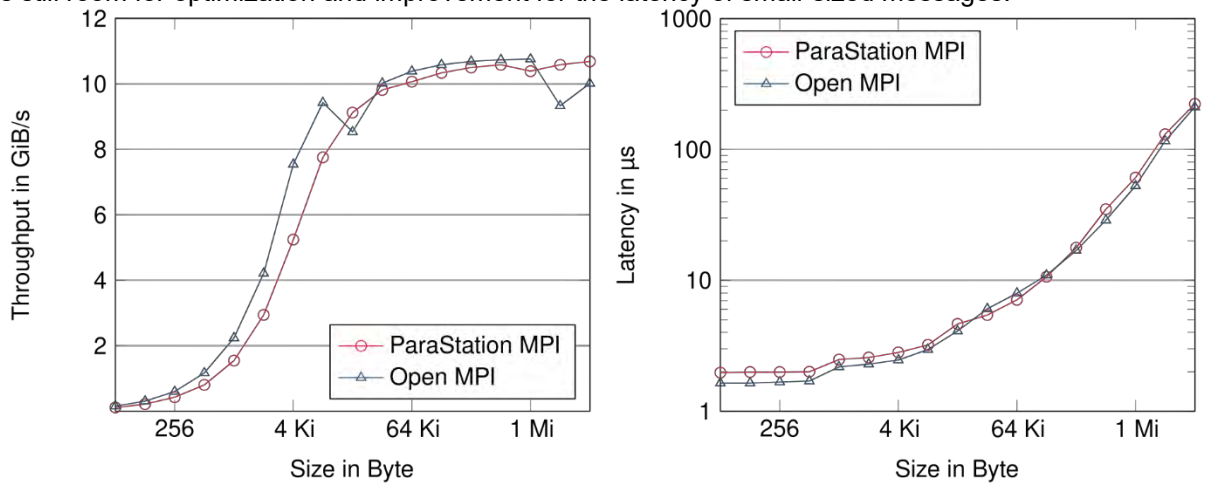


Figure 31: The BXI throughput and latency of ParaStation MPI compared to Open MPI on the DEEP system.

10.2 Extension of RMA capabilities in pscom4portals plugin on top of BXI

The RMA support provided by pscom is implemented based on the same layered architecture in the pscom4portals plugin. Furthermore, the implementation builds upon the re-design of pscom's RMA interface that was developed in the context of the DEEP-SEA project. It enables the efficient execution of both MPI one-sided communication operations and PGAS workloads on top of a single low-level communication layer, thereby improving the composability of different programming models within MSA systems.
The RMA implementation in the pscom4portals plugin layer supports the pscom RMA interfaces, including memory region registration, RMA communication, and synchronization. This is realized by leveraging the one-sided communication operations offered by the Portals 4 API. To register a memory region in the pscom4portals plugin, a Matching List Entry (ME) with specific match bits is created by calling `PtlMEAppend`. The specified match bits, Process Identifier (PID), and Portal Table Index (PTI) are packed into the remote key buffer and used to generate a remote key to access the registered memory region. RMA communication operations (i.e., put, get, and atomic operations) have been mapped onto their Portals 4 counterparts `PtlPut`, `PtlGet`, `PtlAtomic`, `PtlFetchAtomic`, and `PtlSwap` in the pscom4portals plugin. The support for RMA synchronization within pscom4portals is realized by using light-weight Counting Events (CEs) of Portals 4.
The preliminary evaluation of the native RMA support provided by the pscom4portals plugin is conducted on the DEEP system. This analysis compares a new implementation of MPI one-sided communication leveraging the new RMA interface of pscom with an old implementation based on two-sided communication semantics in ParaStation MPI. The newly implemented pscom RMA API using Portals 4 on top of BXI shows important performance improvements compared with the former two-sided-based implementation in ParaStation MPI. Figure 32 and Figure 33 show that a latency reduction of approx. 30% for `MPI_Put` and `MPI_Accumulate` and 60% for `MPI_Get` for small messages (≤ 1024B) with the new implementation is obtained. This is achieved by using hardware-accelerated RMA and a light-weight synchronization mechanism. We also observe a significant latency reduction up to 80% for MPI atomic operations, such as `MPI_Get_Accumulate` as shown in Figure 33. This is due to an internal lock within ParaStation MPI that is required to guarantee the atomicity of the two-sided-based approach.

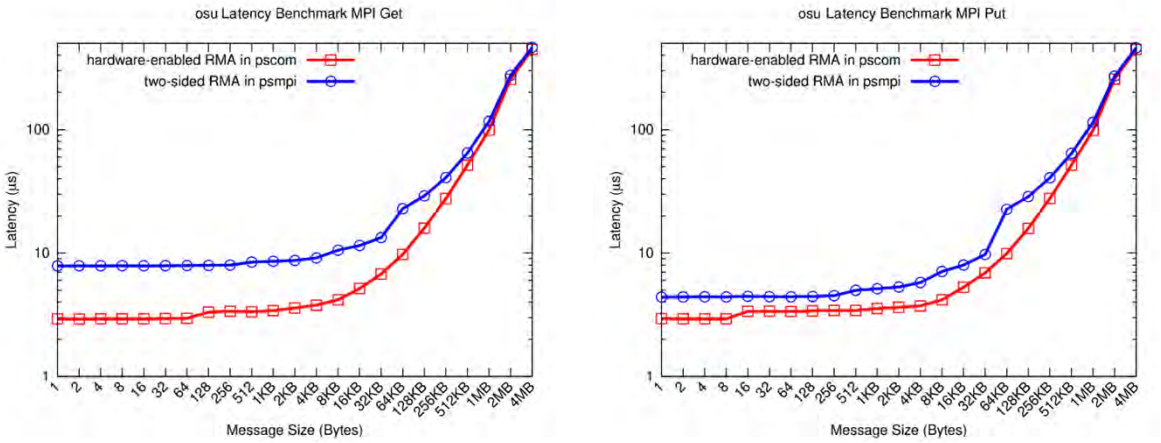


Figure 32: Comparison of Latency of MPI_Put (left) and MPI_Get (right) communication based on hardware acceleration and two-sided communication semantics for OSU one-sided micro benchmarks.

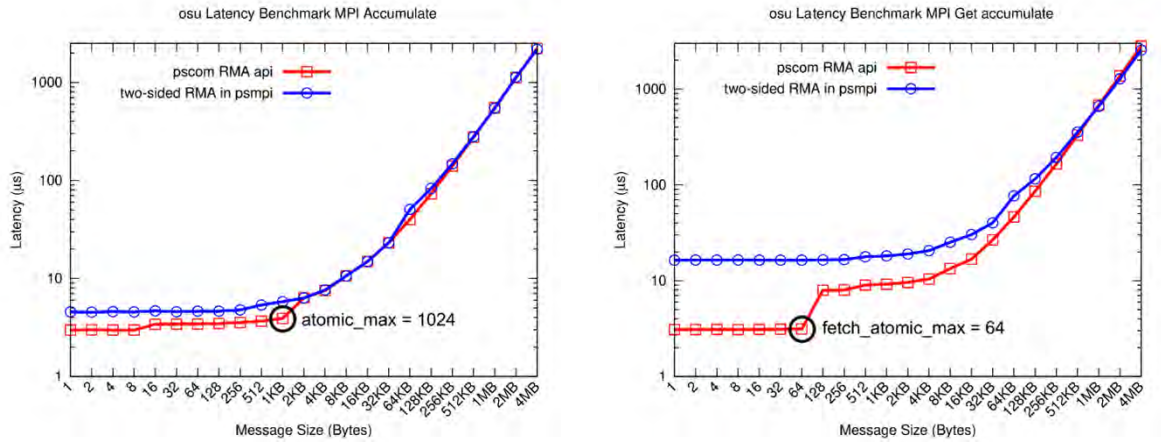


Figure 33: Comparison of Latency of MPI_Accumulate (left) and MPI_Get_accumulate (right) via pscom RMA API and the two-sided RMA communication in ParaStation MPI for OSU one-sided micro benchmarks.

10.3 Transparent Network Bridging

The pscom4portals plugin in conjunction with pscom's gateway capabilities enables the transparent bridging to and from the BXI network. This way, MSA systems with a heterogeneous network landscape can run MPI applications across multiple modules, even if they use different underlying interconnection technologies including BXI.

The DEEP system is used as the testbed for evaluating ParaStation MPI's network bridging capabilities to/from BXI. The DEEP system contains cluster nodes connected via IB and four nodes attached to the BXI high-speed interconnect. A preliminary throughput analysis of the network bridging between a cluster node and a BXI node on the DEEP system has been conducted by running the osu_bw benchmark and shown in Figure 34. This preliminary evaluation of pscom's network bridging capabilities demonstrates its viability for MSA systems using BXI among other high-speed interconnects. Performance improvements can be expected by implementing further optimizations especially w.r.t. to the tuning of plugin-specific parameters.

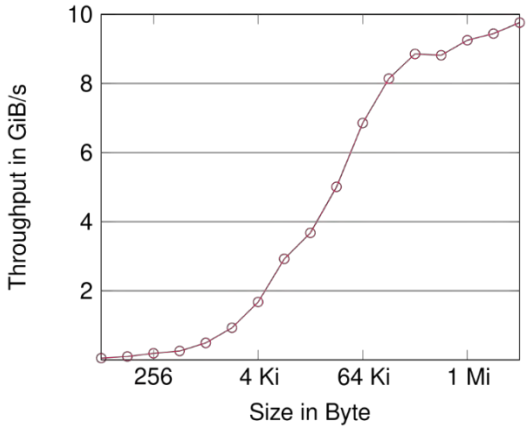


Figure 34: A throughput analysis of the network bridging between InfiniBand and BXI on the DEEP system.

11 HPC Network Simulations and Applications Communication Characterization

This section describes the framework used and extended in the RED-SEA project to characterize, model, and simulate the communication patterns of representative computing- and data-intensive use cases. First, we describe the open-source VEF traces framework that characterizes and models representative communication patterns. Next, we detail how this framework has been used to gather traces in real use cases. We have performed several analyses on these traffic patterns to characterize network congestion by directly analyzing the information recorded in the obtained traces and through simulations using those traces as input.

11.1 Simulation framework description

Simulation is widely used to model the network functionality and evaluate its performance under specific communication patterns generated by the system end nodes when running specific workloads. It is essential to characterize these communication patterns, which will help identify bottlenecks and undesired situations in the interconnection network. Therefore, network simulation tools should be fed using realistic network traffic models, from real benchmarks or applications. This approach has grown in popularity since it permits analyzing the network behavior under realistic traffic situations.

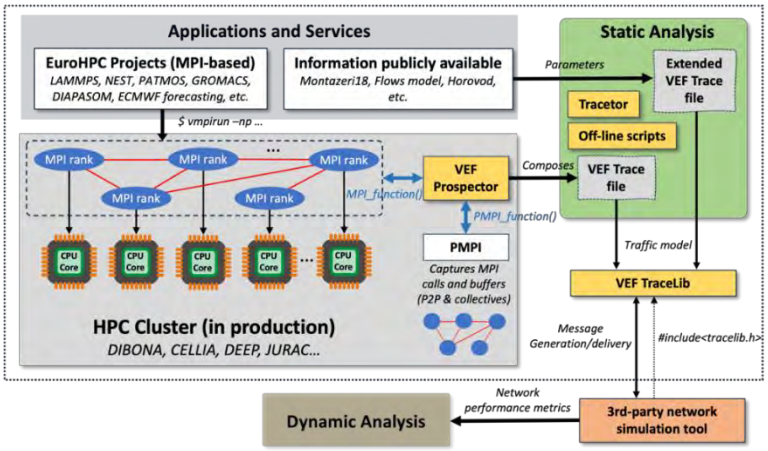


Figure 35. Diagram of the proposed traffic-modeling framework. Text squares in yellow show the applications and tools of the VEF Traces framework.

To characterize specific communication patterns and analyze their impact on the network performance, we have developed an open-source framework, called VEF Traces, which allows for i) instrumenting MPI-based applications communication and recording this communication in traffic traces, ii) modeling realistic network workloads based on the information of their communication patterns made publicly available by HPC systems and Data-center users, iii) reproduce the mentioned communication patterns in any third-party simulation tool, and iv) perform different types of analyses (static and dynamic) using the characterized communication patterns. Figure 35 shows a general overview of the workflow of the VEF Traces framework, including the most important tools (in yellow) available.

Specifically, the VEF traces framework permits recording the communication operations behavior of parallel applications run in HPC clusters into traffic traces. These applications are commonly based on the Message Passing Interface (MPI) programming model, so we would need to run the selected application in a real HPC cluster and use the *VEF Prospector* tool to record its communication pattern into a VEF trace. This tool leverages the *profiling MPI* (PMPI) library to capture the MPI calls (either point-to-point or collective) generated by each MPI rank. The PMPI information per MPI rank is stored locally at every node in the cluster, and later it is combined using the `vef_mixer` tool into a single VEF Trace. Although trace instrumentation is a reliable way of modeling communication operations, the main problem is its lack of scalability. It is not trivial to reproduce the behavior of a VEF trace, which has been

gathered in a HPC cluster using a given number of MPI ranks, significantly higher than that of a given VEF trace.
This section describes how we have used the VEF traces framework combined with the SAURON simulator to characterize the communication patterns and congestion in high-performance interconnection networks. As it has been described in previous deliverables (such as D1.3), the SAURON simulator has been widely used and tested in the last decade to model specific interconnection network architectures (e.g., InfiniBand or BXI). During the RED-SEA project SAURON has been extended (mainly with the work carried out in WP1) to model the BXIv3 architecture, and to model new proposals for network resource management (designed in WP3). This work has contributed to KPI #2 and #8, since we have achieved to simulate interconnection networks communicating up to 115K endpoints, so these KPIs related to scalability and network topology have been partially achieved thanks to the mentioned simulation framework. In the following sections, we describe a real use case in which we have used the VEF traces framework and the SAURON simulator to model and characterize communication and congestion in high-performance interconnection networks.

11.1.1 Static analysis of VEF traces

The VEF traces framework also provides a set of scripts, called *off-line scripts*, that allow to perform a *static analysis* of the communication operations, which involves looking at their type (P2P or collective), calls number, number of generated bytes, source and destination end-nodes involved in the communication, etc. These scripts generate a large set of plots depicting these metrics, which also can be gathered in a single file report.
In RED-SEA we have promoted different cross-collaborations with other partners, both within this project and from other projects (e.g., DEEP-SEA, IO-SEA or MAELSTROM), intended to generate a large set of VEF traces based on real use-cases, which can be used to feed network simulation tools and measure the impact of different communication patterns on the network performance. We have created a public repository[5] to store the obtained traces. We have also performed a static analysis of all the obtained traces and uploaded one report per trace stored in the repository. Although we have gathered traces from multiple applications, in the following we show one representative example of a specific application from the EuroHPC-JU funded MALESTROM project.

11.1.2 Dynamic analysis of VEF traces

To reproduce the applications' recorded behavior into the VEF traces, the proposed framework provides a software library, called *TraceLib*, which can be invoked in any simulation tool to read the trace records and generate the corresponding communication messages into a simulated network environment.
Functions of this library are also invoked when messages are received at end nodes, so the dependencies that could appear between different messages can be resolved.
Finally, the network simulation is fed using VEF traces and should provide a set of performance metrics that allow the simulator users to understand the communication pattern behavior in the network these metrics could, for instance, provide information about network bottlenecks, reliability issues, power consumption, etc. The analysis of these metrics using the results of simulation experiments is referred hereafter to *dynamic analysis*.

11.1.3 BXIv3 modeling in the SAURON simulator

The BXIv3 network technology model in the SAURON simulator has been performed in the network interfaces and switches. The idea of this model is to abstract the most crucial details from the BXIv3 specification and model them into SAURON with a proper granularity level, bearing in mind Scalability, since we need to simulate more than 100K endpoints using a single BXIv3 fabric. Further details of the BXIv3 switch model have been described in deliverable D1.3.
Regarding the BXIv3 NIC modeling in SAURON, we assume the NIC output ports to operate in the same manner as switch output ports. These output ports at NICs are organized in VCs and are flow controlled from switches using the PFC algorithm. We assume that NICs are plugged to PCIe in the endpoints so applications generating communication operations request a user space buffer in the RAM memory of a given host. This buffer requests access to the NIC at the transport level, which allocates a specific generation queue for that user-space buffer. In manner, the user space buffer is divided into smaller

[5] https://gitraap.i3a.info/jesus.escudero/vef-traces-repository

packets, according to the network MTU (Maximum Transmission Unit), which are finally moved to the NIC output buffer and injected into the network. Note that this model permits that each user space buffer is allocated to a single generation flow, so different user space buffers can be associated to different generation queues at the same host. In this manner, we avoid starvation in the traffic generation of different applications in the same host that request the same NIC.

11.2 Evaluation results using the simulation framework

Although the SAURON simulator provides a wide set of metrics, the most useful for this analysis are the execution time, the Flow Completion Time (FCT) and the cumulative distributed function (CDF) for the NEST, GROMACS, LAMMPS and PATMOS VEF traces, when used to feed the SAURON simulator. The configured scenarios are a 288-node Dragonfly+ topology (a.k.a., Megafly) and a 256-node fat-tree topology. Specifically, Figure 36 and Figure 37 show the execution time for these applications when using two different switch architectures: BXIv3 and BXIv2. As we can see there are small differences since the amount of traffic in the network generated by these traces is small and these traces do not generate excessive contention.

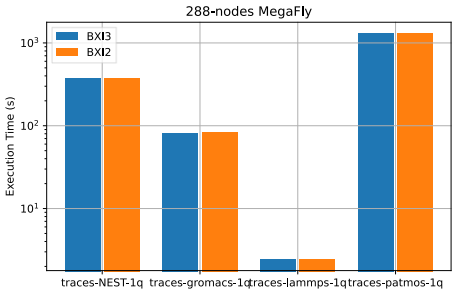


Figure 36: Traces Run Time in a 288-node MegaFly.

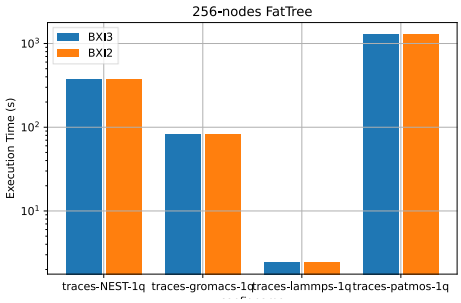


Figure 37: Traces Run Time in a 256-node FatTree.

However, we can observe that BXIv3 always obtains significantly lower FCT values than those obtained by BXIv2, due to the former technology processes the flows faster in average (see Figure 38, Figure 39, Figure 40, and Figure 41). This means that the processing time at the nodes is dominated by the computing times stored in the trace file, which depend on the node architecture used to gather the specific VEF trace.

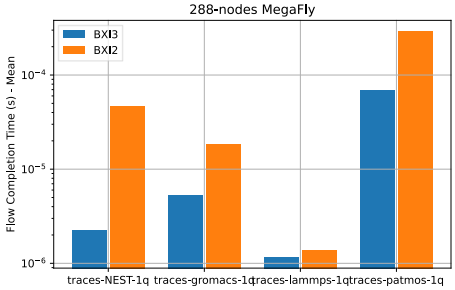


Figure 38: Mean FCT in 288-nodes MegaFly.

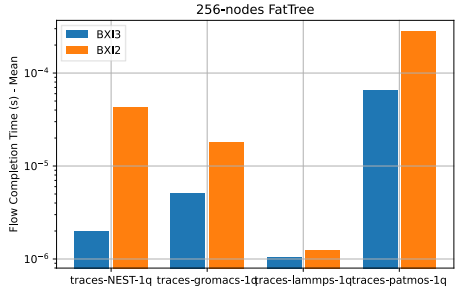


Figure 39: Mean FCT in 256-nodes FatTree.

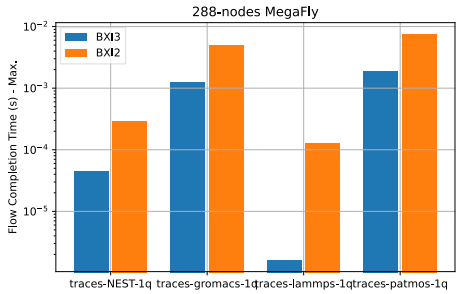


Figure 40: Max. FCT in 288-nodes MegaFly.

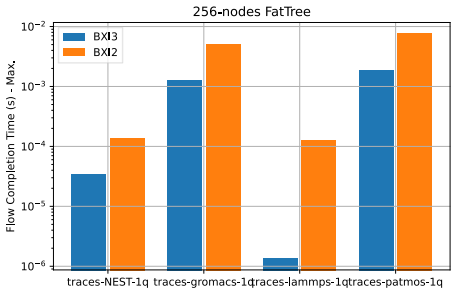


Figure 41: Max. FCT in 256-nodes FatTree.

Finally, Figure 42 shows CDF results using the SAURON simulator when a LAMMPS application has been used to feed a 128-node network configuration using a fat-tree topology. LAMMPS traces have been configured with 128 MPI ranks and run using the COSSIM simulator. In this environment, EXAPSYS has gathered two VEF traces, one configuring COSSIM to simulate ARM processors in the server hosts and the other using RISC-V processor. We have used these traces to feed SAURON, which has been configured using the BXIv2 and BXIv3 network models. As we can see the BXIv3 network model obtains significantly better results compared to those of the BXIv2 one.

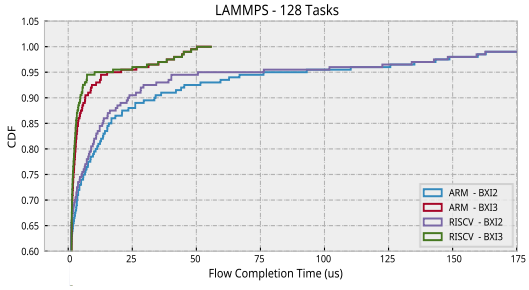


Figure 42: Cumulative Distributed Function (CDF) of a different LAMMPS Traces (128 MPI Tasks) collected by Exapsys in a simulated environment using COSSIM simulator and using two different CPU processors (ARM and RISC-V).

11.3 Scalability study

Finally, we have performed scalability experiments of the SAURON simulator model to test that the BXIv3 model operates under different amounts of traffic workloads when using thousands and hundreds of thousands of nodes. Figure 43 shows the experiments results of a 16K-node fat-tree network when a synthetic traffic pattern is injected in the network and different traffic loads are used. Note that the synthetic traffic pattern is based on network end-nodes generating an all-to-all communication pattern.

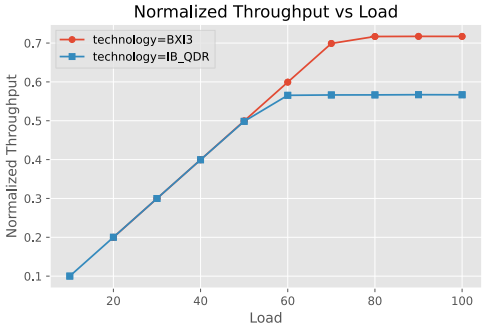


Figure 43: 16K-nodes – Normalized Throughput – BXIv3 vs IB QDR.

As we can see in this figure, the BXIv3 network model operates the InfiniBand network model when traffic loads exceed the 60% of the network capacity.

Figure 44 and Figure 45 show the experiment results that demonstrate one of the most relevant milestones achieved in the SAURON simulator development: the possibility of simulating interconnection networks of more than 100K-node, as defined by KPIs #2 and #8 of the project. Specifically, these figures show the experiment results when generating a synthetic traffic pattern in a network configuration based on a 114K-node Dragonfly topology when used in BXIv2 and BXIv3 networks. Note that this size is not possible for BXIv2 networks, which can only interconnect up to 64K end nodes. Moreover, this size cannot be neither achieved by regular BXIv3 networks, unless level 3 routers are used to interconnect two 64K fabrics, which is expected to be done with BXIv3 networks thanks to the Modular Supercomputing Architecture (MSA) approach.

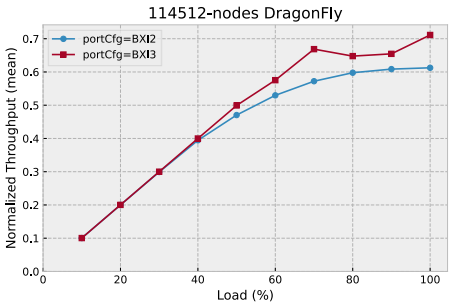


Figure 44: 114K-nodes - Throughput vs Load.

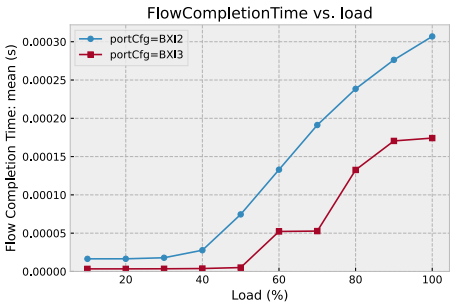


Figure 45: 114K-nodes - FCT vs Load.

12 Congestion Characterization and Control

UPV has enhanced Sauron simulator in several ways to undertake the tasks of work package 3, therefore the improvements are closely linked to tasks 3.1, 3.2 (optimization of collectives communication primitives) and 3.4 (inter-application interference).

12.1 Dynamic analysis using VEF Traces

UPV has designed plots to show the occupancy of the queues in the network to characterize congestion in the network as a previous step to propose congestion reduction techniques in WP3. We use two types of these plots in D3.1.
Figure 46 shows the queue occupancy evolution per time interval. Each colour represents a different network switch. For each switch, we show the highest queue occupancy of the queues (i.e., the highest number of busy queue entries) of that switch. The queue occupancies of the different switches are shown stacked. The maximum queue occupancy for a port is 32. As can be seen, there are time intervals with a high occupancy of the switch queues, especially at the beginning of the trace. This congestion corresponds to the initial part of the trace where we observed a high network latency and low transferred traffic. This means that there is congestion in the network and that the packets are stopped without being able to advance through the network.

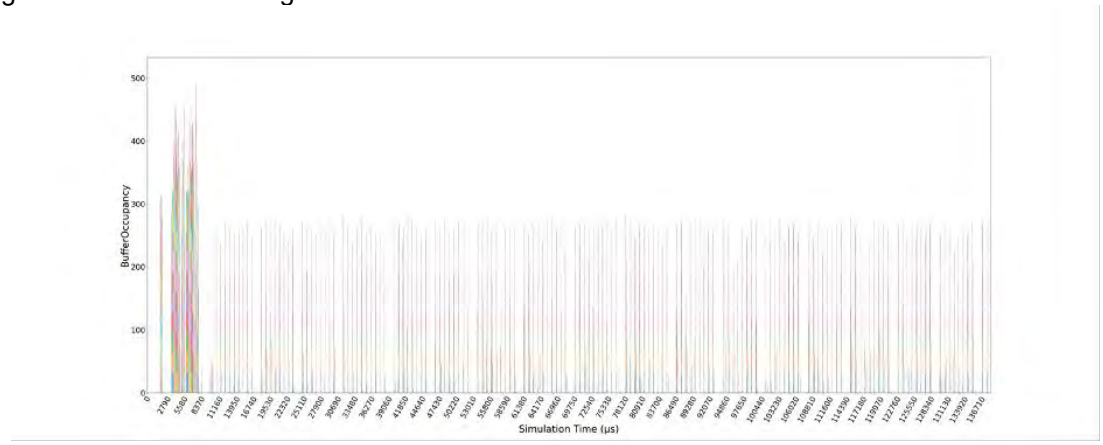


Figure 46: Queue occupancy (number of packets in the queue) evolution along time in an 8-ary 2-tree for the whole LAMMPS trace with 64 tasks. Each colour represents a different network switch. Details are zoomed in the next figure.

Figure 47 is a zoom of the previous where more details can be observed.

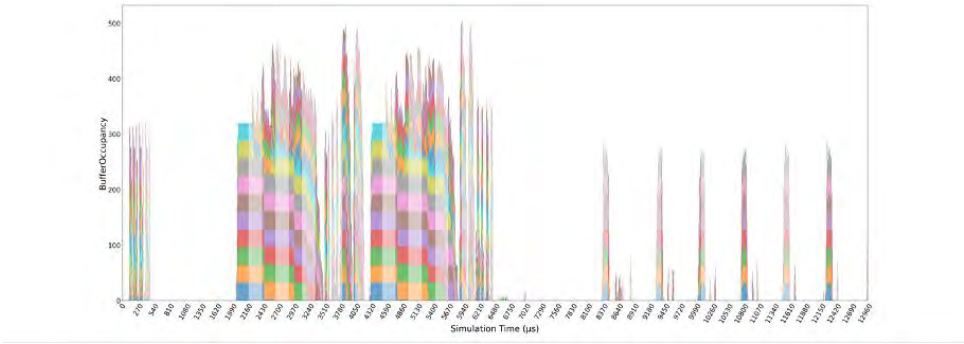


Figure 47: Zoom of Figure 46. Queue occupancy evolution along time in an 8-ary 2-tree for LAMMPS with 64 tasks. Each colour represents a different network switch (there are 16 switches).

The same information is also represented in Figure 48 in a different way. In this plot, the occupancy of each switch (the most occupied queue of the switch is shown) of the network is represented vertically, according to a color scale, while the evolution of the occupancy over time is represented horizontally. In the fat-tree, the second stage switch queues are much less used than the ones from the first stage. In these plots we can see that the switches of the first stage are more used than the ones of the second stage.

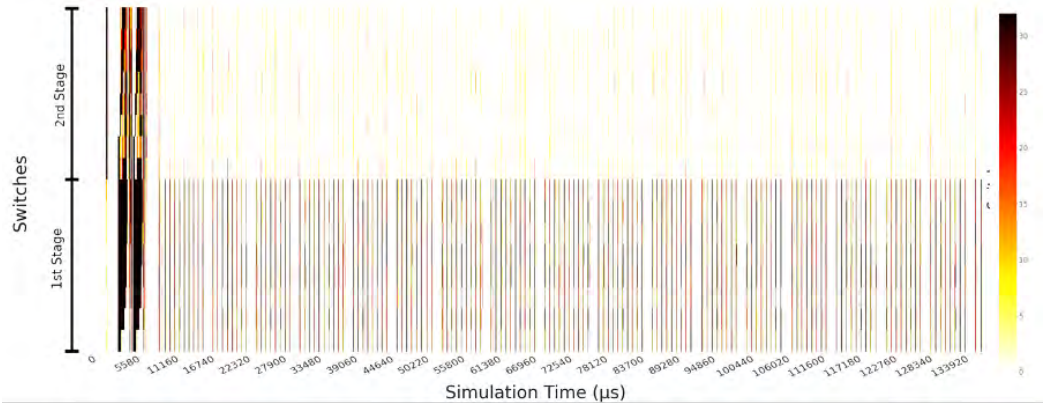


Figure 48: Queue occupancy evolution along time in an 8-ary 2-tree for LAMMPS with 64 tasks. Vertically all the switches of the network are represented and horizontally how the occupancy evolves over time.

12.2 Topology-aware CCP optimizations in TraceLib and in Sauron

In T3.2 we address the optimization of the algorithms that implement the collective communication primitives and to model them we use TraceLib jointly with Sauron.
TraceLib is an open-source library designed to replicate VEF traces behaviour within interconnection network simulators. In particular TraceLib model the behaviour of the algorithms present in the MPI libraries to implement the collective communication primitives. In RED-SEA we have enhanced TraceLib to model several topology-aware algorithms defined in T3.2 that implement the broadcast primitive. They are the LLF (Local Link First) and GLF (Global Link First). These algorithms are two software optimizations based on the knowledge of the topology and are explained in detail in D3.3.

Additionally, in the switches of Sauron we model a hardware assisted mechanism to optimize broadcast primitives. They are based on multicast routing. That is the Sauron simulator has been enriched to model the assistance from hardware to the implementation of CCP.

In Figure 49, we can see the execution time required to process 100 broadcasts (the complete evaluation of these implementation is done in D3.3) of 1 MB each. It is shown for a traditional implementation (binary-tree) and the three topology-aware implementations (HW, LLF and GLF) proposed in WP3. As can be seen, the three topology-aware algorithms are able to reduce the execution time, that is, they are able to process the 100 broadcasts faster than the binary-tree algorithm. As can be observed, the hardware mechanism is the implementation with the shortest execution time (it is able to reduce to a fifth the time obtained by the traditional algorithm, binary tree; it provides a speedup of more than 400%), followed by LLF (it is able to reduce less than half the execution time of the traditional algorithm, the speedup is bigger than 130%), GLF and being the traditional implementation the algorithms that gets the worst result.

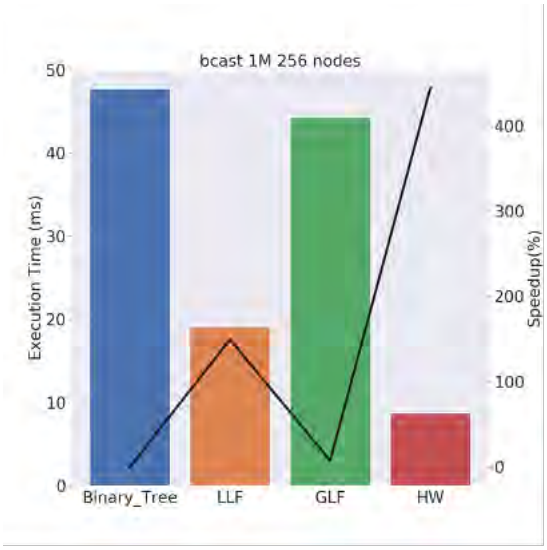


Figure 49: Execution time for 100 1-M broadcast in a 342-node Dragonfly topology for the four broadcast implementations.

Additionally, we have also explained these improvements with the dynamic network throughput (evolution along simulation time of the throughput) shown in Figure 50.

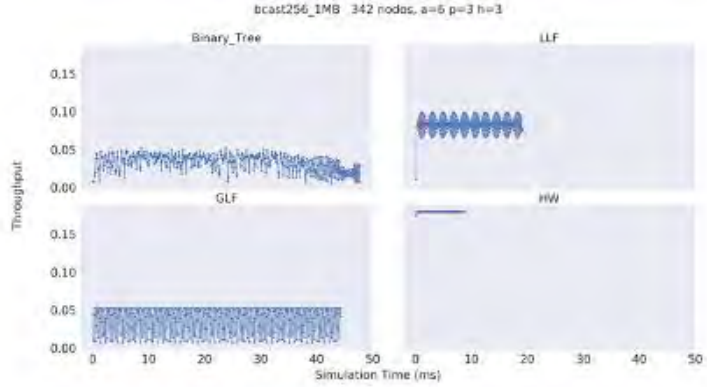
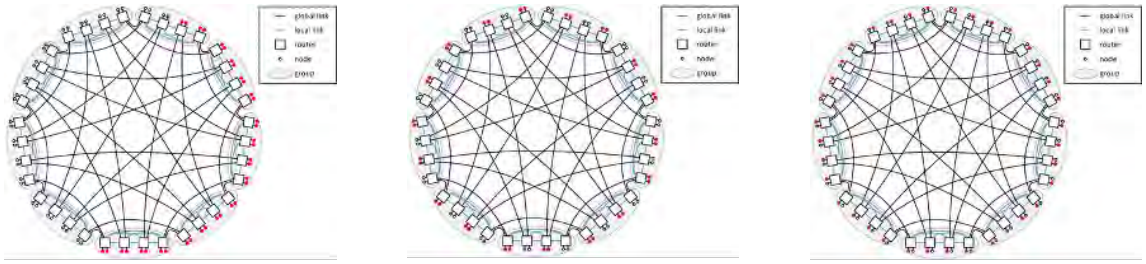


Figure 50: Dynamic throughput for 100 1-M broadcast in a 342-node Dragonfly topology for the four broadcast implementations.

12.3 Support for several VEF-traces in Sauron

To undertake T3.4 several enhancements have been included in Sauron. UPV has upgraded the Sauron simulator to allow the execution of several VEF traces concurrently in Sauron with several mapping policies. The mapping policy defines the collocation of the tasks of the applications to computing nodes in the network. Three different policies of mapping application tasks to network processing nodes have been implemented in Sauron. These mapping policies, shown in Figure 51, have been used in T3.4 when analyzing the inter-application interference.

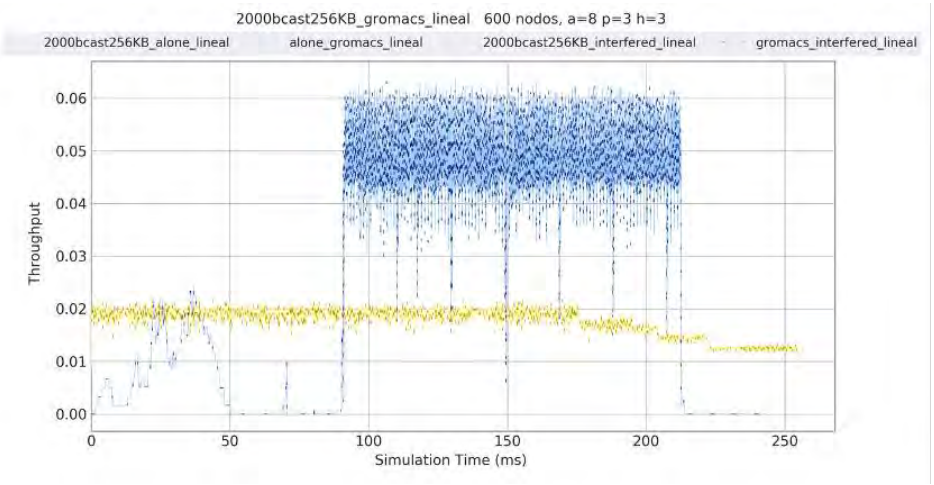
The first one consists of assigning successive compute nodes in the network to the application (Linear mapping) until the application tasks are completed. Switch mapping is the second policy where complete switches are alternated, so that all processing nodes of alternative switches are assigned. Finally, the last policy assigns alternate nodes (Node mapping). These policies are intended to evaluate disparate application mapping situations to analyze the interference produced in each of these situations.



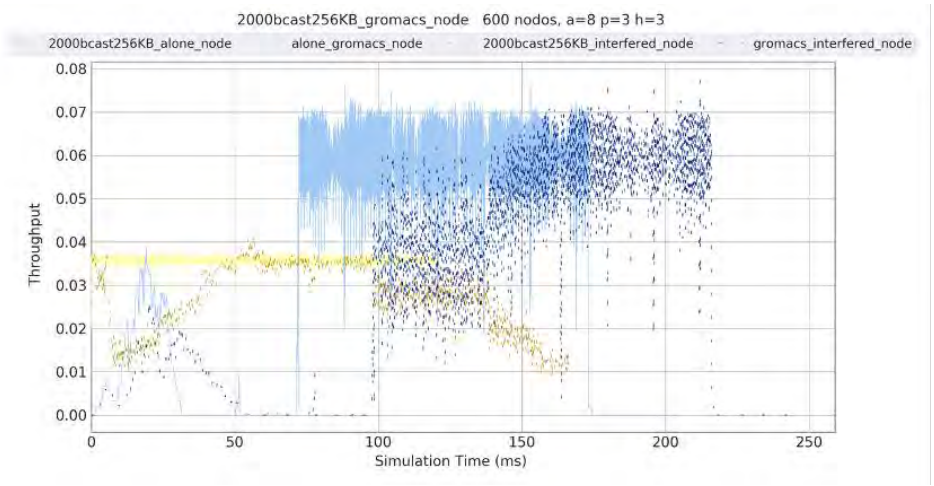
a) Linear mapping b) Switch mapping c) Node mapping

Figure 51: Mapping policies. Application tasks allocation to computing nodes.

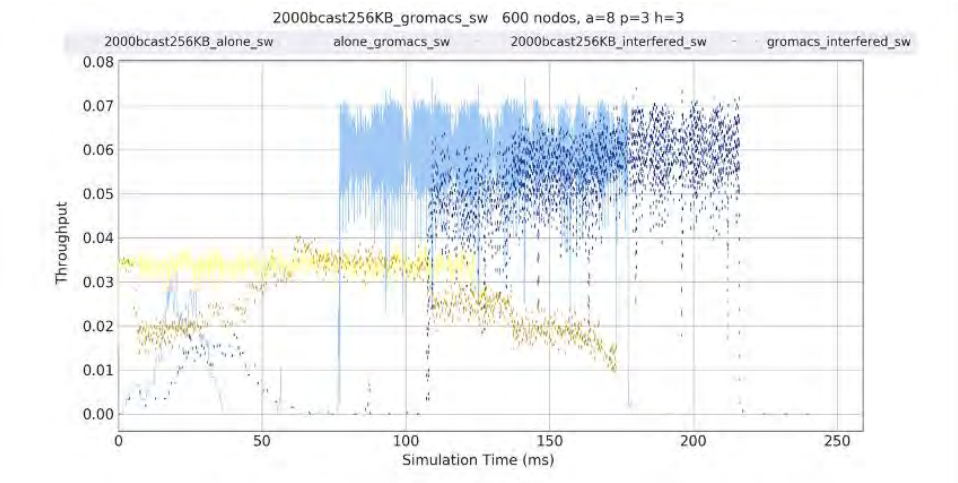
Additionally, when two applications are running concurrently, the execution times are different, and statistic for them must stop being collected in different times. The simulator must stop collecting statistics for the first one when it finishes while for the other it must continue collecting statistics until its end. We have adapted the simulator for doing this. This can be seen in Figure 52, where we present the Dynamic network throughput for the benchmark of 500 broadcasts and Gromacs in concurrent execution and in isolated execution for the three mappings considered.



a) Linear mapping



b) Node mapping



c) Switch mapping

Figure 52: Dynamic network throughput for the benchmark of 500 broadcasts and Gromacs in concurrent execution and in isolated execution for the three mapping policies.

In these figures we can see the dynamic network throughput for both applications running concurrently and also the dynamic network throughput when they are run in isolation. Additionally, the figures also give information about when each of the applications finishes its execution.

13 Conclusion

The RED-SEA developed a plethora of technologies on interconnects. It also formed a great environment for the European BXI interconnect to evolve. Overall, our key objectives and targets on scalability, bandwidth, open standards, etc., have been met while the BXIv3 architecture and first hardware implementations have been produced and are tested in first hardware platforms.

14 Acronyms and Abbreviations

Term	Definition
ACC	Accurate Congestion Management
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BXI	Bull eXascale Interconnect
CCP	Collective Communication Primitive
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
FCS	Frame Check Sequence
FCT	Flow Completion Time
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
FPGA	Field-Programmable Gate Array
GLF	Global Link First
HAS	High-level Architecture Specification
HLA	High Level Architecture
HPC	High Performance Computing
IO	Input/Output
IP	Intellectual Property / Internet Protocol (depending on the context)
ISA	Instruction Set Architecture
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LLF	Local Link First
MAC	Medium Access Control
MPC	Multi-Processor Computing
MPI	Message Passing Interface
MSA	Modular Supercomputing Architecture
MTU	Maximum Transmission Unit
NIC	Network Interface Controller
OSI	Open Systems Interconnection
OSU	Ohio State University
QoS	Quality of Service
P2P	Point to Point
PCI	Peripheral Component Interconnect
PCS	Physical Coding Sublayer
PFC	Priority Flow Control
PGAS	Partitioned Global Address Space
PMA	Physical Media Attachment
PMD	Physical Media Dependent
QFDB	Quad-FPGA DaughterBoard
RDMA	Remote Direct Memory Access
RISC	Reduced Instruction Set Computer
RMA	Remote Memory Access

Term	Definition
RRP	Rate Re-evaluation Period
RS	Reconciliation Sublayer
RTL	Register Transfer Level
SN	Sequence Number
TCP	Transmission Control Protocol
WRR	Weighted Round Robin

Table 2: Acronyms and Abbreviations