APEIRON: a Framework for High Level Programming of Dataflow Applications on Multi-FPGA Systems

Roberto Ammendola[2], *Andrea* Biagioni[1], *Carlotta* Chiarini[13], *Andrea* Ciardiello[3], *Paolo* Cretaro[1], *Ottorino* Frezza[1], *Francesca* Lo Cicero[1], *Alessandro* Lonardo[1], *Michele* Martinelli[1], *Pier Stanislao* Paolucci[1], *Luca* Pontisso[1], *Francesco* Simula[1], *Cristian* Rossi[1,*], *Matteo* Turisini[14], and *Piero* Vicini[1]

[1]Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Roma, Rome, Italy

[2]Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Roma Tor Vergata, Rome, Italy

[3]Università di Roma "La Sapienza", Rome, Italy

[4]CINECA, Bologna, Italy

Abstract. High Energy Physics (HEP) Trigger and Data Acquisition systems (TDAQs) need ever increasing throughput and real-time data analytics capabilities either to improve particle identification accuracy and further suppress background events in trigger systems or to perform an efficient online data reduction for trigger-less ones.

As for the requirements imposed by HEP TDAQs applications in the class of real-time dataflow processing, FPGA devices are a good fit inasmuch they can not only provide adequate compute, memory and I/O resources but also a smooth programming experience thanks to the availability of High-Level Synthesis (HLS) tools.

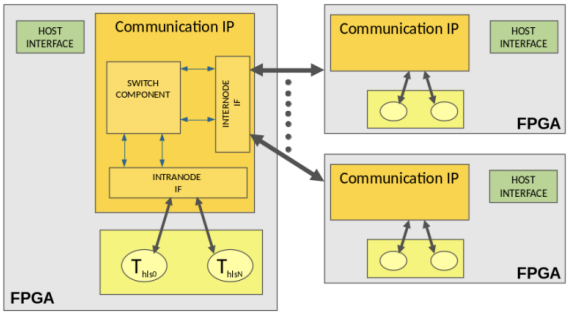
The main motivation for the design and development of the APEIRON framework is that the currently available HLS tools do not natively support the deployment of applications over multiple FPGA devices, which severely chokes the scalability of problems that this approach could tackle. To overcome this limitation, we envisioned APEIRON as an extension of the Xilinx Vitis framework able to support a network of FPGA devices interconnected by a low-latency direct network as the reference execution platform.

Developers can define scalable applications, using a dataflow programming model inspired by Kahn Process Networks, that can be efficiently deployed on a multi-FPGAs system: the APEIRON communication IPs allow low-latency communication between processing tasks deployed on FPGAs, even if they are hosted on different computing nodes. Thanks to the use of HLS tools in the workflow, processing tasks are described in C++ as HLS kernels, while communication between tasks is expressed through a lightweight C++ API based on non-blocking *send()* and blocking *receive()* operations.

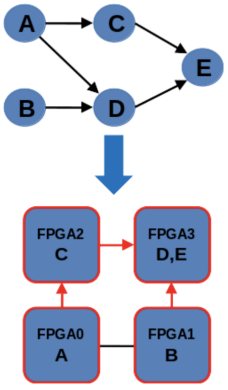
*e-mail: cristian.rossi@roma1.infn.it

1 Introduction

The APEIRON framework [1] aims at offering hardware and software support for running real-time dataflow applications on a network of interconnected FPGAs.
The general APEIRON architecture has been developed on a custom distributed processing platform composed by *m* input data streams recombined through *n* processing layers using a low-latency, modular and scalable network infrastructure. This mimics the workflow of a typical TDAQ application: data keeps flowing from the readout to a trigger processor or a storage system through layers of processing stages.
Developers have the capability of deploying scalable application on a multi-FPGAs system via a dataflow programming model inspired by Kahn processing networks. This will allow to map computational dataflow graph (Fig. 1b) onto the underlying network of FPGAs via a simple configuration tool that instructs the framework to create all the files required for the FPGA bitstream generation and to build the application interconnection logic (Fig. 1a).



(a) APEIRON interconnection logic: Communication IPs managing data streams I/O and communication between HLS computing tasks. (represented as yellow ovals).



(b) Dataflow graph mapped on 4 interconneted FPGAs system

Processing tasks can be implemented in C/C++ and deployed on the different FPGA nodes thanks to the use of High Level Synthesis tools (e.g. Xilinx Vitis). Communication between tasks is expressed through a lightweight API (called HAPECOM) based on non-blocking *send()* and blocking *receive()* operations and is implemented by the APEIRON communication IP (Fig. 1a).

2 Motivation

The main motivation for the design and development of the APEIRON framework is that the currently available HLS tools do not natively support the deployment of applications over multiple FPGA devices, which severely chokes the scalability of problems that this approach could tackle. To overcome this limitation, we envisioned APEIRON as an extension of the Xilinx Vitis HLS framework able to support a network of FPGA devices interconnected with a low-latency direct network as the reference execution platform. For what concerns its usage in HEP experiments, APEIRON project aims at developing a flexible framework enabling the rapid development of both traditional "low level" trigger systems or data reduction stages in trigger-less ones or streaming readout experimental setups characterized by high event rates.

3 APEIRON Building blocks

3.1 Communication IP

The Communication IP represents the main enabling component of the APEIRON framework and is based on the HPC direct network designs previously developed by our group, like APEnet [2] and ExaNet [3].
The Communication IP implements within the framework a direct network which allows low-latency data transfer between processing tasks deployed on the same FPGA (intra-node communication) and on different FPGAs (inter-node communication). These processing tasks are implemented as HLS kernels; the details of their definition in the framework and of their interface with the IP are presented in Sec. 3.2.1.

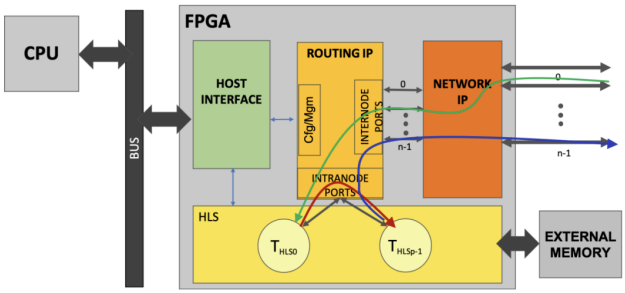


Figure 2: Communication IP hardware block structure with HLS kernels performing intra-node (red line) and inter-node (green line – receive, blue line– send) communications.

Figure 2 shows the Communication IP hardware block structure, which contains a Network IP and a Routing IP, both developed in VHDL for Xilinx Alveo U200 and U280 cards.
The Routing IP defines the switching technique and routing algorithm and consists in the Switch component, the Configuration/Status Registers and the InterNode and IntraNode interfaces. The Switch component dynamically interconnects all ports of the IP, routing between source and destination ports. Dynamic links are managed by routing logic together with arbitration logic: the Router configures the proper path across the switch while the Arbiter solves contentions between packets requiring the same port.
For inter-node communications, the routing policy applied is the dimension-order (DOR) one: it consists in reducing the offset along one dimension to zero before considering the offset in the next dimension in antilexicographic order.
The employed switching technique (i.e., when and how messages are transferred) is Virtual Cut-Through (VCT) [4]: the router starts forwarding the packet as soon as the algorithm has picked a direction and the buffer used to store the packet has enough space. The deadlock-avoidance of DOR routing is guaranteed by the implementation of two virtual channels for each physical channel (with no fault-tolerance guaranteed) [5].
The transmission is packet-based: the Communication IP sends, receives and routes packets with a header, a variable size payload and a footer.

3.2 Runtime Software Stack

The APEIRON framework currently supports Xilinx Ultrascale PCIe-based accelerator cards. We designed a runtime software stack based on the Xilinx Runtime (XRT) architecture, which

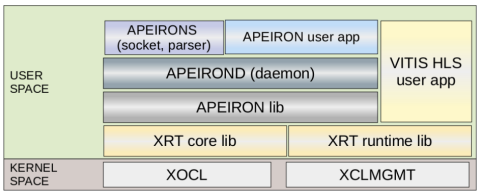


Figure 3: APEIRON Software Stack scheme

is implemented as a combination of user-space and kernel driver components [6]. The APEIRON runtime software stack is built on top of the XRT one, adding three layers as shown in Fig. 3, to:

- add the functionalities required to manage multiple FPGA execution platforms (e.g., program the devices, configure the IPs, start/stop execution, monitor the status of IPs, ...);
- eliminate, or at least reduce, the impact of changes in XRT API introduced with any new version of Vitis on the APEIRON host-side applications;
- decouple the APEIRON software stack from the specific platform, easing the future porting of the framework to different platforms/vendors, ideally by extending the APEIRON library layer only.

Apeirond is a persistent daemon used to manage multiple access requests from user apps to the board. It uses functions exposed by the APEIRON library to operate on the devices. *Apeirond* module accepts client connections over a network socket (using the module called *apeirons*) and oversees creating the socket with the client and handling the incoming command (e.g., reading a register or flashing the board). The protocol currently used is based on a TCP/IP socket while messages are serialized and deserialized in JSON format to simplify the parsing phase.

3.2.1 Workflow for FPGA bitstream generation

Users should prepare a YAML configuration file describing the attributes of each HLS kernel (number of input and output channels, IntraNode port of the Communication IP to which the kernel is connected). Starting from this, the APEIRON framework links the Communication IP and the HLS kernels that are connected to it and generates the bitstream for the overall design. The only requisite that HLS kernels must satisfy in order to be linked to the framework is in the format of their prototype that must adhere to this form:

```
void example_apeiron_task(
    [optional kernel-specific list of parameters]
    message_stream_t message_data_in[N_INPUT_CHANNELS],
    message_stream_t message_data_out[N_OUTPUT_CHANNELS])
```

In this way, the HLS kernel implements a generic stream interface for each communication channel based on the AXI4-Stream protocol.

3.2.2 HAPECOM Communication API

The communication between kernels is expressed through HAPECOM: a lightweight C++ API based on non-blocking *send()* and blocking *receive()* operations. This simple API allows

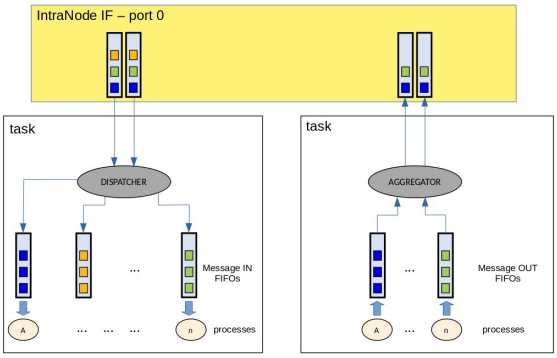


Figure 4: Interface between Intranode Port 0 and the corresponding HLS Task mediated by *Aggregator* and *Dispatcher*.

the HLS developer to perform communications between kernels, either deployed on the same FPGA (intra-node communication) or on different FPGAs (inter-node communication) without knowing the details of the underlying packet communication protocol. The HAPECOM Communication API can be represented with the following pseudo-code:

```
size_t send (msg, size, dest_node, task_id, ch_id);
size_t receive (ch_id);
```

where:

- `dest_node` is the n-Dim coordinate of the destination node (FPGA) in an n-Dim torus network;
- `task_id` is the local-to-node receiving task (kernel) identifier (0-3);
- `ch_id` is the local-to-task receiving FIFO (channel) identifier (0-127).

The Communication Library leverages AXI4-Stream Side-Channels to encode all the information needed to forge the packet header. Two APEIRON IPs manage the adaptation toward/from IntraNode ports of the Routing IP: they are *Aggregator* and *Dispatcher*, shown in Fig. 5. The *Dispatcher* receives incoming packets from the Routing IP and forwards them to the right input channel, according to the relevant fields of the header. The *Aggregator* receives outgoing packets from the task and forges the packet header, then filling the header/data FIFOs of the Routing IP IntraNode port.

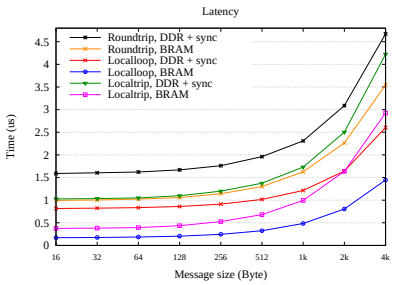
3.3 Validation tests: latency and bandwidth tests

In order to assess the performance of the Communication IP, we measured latency and bandwidth values for intra-node and inter-node communication by implementing in APEIRON a multi-task HLS kernel (*krnl_sr*), connected to every intra-node port of each node and configurable by the host in different modes.
In latency test, the kernel in "send_receive" mode reads a payload data item from the FPGA memory (either BRAM or DDR) and sends and receives it through/from the Communication IP to/from a second interconnected FPGA, where a kernel in "pipe" mode has the task of receiving a single packet and bouncing it back to the initiator FPGA, allowing the measurement of inter- and intra-node latency dividing the roundtrip latency by two (Roundtrip and

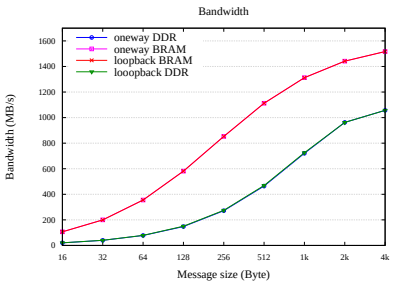
Localtrip modes respectively). In Localloop mode, communication involves the same task as either sender and receiver through the intra-node port it is connected to.
The bandwidth test is instead carried out by transferring multiple data packets with fixed payload size from a “sender” HLS kernel which reads data from the source buffer in FPGA memory (either DDR or BRAM) and pushes them through the Communication IP to another FPGA where a “receiver” HLS kernel writes data into the destination buffer in memory. After receiving the number of data packets whose integrated payload adds up to the size of the receive buffer, the second FPGA pings back a single “ACK” packet with minimal payload to confirm the reception (one-way mode). In loopback mode sender and receiver are two tasks on the same FPGA.

3.3.1 Results

The results obtained for latency tests are reported in Fig.5a indicating the type of tests performed and what kind of FPGA memory is used. In detail, the result obtained shows how the latency values get worse when working with DDR memory, due to overhead issues and to the time required to load the sent buffer from CPU on the FPGA and to store the received buffer from the FPGA to CPU (we refer to these as “sync” operations).
Latency reaches a value slightly below 1 μs for 16 B payload packets in the inter-node "roundtrip" BRAM case, and a value of ~250 ns in the intra-node "localloop" BRAM case.
For what concern bandwidth tests, in Fig.5b it is possible to notice that, referring to the BRAM cases, bandwidth tends to saturate while increasing the size of the packets sent. In particular, for packets of size 4 kB the bandwidth reaches a value of ~12.0 Gbps for the completely overlapped intra-node loopback BRAM case and for the and inter-node BRAM case, with a maximum theoretical value of raw bandwidth equal to 12.8 Gbps (given by the BW limit of 128bit 100MHz of the data injection at router port): the difference is mainly due to the packet protocol overhead.



(a) Latency values measured between HLS Kernels for an intra-node (loopback) and inter-node (roundtrip) communication.



(b) Bandwidth values measured between HLS Kernels for an intra-node (loopback) and inter-node (oneway) communication.

4 APEIRON Use Case

We adopt a Convolutional Neural Network (CNN) we developed to perform online particle identification with the RICH detector of the NA62 experiment [7] as the example processing task to demonstrate the scalability features of the framework. This CNN model has been

developed using Tensorflow/Keras and deployed on FPGA with the HLS4ML [8] software package. The model receives as input a 16x16 image of the hit photomultipliers (PMTs) map for each physics event and produces an estimate for the number of charged particles it contains. Considering the high event rate of the experiment, sustaining an adequate processing throughput is the main challenge for such a system.

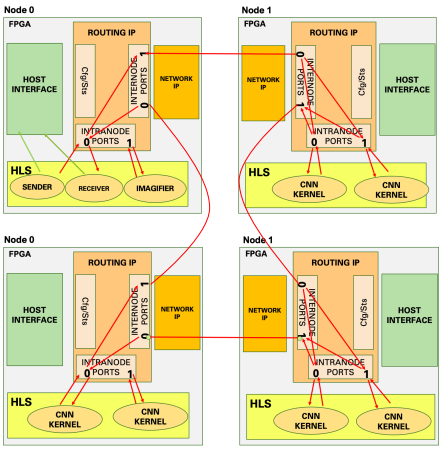


Figure 6: Test setup for the *Imagifier*+CNN kernel system.

The setup used to test the performance of this implementation with APEIRON is depicted in Fig.6. Up to four interconnected FPGAs (Xilinx® Alveo U200) are used as nodes of an APEIRON deployment with distinct roles:

1. **Preprocessing node**: data are loaded from Host memory and sent through the network via an HLS kernel (*krnl_sender*). Data are then processed by an *Imagifier* HLS kernel which turns the PMT hitlist information into a 256bit word (16x16 B&W image) that is sent to the Computing node through the external links. As second task, this node is in charge of receiving the output of the CNN computation and storing it on Host memory via an HLS kernel (*krnl_receiver*). The processing time, from the first packet sent to the last received, is measured on this node.

2. **Computing node**: images coming from external links are taken as input and dispatched to one or both the CNN HLS kernels (depending on the configuration) to compute the predictions. Results are then sent back to the preprocessing node.

We have scaled the system from 2 nodes (one preprocessing and one computing) up to 4 as shown in Fig.6 while the processing time per event and the integrated processing throughput of the system were measured; results are tabulated for the former and plotted for the latter in Fig.7 (throughput is in millions of events per second, MHz in figure). The presented results show the good scaling of system performance with the number of nodes. The flattening slope of the curve when the number of CNNs goes beyond 4 is mainly due the saturation of the data injection rate in the *krnl_sender* on the preprocessing node.

5 Conclusions and Future Work

The APEIRON framework enables the development and deployment of Vitis HLS dataflow applications distributed across multiple-FPGA systems, as shown by the presented use case.

# nodes	# CNNs	Processing time per event (us)
2 nodes	1 CNN	3.440
	2 CNNs	1.720
3 nodes	2CNNs	1.720
	4CNNs	0.860
4 nodes	3CNNs	1.147
	6CNNs	0.731

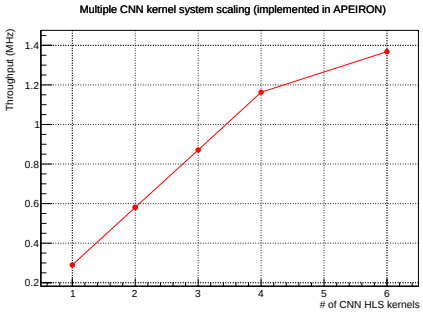


Figure 7: Processing time and throughput scaling with an increasing number of CNN HLS kernels.

The co-design of APEIRON software stack together with its Communication IP allowed reaching very low and deterministic latency and a high fraction of the channel raw bandwidth for communications between FPGAs, addressing two fundamental bottlenecks for real-time distributed dataflow applications at the same time. We are working to implement a new channel interface based on the Xilinx® 10G/25G High Speed Ethernet Subsystem in order to enable interoperability with standard switched networks, either to support input and output streams (e.g. UDP over IP) or to implement a switched network topology.

References

[1] R. Ammendola, A. Biagioni, C. Chiarini, A. Ciardiello, P. Cretaro, O. Frezza, F.L. Cicero, A. Lonardo, M. Martinelli, P.S. Paolucci et al., *Apeiron: composing smart tdaq systems for high energy physics experiments* (2023), 2307.01009

[2] R. Ammendola, A. Biagioni, O. Frezza, A. Lonardo, F.L. Cicero, P.S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, P. Vicini, Journal of Instrumentation **8**, C12022 (2013)

[3] R. Ammendola, A. Biagioni, P. Cretaro, O. Frezza, F. Lo Cicero, A. Lonardo, M. Martinelli, P. Paolucci, E. Pastorelli, F. Simula et al., *The Next Generation of Exascale-Class Systems: The ExaNeSt Project*, in *Proceedings - 20th Euromicro Conference on Digital System Design, DSD 2017*, edited by M. Novotny, H. Kubatova, A. Skavhaug (IEEE, United States, 2017), pp. 510–515, 20th Euromicro Conference on Digital System Design, DSD 2017 ; Conference date: 30-08-2017 Through 01-09-2017

[4] P. Kermani, L. Kleinrock, Computer Networks **66**, 4 (2014), leonard Kleinrock Tribute Issue: A Collection of Papers by his Students

[5] W.J. Dally, C.L. Seitz, IEEE Transactions on Computers **C-36**, 547 (1987)

[6] *https://xilinx.github.io/xrt/master/html/index.html*

[7] R. Ammendola, B. Angelucci, M. Barbanera, A. Biagioni, V. Cerny, B. Checcucci, R. Fantechi, F. Gonnella, M. Koval, M. Krivda et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **929**, 1 (2019)

[8] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran et al., Journal of Instrumentation **13**, P07027 (2018)