A Real-time Control Computer for the E-ELT

Document: GF-PDR-07

Interconnect strategy

Version 1.0

19th January 2016

Change Record

Version	Date	Author(s)	Remarks
0.1	11 Jan 2016	D. Gratadour	Initial skeleton version
0.2	11 Jan 2016	M. Lainé	Minor additions
0.5	15 Jan 2016	M. Lainé	0.5 version
1.0	19 Jan 2016	M. Lainé	Final version

Applicable Documents (AD)

These are the Green Flash PDR documents

No.	Title	Reference	Issue	Date
AD01	Introduction	GF-PDR-01		
AD02	Management plan and WP definition	GF-PDR-02		
AD03	Requirements Specification	GF-PDR-03		
AD04	System Architecture	GF-PDR-04		
AD05	Distributed GPUs for real-time HPC	GF-PDR-05		
AD06	FPGA Solution for hard real-time	GF-PDR-06		
AD07	Interconnect Strategy	GF-PDR-07		
AD08	Interface Control Document	GF-PDR-08		
AD09	Supervision Strategy	GF-PDR-09		

Reference Documents (RD)

These are documents external to the Green Flash project

No.	Title	Reference	Issue	Date

Acronyms and abbreviations

Table 1 Acronyms and Abbreviations

AD	Applicable Document
AO	Adaptive Optics
CANARY	Durham/LESIA on-sky AO demonstrator
CPU	Central Processing Unit
CUDA	NVIDIA GPU based software development language
DARC	Durham AO Real-time Controller
DDS	Data Distribution Service
DM	Deformable Mirror
DRAGON	Durham laboratory-based AO demonstrator bench
ELT	Extremely Large Telescope
E-ELT	European ELT
ESO	European Souther Observatory
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HLS	High Level Synthesis
HPC	High Performance Computing
MIC	Many Integrated Core
MVM	Matrix-Vector Multiplication
NIC	Network Interface Controller
PCIe	Peripheral Component Interconnect express
RD	Reference Document
RTC	Real-Time Control
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SPARTA	ESO VLT AO Real-time Control System
SHERE	VLT Planet finder instrument
UDP	User Datagram Protocol
UK ATC	United Kingdom Astronomical Technology Centre
VLT	Very Large Telescope
WFS	Wave-Front Sensor
WP	Work Package

Table of Contents

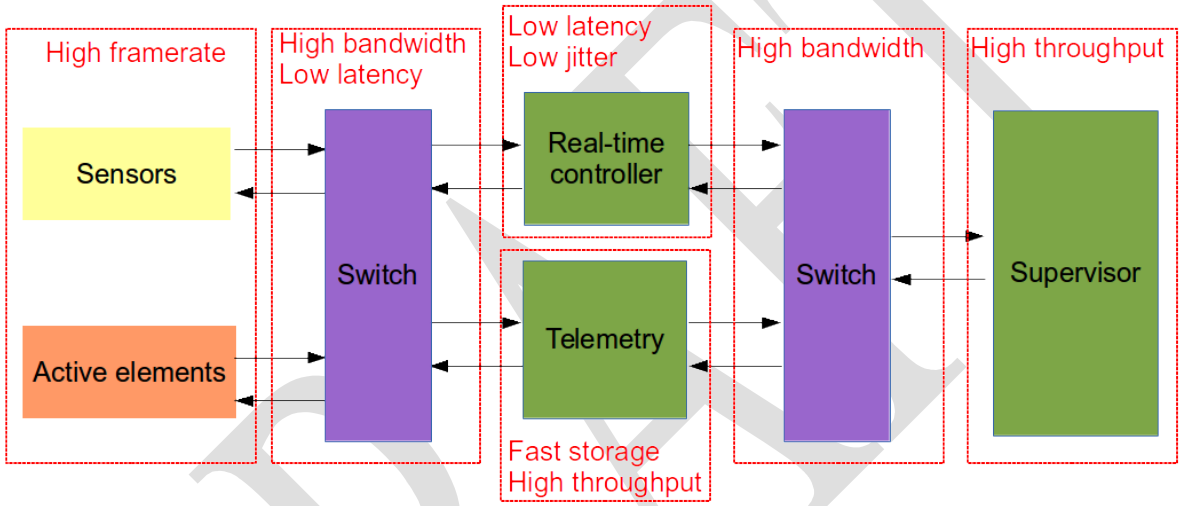
1 Scope

This document specifies the interconnect strategy that will be adopted in Green FLASH.

The first part of this document (section 2) is some sort of white paper for the concept of smart interconnect. In the second part, we describe the specification of such interconnect for the green FLASH project and an implementation plan.

2 Concept of smart interconnect

Data movements in an AO RTC are complex and involve several data flows of different natures and properties. The following figure sketches the high level architecture of an AO RTC.



2.1 Real-time AO Data pipeline

This is the low latency, low jitter data pipeline. The rate of streaming data from the sensors is imposed by the observing conditions (turbulence speed, brightness of the guide source) and the requirements in terms of image quality and stability at the output of the telescope. To ensure stable performance, the overall latency of the "acquisition – processing – feedback" process has to be minimal and deterministic. It fixes the core requirements of the RT box. The latter receives raw data from sensors through a custom format (to enable fast readout) and performs a first level of reduction (the baseline being batched-centroiding, i.e. centroiding on many small regions) before transmitting the resulting smaller data set to the next component of the data processing pipeline: wavefront reconstruction.

2.2 AO real-time loop optimization

At another level, because they are individually based on distributed architectures to meet the compute performance requirements, the RT box and the supervisor module both require an optimized internal interconnect strategy. Inter-nodes low latency communication is critical to the RT box to ensure deterministic performance but it is also critical to the supervisor module to ensure best performance. In such distributed environments enabling direct memory access (DMA) at the intra- and inter-nodes levels and transfer – compute overlaps are the keys to performance. However, while the RT box and

supervisor module share the same top level requirement of low latency data transfer, they may not share the same interconnect implementation which may be hardware / application specific.

Additionally, the interconnection between these two systems is crucial for operations. The RT box broadcasts telemetry data to the supervisor which processes them through a statistical analysis to compute an updated wavefront *reconstructor* control matrix which is uploaded to the RT box. Given the size of the control matrix (typically 64 Gbytes) at the scale of the E-ELT, the process of providing regular updates without impacting the AO loop is also a challenging aspect, especially if different interconnect strategies are implemented on both modules.

2.3 AO requirements for a smart interconnect

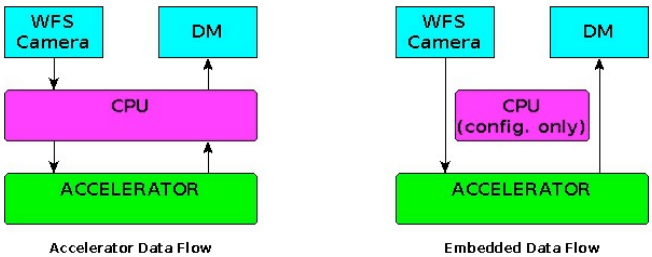
The existence of these three levels of data flows (endpoints to RT, subsystems internal and supervisor to RT) stresses the need for a unified interconnect strategy across the system based on a single versatile solution able to handle the different protocols and hardware topologies. The goal is to increase maintainability and modularity by standardizing the interconnect, identified as the real back bone of the system, but with maximum flexibility in order to keep the ability to implement locally the relevant protocols and links for an optimized approach.

2.4 State of the art in HPC

In commodity clusters similar specifications are addressed enabled by NIC with specific firmware and corresponding drivers and end-user API supporting these features and possibly the interoperability with the run time system for an optimized integration in the programming strategy. The interconnect strategy is a critical aspect of the HPC system architecture. *In an optimized approach, interconnects are tightly integrated with systems components and provide support for accelerating global communication operations.*

Most basic devices are only in charge of the OSI model Data Link layer (2). The majority of NICs though take care of the layer 2 through 5 connecting with the OS kernel interface at the Session layer thus freeing CPU cycles – mainstream protocols such as TCP being handled by the device. Most advanced devices implement 6th Layer protocols (data encryption, conversion, …) usually with the help of an, often proprietary, user API and dedicated driver (camera adapters, InfiniBand HCA or video transceivers are some examples).

Additionally, FPGAs as well are used in order to free even more CPU resources. Using proprietary IPs and custom HDL/Verilog development to implement on board calculation before datas are handled to the Application layer. This kind of development being very specialized, rather costly and long. An effort has been made with openCL in order to allow developments for FPGAs using its C-like language. But, aside from an Altera implementation of openCL containing I/O channels feature for any source (in fact memory and UDP), the embded dataflow paradigm where part to all of the datas comes from the network fabric is not allowed. Rather it focuses on the accelerator paradigm: loading data, computing, giving results back with interaction with the CPU.



Future interconnects, and associated new I/O middleware and APIs, must be able to provide the capability to overlap I/O communication with concurrent computation. Highly efficient interconnects or networks and close integration of their features with I/O middleware will immediately benefit operators and users of HPC systems. Our efforts here focus on the potential for significant improvements in end-to-end performance (i.e., in the I/O speed) as offered by introducing new features at the interconnect layer. (From ETP4HPC Strategic Research Agenda)

2.5 Proposal for a unified smart interconnect strategy

We propose to develop a concept of smart interconnect, tailored to the AO application, enabling new features at the Network Interface Controller (NIC) level in order to:

- Locally reduce the bandwidth requirement in the cluster by performing simple data reduction tasks on the NIC
- Optimize data broadcasting / re-routing depending on environment topology and occupancy
- Handle both data streams from sensors and nodes inter-communications in a unified approach

This work aims at providing a high performance low latency interconnect solution, based on standards and exploring the use of new smart features in the context of real-time control for AO. It requires to provide access to the enabled features at the middleware and runtime system levels. The NIC could be defined as an actor of the data flow processing, available at the programing model level to leverage both RDMA capabilities for efficient data routing or smart broadcasting and generic computing blocks for acceleration.

The developed solution will host smart features including tailored DMA engines, able to be adapted to different nodes topology, for a tighter integration of system components and efficient data accesses across the system. Computing blocks, will also be integrable in these data flows design, to leverage the available FPGA compute capacity and increase the global energy efficiency of the system. The details of the implementation, relying on QuickPlay, are described below.

In our approach, the user can build, in a unified and simplified framework, his own data flow design on the NIC (including communication protocols and computing blocks) and the driver and end-user API to provide access to these features at the middleware and the programming model levels.

Enabling smart features on the NIC can be done statically, by providing the user access to a predefined collection of features, included in the NIC design, through a driver and an API interoperable with standard middleware. We propose to follow another approach in which the user can build, in a unified and simplified framework, his own data flow design on the NIC (including communication protocols and computing blocks) and use the driver and the end-user API to provide access to these features in the programming model. This can be made possible through the use of an integrated development environment for the FPGA board including FPGA design, High Level Synthesis (HLS) for custom computing blocks, simulation, hardware emulation and end-user API development platform. As the project objective is to find an optimum design for a prototype system, this architecture will evolve significantly during the course of the project.

This concept of a smart NIC will be tested with data streams from sensors, transmitted on standard links and protocols, being reduced *on the fly* while they are transmitted to the compute engines. Hence, handling standard protocols (UDP, RTPS), overlapping communication and computations and direct memory access on the compute engines are the three main features we plan to enable in our smart interconnect concept. We believe these features also address the needs of a wider application spectrum and our goals are to make this concept interoperable with mainstream open source middleware (CORBA, DDS) and, through external collaborations, to study the possibility of

integrating these features in new programming models.

3 Interconnect definition

The objective 1.2 of the Green Flash project aims at prototyping a new interconnect solution, based on the FPGA technology, relying on standards and exploring the use of new smart features in the context of real-time control for AO. Our goal is to develop a COTS NIC solution compatible with existing high performance switch solutions, based on standard serial protocols (TCP/UDP and RTPS through 10G to 100G Ethernet), and supported in mainstream non proprietary middleware.

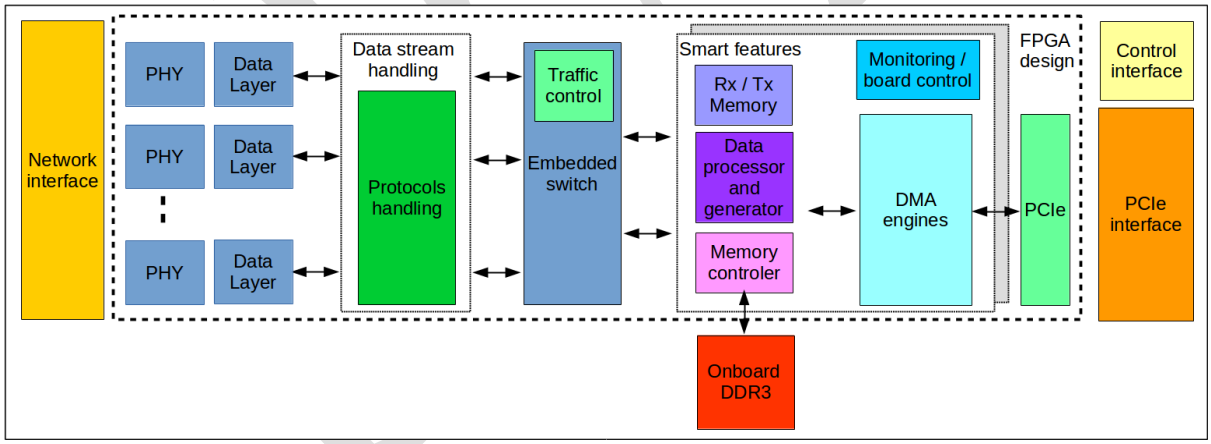
3.1 General architecture

This NIC solution would be based on a modular hardware design with several network link options and supporting several protocols: TCP, UDP, RTPS (for real-time broadcast) and GigeVision (for the interface with WFS cameras) through IP blocks on a FPGA used as a controller between these network links and a baseline PCIe interface. This NIC should be compatible with mainstream middleware: CORBA and DDS, in compliance with SPARTA standards.

Additionally, following the SPARTA WPU approach, we propose to enable advanced features on the NIC to allow the user to define complex data flows involving different interfaces, embedded computing blocks and tunable DMA engines. By doing so, we

- allow the leveraging of available resources on the FPGA to increase the compute performance of a single node
- allow the local reduction of bandwidth requirements for data transactions on the local interconnect (PCIe)
- can introduce tailored DMA approaches to cope with varying node hardware topologies

The following figure sketches the contemplated general design of the smart FPGA-based NIC.



3.2 Hardware description

Our board prototypes will be made around a Kintex 7 FPGA from Xillinx in the first place and then evolve towards an Arria 10 from Altera (see AD06).

3.2.1 10G / 100G Ethernet

As said in the beginning Ethernet is our target for data link layer, at speed of 10Gbps to 100Gbps. Different bandwidths exist that have been or are being defined in the IEEE 802.3 Ethernet standard by

different industrial consortium, 10Gbps (since 2002), 25 and 50 (Sept. 2016), 40 and 100Gbps (since 2010 through 2015 for fiber standardisation). They came in different flavor, as for example 100Gbps Ethernet is made of either four 25Gbps lanes or ten 10Gbps lanes. The IEEE 802.3 working group is working to expand Ethernet speeds further in a near future. As of for 2015, 400Gbps is under development based on 100Gpbs technologies the standard is planned for 2017. 1Tbps was envisioned for 2020 but hasn't been discussed yet. Different bandwidth that are carried by different standard that could lead to a range products responding to different physical interfaces, bandwidth needs and/or financial considerations.

It is also interesting to consider protocols overhead but in a closed environment such as a HPC network – with almost no packet loss, large data exchange and usually skimmed communication patterns (no encryption for example) – TCP efficiency with jumbo frames is generally around 99% – 78 bytes of control headers for 8960 bytes of data par packets, considering the packets are full.

Regarding our prototypes, the Kintex 7 based one will manage a QSFP connection consisting of four 10G Ethernet lanes. Thus providing bandwidth from 10 to 40Gbps, the later being our minimal requirement in the scope of the GreenFlash RTC. E-ELT's WFSs being expected to produce close to 40Gbps of data. The Arria 10 based prototype will, on its part, handle up to four QSFP links. Reaching a mutualised grand total of 160Gbps which could be used as a proof of concept for 100G Ethernet connectors equipped interconnect.

Regarding FPGA based interconnect, the majority of recent products available are PCIe x8 Gen3 capable. Meaning that they can attain a theoretical max bandwidth of 63Gbps. Furthermore that is partially attainable in regards with the kind of transfers occurring over the PCIe BUS. For example high-performance storage controller can approach 95% of PCIe raw data rate, realising long continuous unidirectional transfers leading us around 60Gbps. Thus, the limiting factor will be on the side of the PCIe interface.

3.2.2 PCIe

The interconnect is to be paired with accelerators that usually use the x16 PCIe slots, thus a x8 PCIe interface will be our solution of choice in order to maximize accelerators density in a given node. Both FPGAs family are gen3 capable, Kintex 7 with a soft IP and Arria 10 with up to four hard IP. Being limited by the x8 connector they'll reach the 60Gbps bandwidth mentioned above. That will comply with our Ethernet connectivity needs but in order to reach 100G Ethernet higher internal bandwidth will be needed throught the upcoming 4th version of PCIe specifications.

Hopefully future FPGA SOC (Altera's Stratix10, Xillinx's Virtex Ultrascale+) come with up to 6 PCIe hard IP for x16 gen3 or 8x gen4, both reaching a max bandwidth of 126Gbps. If we consider that in the worst case we reach an occupancy of 80% on the PCIe BUS that leads us around 100Gbps. It is to note that reaching the full 100G Ethernet bandwidth is a feat of strength yet to be attained in a practical environment but theoretically a 100G ethernet interconnect could be prototyped.

However PCIe is made such that point-to-point exchanges between are always scaled down to the lowest. Thus, even if PCIe gen4 is backward compatible with lower generations (programmatically and physically). A device x8 gen4 in a gen3 PCIe domain won't be able to reach full network bandwidth on the PCIe bus. PCIe gen4 being expected to mature in a near future – connectors and

data-rates are to be finalized during 2017 – that could allow further investigation and prototyping as a continuation of the Greenflash effort.

3.2.3 Onboard Memory

Of course smart features should imply some on-board memory. The implementation of smart feature without memory at user's disposal would be a too great constraint for the product to be of interest. We aim our smart interconnect to allow implementation of calculation seamlessly integrated in the communication patterns, therefore it could be legitimate to think that no memory should be needed. Nevertheless, regarding a simple feature such as a collective reduce operation over the network, the interconnect would need memory space in order to retain the intermediate results before handling back the datas to the application level. Meaning a few bytes times the number of elements in the resulting data set should be enough per feature. Besides this feature is intended to sit next to other default and/or user defined features who could also need memory space. In the AO domain we could easily imagine implementing user features from pixel calibration to centroid computation, including dark/background subtraction or flat field correction… Whatever the user could envision in fact. We're considering only a few GigaBytes of memory in order to give users the freedom they'll need to implement as many features they see useful. In any case our product's still an interconnect and should mainly act as one. It may offer some computation offload but is not intended to be an accelerator.

3.3 Development environment

QuickPlay has been developed by PLDA, with the ultimate objective of drastically speeding-up the development and validation of FPGA projects. It enables direct use of FPGA's by users without specific hardware skill, hence making easier the penetration of the FPGA technology in new domains.

The overall process of implementing a design using QuickPlay is straightforward. It consists of:

1. Developing a C/C++ functional dataflow model of the hardware engine
2. Verifying the functional model with standard C/C++ debug tools
3. Specifying the target FPGA boards and interfaces (PCIe, Ethernet, DDR, QDR, etc)
4. Compiling the HW engine

This process is comprehensive from the design of a functional model to the effective validation of the FPGA design on hardware platforms supported by QuickPlay. In order for this simple process to work seamlessly, the generated hardware engine must be guaranteed to function identically to the original software model. QuickPlay uses an intuitive dataflow model that mathematically guarantees deterministic execution, regardless of the execution engine. Such model consists of concurrent functions, called "kernels", communicating with streaming channels, which correlates well with how applications would be sketched on a whiteboard. In order to guarantee deterministic behavior, these kernels must communicate with each other in a way that prevents data hazards, such as race conditions. This is achieved with streaming channels that are:

- Blocking reads and non-blocking writes,
- Lossless
- Point-to-point.

The objective 1.3 of Green FLASH is to complement the ecosystem of this existing integrated FPGA development environment by providing data handling and computational blocks tailored to the AO RTC application, and support for several FPGA options and board designs. The end user API will be made interoperable with mainstream non proprietary middleware. Our goal is to enrich the QuickPlay

ecosystem by providing reusable IP cores for communication protocols and custom compute blocks. Additionally, we plan to provide support in QuickPlay for the various boards developed in the context of the green Flash project to explore the issues related to multiple hardware support and derive a possible road map for a wider application range.

3.4 Smart NIC driver

Regarding drivers, one is mandatory in order to control, configure and use a device from the operating system (OS – Linux here). Usually PLDA (as other constructors) pack their FPGAs with such a driver and a user level API that eases the board controls. The driver provides basic functions that allow read/write operations in the device registers, while the API uses these functions in order to provide larger services such as memory access, on-board engines configuration, networking configuration and such. We wish to keep a simple driver for board registers access and make the end user API operable with mainstream non proprietary middleware as said above.

Another way would have been to make the OS recognize the device as a network adapter connecting with the kernel's network stack interface at the right level allowing the use of the legacy APIs for networking. Unfortunately it might not be an available option given the time for the project. Additionally such an implementation will force to commit ourselves into implementing one particular protocol on board and in the driver, which at this early stage of developments we think might not be a wise idea to do so.

3.5 Ecosystem for the AO application

This project aim to test different RTC topology (using GPUs, FPGAs clusters), thus APIs and runtimes such as CUDA or openCL will probably be present in the software ecosystem, they are already present in most high-end HPC systems anyway. In this regard interconnect DMA capabilities should extend to these memory domains in order to avoid unnecessary copies and buffering. Among the mainstream middleware available, GreenFlash targets mainly MPI (openMPI) and DDS (openDDS). Both being mainstream middleware in the HPC community, and non-proprietary. Besides they provide kernel bypass facilities that can be used at our advantage. Finally both openMPI and openDDS presents a software architecture that allow oneself to plug is own code in a well defined interface, openMPI implementing that to a much greater extent than openDDS. See AD08 for motivations

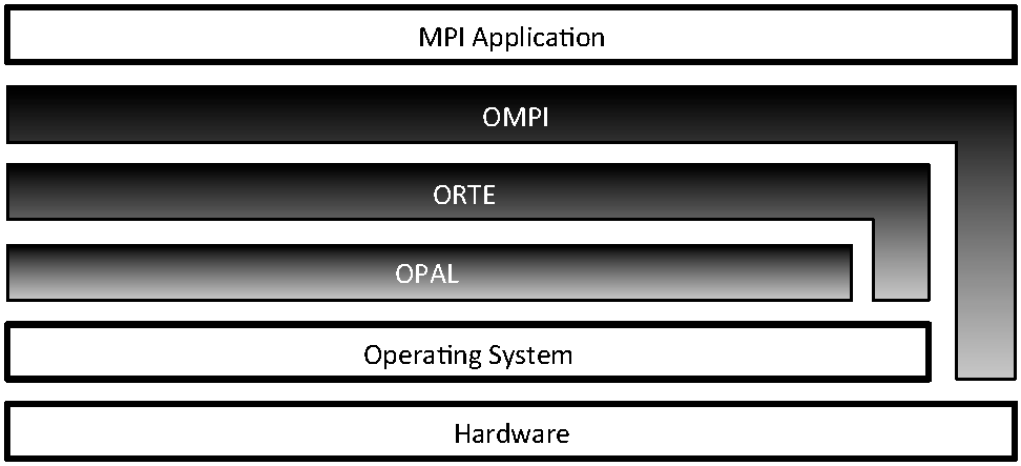
3.5.1 OpenMPI

OpenMPI is an open source implementation of the Message Parsing Interface specification that is developed and maintained by a consortium of academic, research, and industry partners. It has three main abstraction layers,

- *Open, Portable Access Layer (OPAL)*: OPAL is the bottom layer of Open MPI's abstractions. Its abstractions are focused on individual processes (versus parallel jobs). It provides utility and glue code such as generic linked lists, string manipulation, debugging controls, and other mundane—yet necessary—functionality. OPAL also provides Open MPI's core portability between different operating systems, such as discovering IP interfaces, sharing memory between processes on the same server, processor and memory affinity, high-precision timers, etc.
- *OpenMPI Run-Time Environment (ORTE)* (pronounced "or-tay"): An MPI implementation must provide not only the required message passing API, but also an accompanying run-time system to

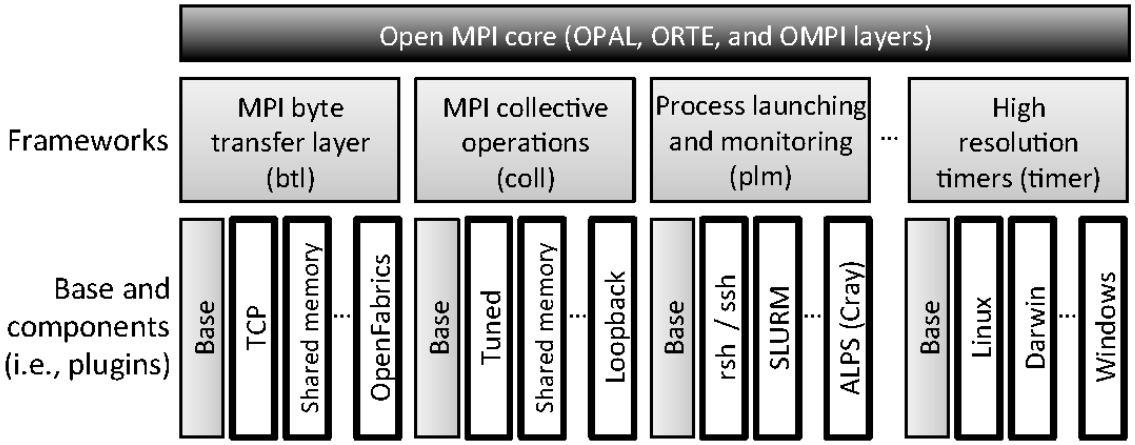
launch, monitor, and kill parallel jobs. In Open MPI's case, a parallel job is comprised of one or more processes that may span multiple operating system instances, and are bound together to act as a single, cohesive unit.

•*OpenMPI (OMPI)*: The MPI layer is the highest abstraction layer, and is the only one exposed to applications. The MPI API is implemented in this layer, as are all the message passing semantics defined by the MPI standard. Since portability is a primary requirement, the MPI layer supports a wide variety of network types and underlying protocols. Some networks are similar in their underlying characteristics and abstractions; some are not.



openMPI abstraction layer architectural view showing its three main layers

Each layer are made of frameworks which can be seen as interfaces implemented into different components. The OMPI layer contains the BTL (Byte Transfer Layer) framework which contains a different components one for each transport layer supported. On the figure below we can see for the BTL, TCP, shared memory and Openfabrics, some of the components available for this framework.

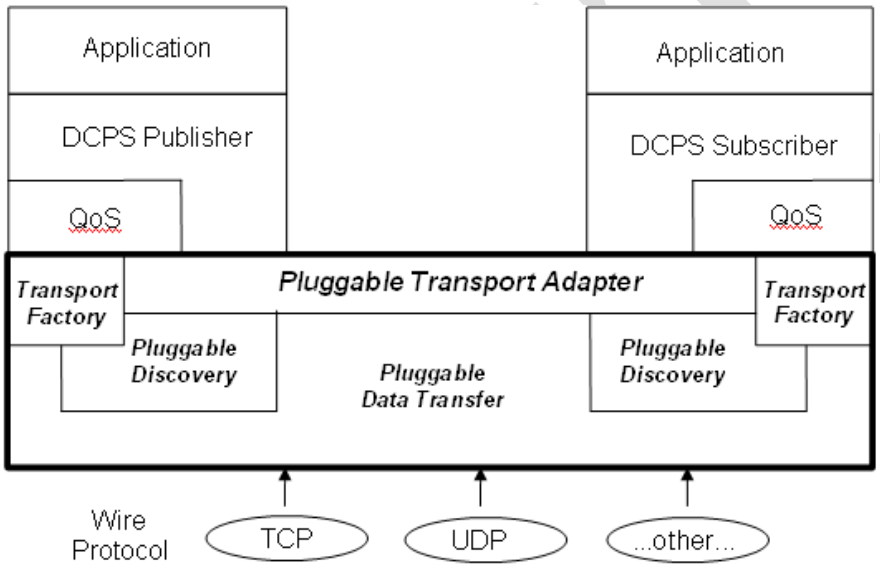


openMPI Framework architectural view, showing a few frameworks and components

This set of layers, frameworks and components is referred to as the Modular Component Architecture. Finally openMPI allows users to define their own framework components. Two components are of peculiar interest to us the Point-to-point Management Layer (PML, fragmenting, reassembly, top-layer protocols, etc.) and the BTL (point-to-point byte movement). In these framework we aim to implement our own components using the Quickplay's driver and user API.

3.5.2 OpenDDS

OpenDDS is an open-source C++ implementation of the Object Management Group's specification "Data Distribution Service for Real-time Systems". OpenDDS follows a publisher, subscriber paradigm. It offers the most common transport protocols which are: TCP/IP, RTPS/UDP, UDP/IP, IP multicast.



Hopefully, openDDS separates the transport from the higher level protocols in order to accommodate with legacy transport protocols. That feature will be of great help to us in order to implement our own transport.

This demonstration of concept on openDDS can be reproduced easily on proprietary RTI DDS, the latter providing the same kind of pluggable transport adapter[1].

4 Implementation plan

This work on smart interconnect is mainly concentrated in WP5 of Green FLASH (see AD02 for a full list of WP).

The goal of this WP is to provide a comprehensive study of a smart interconnect concept, in the context of the AO application, including hardware, firmware, middleware and development environment considerations. Through this integrated approach, we propose to provide a tailored interconnect solution for the AO RTC supporting the various standard communication protocols in the system (internodes, with sensors, etc …) in a unified approach and including the ability to integrate computing blocks at the NIC level. In this WP several NIC prototypes will be delivered and their

1 https://community.rti.com/examples/creating-custom-transport

interoperability with mainstream middleware (including the integration of these features in the development environment) will be addressed.

4.1 Task 5.1: High bandwidth FPGA NIC

Objectives. Develop a common architecture and several versions of a high bandwidth NIC (10 Gbe and infiniband). Dependencies for this work package are the deliverables listed in WP 2.4, including the validation of the system architecture, and the integration in QuickPlay of required building block to develop the Smart NIC, as depicted in WP 6.3.

An iterative process will be used to target "best of class" smart NIC application. It is one of the key benefit of the used development methodology to allow quick analysis of various architecture assumptions and also quick validation on FPGA hardware

So the following sequence of tasks will be executed several time:

1. Target architecture description
2. Application model development
3. Hardware development, emulation and test
4. Middleware adjustment if necessary
5. Verification and Validation, to check proper and bug free implementation at system level
6. Performance measurement, to check adherence of current solution to criteria defined in WP 2.4
7. Turn back to another solution – most likely derived from previous findings.

Obtained results will be compared after a few iterations to check for improvement and select best compromise. Provision is made for the delivery of four different Smart NIC designs, under the assumption that 2 months will be sufficient to run one iteration and an extra month will be devoted to analyze the return on experiment.

4.2 Task 5.2: Smart features to middleware

Objectives. The goal of this task is to define and test a strategy for the implementation of the smart interconnect features at the level of the middleware. This includes communication features for middlewares oriented towards data broadcasting (DDS) and system monitoring (CORBA) but also compute capabilities for middleware such as MPI or more generally new programming models.

On one hand, the efficient DMA capability of the smart interconnect must be integrated in the communication pipeline for an efficient broadcasting of the data. This is of particular importance for the AO application considering the several levels of data flows in the system described in the concept section of this document. Of particular focus will be the interaction between these mainstream middleware and the communication protocols with sensors and deformable optics. This work includes:

- defining a strategy to implement the smart DMA features in open distributions of DDS and CORBA
- providing feedback to task 6.1 and 6.3 for the hardware / development environment implementations
- performing a first level of implementation of these features in the context of the AO application in open distributions of DDS and CORBA
- assess the performance in relation with task 8.2 and WP9 and compare to a baseline solution

based on standard middleware distributions

- study the possibility to expend these features to other applications in relation with task 8.2

On the other hand, the possibility of adding computing blocks in the NIC design could be exploited to:

- locally reduce the bandwidth requirement in the cluster by performing simple data reduction tasks on the NIC
- leverage more computing power in the node by exploiting the FPGA capabilities

The concept of smart interconnect should thus also be studied at the job scheduling / programming model level. We propose to study possible strategies for implementing these smart features at this level through:

- a survey of mainstream runtime systems and their specifications
- the definition of strategies to ensure the interoperability between these runtimes / programming models, the NIC firmware, driver and development environment in relation with tasks 6.1 and 6.3
- a possible first level implementation after down selection of a target runtime
- a performance assessment in the context of the AO application, in relation with task 5.2 and WP9

4.3 Task 5.3: Development environment and IPs

Objectives. Develop the QuickPlay ecosystem to add smart features and open the way to further developments**.** The QuickPlay development environment needs to be enriched with few components / building blocks to enable the development of the NIC features:

- Prototype boards developed for the GreenFlash will be integrated in QuickPlay (ref. WP 4)
- Protocol support: TCP support, InfiniBand (IB) support and Real-Time Publish Subscribe (RTPS) support will be added to currently supported UDP and PCIe.
- Computing IP's: a set of IP's implementing dedicated algorithm will also be added to QuickPlay IP Store
- Finally integration of Camera Link and GigE Vision will be analyzed

Following items will be provided for each building block:

1. The Interface and Property View (IV), which represents the block at the highest level of abstraction. It is a YAML file describing the component in term of in terms of available interface ports and their configurability, the IP high level configuration parameters, the IP nature from a QuickPlay point of view (kernel, memory, interfacing IP).
2. The Functional View (FV) which represents the behavior of the block from a programmer perspective. It is a source code which can be used to emulate the IP. C++ is currently used, but different languages could be considered if required
3. An IP-XACT description of the block, along with a configuration script in case of a dynamic block (that is a block which configuration will depend upon its specific usage in QuickPlay).

Proper integration of the building block within QuickPlay will be verified and validated on hardware.

Functional improvement of QuickPlay:

- How to implement DMA mechanism between the Smart Interconnect and all the element connected to it (GPU's, other accelerator components / devices) will be studied and developed
- How to secure scalability of the number of used NIC devices within a single application will also be studied and developed

From a middleware perspective, each building block will require integration of a specific driver to accommodate Quickplay API. The question of specific API primitives for proper use of the smart interconnect concept and support of existing Network Interconnect libraries will also be addressed.