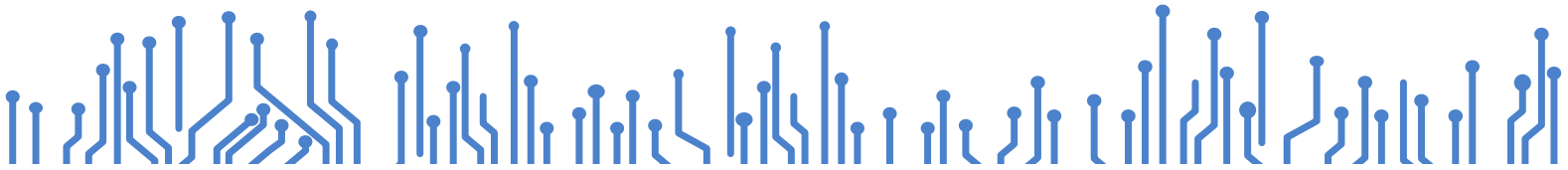
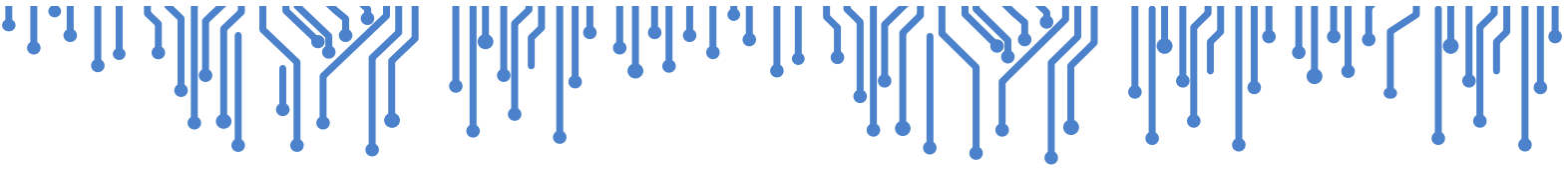
REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems



RECIPE

WP4 Architecture Level and Middleware support

D4.1 RECIPE Report on Reconfigurable Accelerator Infrastructure Deployment



http://www.recipe-project.eu

European Commission | This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 801137

Grant Agreement No.: 801137
Deliverable: D4.1 RECIPE Report on Reconfigurable Accelerator Infrastructure Deployment

Project Start Date: 01/05/2018 **Duration**: 36 months
Coordinator: *Politecnico di Milano, Italy*

Deliverable No:	D4.1
WP No:	4
WP Leader:	UPV
Due date:	30/11/2018
Delivery date:	30/11/2018

Dissemination Level:

PU	Public Use	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

DOCUMENT SUMMARY INFORMATION

Project title:	**REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems**
Short project name:	RECIPE
Project No:	801137
Call Identifier:	H2020-FETHPC-2017
Thematic Priority:	Future and Emerging Technologies
Type of Action:	Research and Innovation Action
Start date of the project:	01/05/2018
Duration of the project:	36 months
Project website:	http://www.recipe-project.eu

D4.1 RECIPE Report on Reconfigurable Accelerator Infrastructure Deployment

Work Package:	WP4 Architecture Level and Middleware support
Deliverable number:	D4.1
Deliverable title:	RECIPE Report on Reconfigurable Accelerator Infrastructure Deployment
Due date:	30/11/2018
Actual submission date:	30/11/2018
Editor:	A. Cilardo
Authors:	A. Cilardo, C. Hernandez Luz
Dissemination Level:	PU
No. pages:	24
Authorized (date):	30/11/2018
Responsible person:	W. Fornaciari
Status:	Submitted

Revision history:

Version	Date	Author	Comment
v.1.0	12/11/2018		First outline
v.2.0	22/11/2018		Complete draft
v.2.2	26/11/2018		Released version for internal review

Quality Control:

	Who	Date
Checked by internal reviewer	M. Zapater Sancho	28/11/2018
Checked by WP Leader	UPV	28/11/2018
Checked by Project Technical Manager	G. Agosta	30/11/2018
Checked by Project Coordinator	W. Fornaciari	30/11/2018

COPYRIGHT

ACKNOWLEDGEMENTS

RECIPE is a project that has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 801137. Please see http://www.recipe-project.eu for more information.

The partners in the project are Universitat Politècnica de València (UPV), Centro Regionale Information Communication Technology scrl (CeRICT), École Polytechnique Fédérale de Lausanne (EPFL), Barcelona Supercomputing Center (BSC), Poznan Supercomputing and Networking Center (PSNC), IBT Solutions S.r.l. (IBTS), Centre Hospitalier Universitaire Vaudois (CHUV). The content of this document is the result of extensive discussions within the RECIPE ©Consortium as a whole.

DISCLAIMER

Contents

1 Introduction

At the architectural level, RECIPE targets the support for deeply heterogeneous systems, with different mixes of accelerators ranging from Graphics Processing Units (GPUs) to Field Programmable Gate Array (FPGA) devices. During the first phase of the project, the partners carried out a careful analysis of the current technological trends, which led to identification of key choices at the architectural level. The basic choices involve server-grade CPUs as well as high-end GPU and FPGA cards targeted at the datacenter/HPC market. Concerning manycore technologies, the consortium discarded the choice of discontinued products, like Intel Phi, which was originally mentioned in the RECIPE proposal, and those targeting non-HPC segments, e.g. the Mellanox Bluefield manycore system. On the other hand, as planned, the project will intensively rely on the FPGA-based infrastructure provided by the MANGO H2020 FETHPC project [5], offering a large-scale *multi-FPGA* setting that perfectly fits the purposes of the RECIPE project. The spectrum of architectures covered by RECIPE is summarized in Figure 1, showing heterogeneous types of RECIPE nodes within the whole RECIPE system.

In line with the plans described in the RECIPE Grant Agreement, the project is interested in the full integration of the compute resources with high-performance fabric (InfiniBand) functions and related system-wide communication mechanisms, extended to FPGA-based accelerators. For that, the MANGO system will play a particularly important role, as it will provide the starting point for implementing high-performance communication interfaces and related drivers/primitives exposed to the user applications.

This deliverable, in particular, describes the outcome of Task 4.1 *Node-level support for disaggregated FPGA resources* (Task Leader: CERICT; Participants: UPV, POLIMI). The task dealt with the integration of FPGA acceleration in RECIPE and the support for networking/communication in the cluster of FPGA devices imported from MANGO for integration of reconfigurable hardware acceleration in RECIPE. The deliverable first summarizes the scope of FPGA-based heterogeneous acceleration in RECIPE. In particular, Section 2 identifies four different FPGA design styles, as seen from the RECIPE application partners' perspective, and their corresponding implications in terms of performance and programmability. Section 3 then presents the communication infrastructure available for connecting the MANGO host system and the FPGA devices, as adapted from the MANGO project. Section 4 provides the details of the node-level commu-

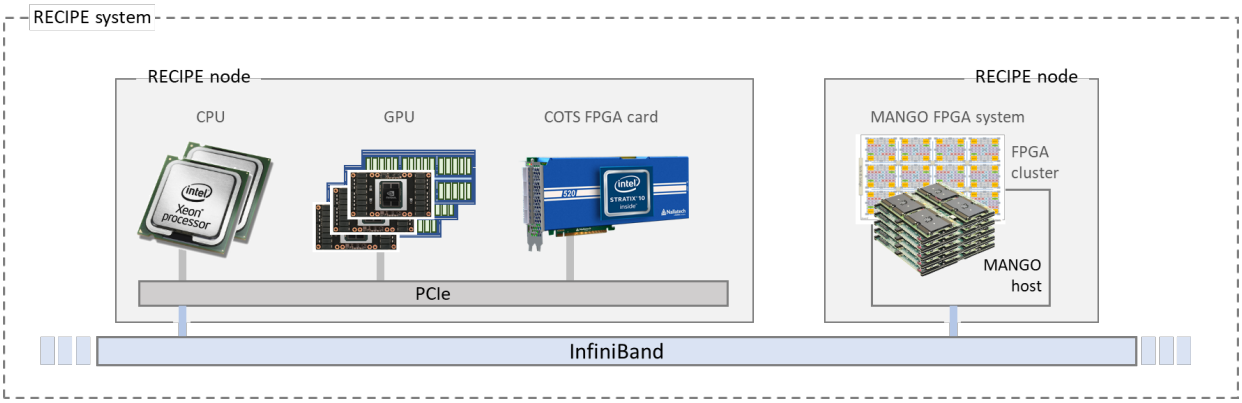


Figure 1: Heterogeneous nodes within the RECIPE system

nication support that will be deployed in RECIPE, while Section 5 describes the infrastructure involving system-level communication.

2 FPGA acceleration in RECIPE

As highlighted in the previous section and shown in Figure 1, FPGAs will be made available in the RECIPE architecture both as commercial off-the-shelf acceleration cards and as dedicated compute nodes provided by MANGO. This will complement homogeneous and standard GPU compute resources, also part of the system. Although FPGAs will not be the only type of heterogeneous acceleration considered in RECIPE, the partners will investigate their potential role as specialized acceleration platforms suitable for some classes of applications and use-case requirements targeted by the project. Because the integration of FPGA acceleration in user applications is particularly challenging, the consortium first evaluated and classified the practical approaches available for its use in HPC settings. The outcome of this evaluation –summarized in this section– is particularly relevant for the RECIPE application partners as it will guide them through the process of assessing the potential use and profitability of the FPGA-based acceleration nodes. In general, the potential parallelism inherent in large-scale applications can fall in one –or multiple– of the following classes: *Data parallelism*, where the same sequence of operations is performed on different pieces of data in an independent fashion; *Thread-level parallelism*, where independent and unrelated flows of execution can proceed simultaneously, each requiring its own control flow; *Instruction-level parallelism*, where instructions of the same sequential flow of execution turn out to be independent of each other and can potentially be executed in parallel; *Pipeline parallelism* where subsequent sequences of operations, e.g. different iterations of the same loop, can be partially overlapped in time by properly handling the pipeline schedule, including the case where dependencies exist between distinct operations in consecutive iterations. Data parallelism and pipeline parallelism are especially relevant for heterogeneous acceleration based on reconfigurable hardware, as FPGAs provide dense arrays of fine-grain resources, often specialized for arithmetic computation, that can be customized in an application-driven fashion with the highest degree of flexibility.

For the development of its heterogeneous abstraction layer, RECIPE will explore four different styles of use for FPGA acceleration, depending on the specific use case requirements:

- full-custom HDL implementation, suitable for performance-critical, relatively simple and regular kernels;
- optimized library-based design, for well-supported kernels to be implemented in hardware, e.g. linear algebra;
- pure-hardware HLS-based design, for non-standard kernels or control-intensive parts of the application that are not performance-critical;
- software-programmed accelerators, particularly the nu+ vector core and associated LLVM-based compiler imported from MANGO, which is suitable for control-intensive parts of the software application that do not match the restrictions of HLS and/or data-intensive kernels benefitting from custom vector-style approaches.

The above design styles will not be mutually exclusive in RECIPE, as different styles can be used for different layers of the kernels being accelerated on the FPGA platform. The main objective in the project is to provide the RECIPE application partners with the full spectrum of implementation choices and guide them through the process of designing and integrating hardware-accelerated kernels. These will be encapsulated under a software application programming interface, relying on the communication infrastructure specified in this deliverable and deployed during the project, for experimenting with various mixes of accelerators, i.e. by replacing selected CPU- or GPU-implemented kernels, identified by the application partners, with their FPGA counterparts, as described in Deliverable 1.2.

2.1 Full-custom design

Full-custom design refers to low-level hardware design, typically performed at the register-transfer level (RTL) by means of a hardware description language (HDL), for example VHDL, Verilog, or System Verilog. This is the traditional approach to FPGA development as seen from the hardware designer's perspective. A general design practice, allowing at least some degree of abstraction compared to pure hardware-level design, relies on the identification of data storage and data processing flows within the application, determining the so-called datapath in the hardware structure, and the separate definition of the logic of control steering the datapath, normally implemented as a finite state machine (FSM) or as a micro-programmed controller. Typical HDLs normally support such design patterns, e.g. well defined coding styles used to describe synthesizable FSMs. Along with modular and hierarchical design, such coding practices allow the hardware developer to contain the complexity of the design process. Nevertheless, because of the very low level of abstraction it offers, full-custom design is in most cases out of reach for ordinary HPC programmers, so this design style is not meant to be directly exposed to the end user. However, the project will also support the integration of custom components within its FPGA-based platform for a number of reasons:

- for simple or moderately complex functions that are particularly performance-critical, extending the range of accelerators with a purpose made component might in general be worth the effort;
- for a few particular applications relying on special low-level building blocks, e.g. cryptographic functions, pattern matching operations, etc., the full potential of FPGA acceleration might only be exploited by a custom design;
- special features available on modern FPGAs are difficult to abstract away and their exploitation, again, might require lower-level design.

In general, for the investigation carried out by the project, full-custom design can also serve another purpose, i.e. provide indications about the upperbounds to the performance improvements that can be achieved by heterogeneous FPGA-based acceleration, as well as a measure of the gap incurred by the other design styles in terms of performance.

2.2 HLS-based design

High-level synthesis (HLS) is today a consolidated approach to the design of FPGA-based components, relying on tools that are able to translate portions of high-level software code, e.g. written in C/C++ language, into an RTL implementation. HLS essentially provides a compilation infrastructure, resembling normal compilers in many respects, which captures a representation of the high-level functionality and maps it to custom hardware structures. That way, HLS provides a number of potential advantages for high-level programmers:

- it relies on familiar high-level languages for functional specification (although in practice many low-level hardware-related aspects are ultimately exposed to the programmer);
- it potentially allows the recognition of recurrent patterns in the code, e.g. related to loop structures, possibly enabling automated transformations/optimizations;
- it allows the development and functional verification of accelerator components in software environments through hardware/software co-simulation and debug;
- depending on the tool, it often provides automated support for the integration of the HLS-generated components in the surrounding infrastructure (on- and off-chip memory, communication interfaces and buses, etc.).

Ultimately, by raising the level of abstraction, HLS can potentially make FPGA programming accessible to end users as well as increase productivity for hardware designers. Current practices suggest that, at least for high-level code exhibiting known patterns, the resulting performance can get close to hand-coded RTL, with some increase in terms of hardware resources (in the order of $10-15\%$ in the best case), although this might not be the case for general code.

As mentioned above, an important aspect of HLS is that it takes care of recurrent optimizations that can be applied to known patterns in the software code, e.g. those enabling potential pipeline parallelism. This particularly applies to loop optimization. In that respect, the two typical optimization techniques that are supported by HLS tools include loop unrolling, which essentially replicates hardware resources to execute multiple loop iterations in a physically parallel fashion, and loop pipelining, which overlaps multiple iterations in time, executing new iterations as soon as dependencies are resolved. Additionally, HLS allows the customization of the memory infrastructure supporting the hardware components, particularly on-chip memory, as well as the automated generation of interfaces to off-chip memory. This is particularly important as parallelized code translated to hardware may easily hit data access bandwidth limitations as the main performance bottleneck. Consequently, HLS tools support advanced on-chip memory implementation and access optimizations, such as port size customization and static coalescing, port sharing, application-driven multi-banking, replication or dedicated allocation of memory modules, double-pumping relying on faster memory clocks, etc. An additional opportunity for optimization exploited by HLS is the use of arbitrary precision data types, particularly for fixed-point variables as well as for floating point numbers. The latter also enable a number of further optimization strategies that might dramatically impact the resource-efficiency (and hence performance and power consumption) of the resulting hardware components, for example removing rounding and conversion between sequences of floating-point operations, balancing trees of operations for reduced delays, etc. Note that this kind of optimizations usually violates the standard-compliant notion of correctness in floating-point operations, so HLS tools are sup-

posed to provide the user with different options and possibly some support to the evaluation of the performance-accuracy trade-offs. In addition to (semi-) automated optimizations, typical commercial HLS design environments provide support for performance analysis and identification of bottlenecks, e.g. limiting factors in loop unrolling or pipelining, memory organization, etc. guiding the end user through the process of code adaptation and optimization.

In typical flows, HLS-generated components become entries in the Intellectual Property (IP) catalog made available by the design environment, and can be instantiated in higher-level FPGA-implemented systems. The reference model for such systems might either take a data flow approach, where hardware components essentially have unidirectional data ports and are directly connected with each other, and a system-based approach, e.g. a bus-based system where HLS-generated components act like memory-mapped devices and use memory-based mechanisms to exchange data with the other components, which might also include programmable soft cores. In some cases, the two approaches can also be combined, with some of the system components being directly interconnected to match the high-level data flow in the original program to the underlying hardware structure. This mapping, however, is very unlikely to be performed automatically by the HLS tool and typically requires programmer's awareness.

The two reference HLS environments that will be considered by RECIPE are provided by the two leading FPGA manufacturers, Xilinx and Intel FPGA. They both provide their own HLS environment, offering all the essential features that are described above plus possibly specific functions supporting integration and verification. For example, Intel FPGA's HLS environment supports x86 emulation, with a compilation tool (i++) featuring command line compatibility with g++ as well as standard debug tools, and integrated cosimulation supporting testbenches with concurrent x86 execution and RTL simulation.

An important development in the area of HLS is the support for OpenCL, a programming model originally introduced as a unified, nonproprietary model for GPU programming. In fact, both Xilinx and Intel FPGA support OpenCL-based design tools and, indeed, these are presented as a different, more abstract design entry solution compared to direct use of HLS (although HLS is still required behind the scenes). While C/C++ programs used with HLS require specific pragmas to control low-level aspects, e.g. memory interface, pipelining, etc, OpenCL kernels do not strictly require additional annotations, unless the developer is seeking improved optimization. That however makes an OpenCL even less performance-friendly than HLS, making its use uncertain in cases where the full potential of FPGAs is to be evaluated and compared to alternative acceleration platforms. Furthermore, while in principle OpenCL is meant to provide a shared model across heterogeneous architectures, ranging from GPUs to FPGAs, the actual execution models that are exploited in the two cases remain different. This divergence essentially boils down to the difference between vector-like parallelism and pipeline parallelism which are exploited in the two cases, plus a number of features that are specifically aimed for FPGA design, e.g. heterogeneous memory support, channels, etc. In conclusion, while OpenCL for GPUs and for HLS-based FPGA design can in principle ensure functional portability across architectures, it is very likely that, for practical purposes, the code needs to be re-written when moving from one type of architecture to the other. RECIPE will consider the use of OpenCL along with HLS for at least one selected kernel of interest for the project, which will be used to evaluate the performance gaps and comparisons against optimized hand-written RTL code.

2.3 Library-based design

HLS-based flows essentially assume that the user application is built from scratch, possibly relying on low-level library components, e.g. custom-configured floating point units. By library-based design we refer here to a design style where *coarse* functionalities, e.g. kernels included in typical HPC applications, are pre-designed and then possibly coupled to other hardware components in the FPGA accelerator. Such functionalities will then be directly exposed to the software HPC applications. Library-based design of course facilitates design reuse and reduces verification cycles, effort, and risk. Importantly, in a HPC setting, it provides a realistic path towards the integration of FPGA acceleration in complex software applications, where it is very unlikely that HPC programmers can develop their own low-level compute kernels from scratch, no matter what the level of abstraction for FPGA design is. Currently, catalogs of high-level FPGA-based functions cover a number of different application domains[1]: Compression, Data Analytics, Financial Computing, Genomics, Machine Learning, Mathematics, Security, Development tools, Video and Image Processing, Networking. Specific examples of libraries include: Intel FPGA Deep Learning (DL) Acceleration Suite [3], Accelize's compression, cryptographic, and transcoding libraries [1], Xilinx BLAS libraries [8], Accelogic's LAPACKrc and compression libraries [2], LabView FPGA Module VIs and functions [4]. The importance of coarse-grained FPGA library components is growing in importance, because of the role of FPGA acceleration in cloud settings (consider for example the case of F1 instances in the Amazon Web Services).

An interesting aspect in library-based design relates to the circuit-level FPGA design cycle. Normally, the FPGA implementation cycle includes a fine-grained tile-based placement-and-routing phase, usually requiring a long searching time, in the order of hours or even days for large-scale design. To reduce compilation time, hard macros consisting of synthesized, placed, and routed circuits can be included in the design, or hierarchical/modular floorplanning can be used. Often, pre-designed library components are used in the context of dynamic partial reconfiguration (DPR), requiring the separation of dynamic logic, reconfigured during operation, and static logic, which is kept unchanged during operation of the FPGA device. Re-used modules with pre-placement and routing can be stored in a library for later reuse. In some cases, PDR is used for enabling remote access and configuration of FPGA resources in cloud-like settings. A few aspects of scientific relevance in this context include efficient approaches for fast modular placement of fine- and coarse-grained FPGA resources, support for reconfigurable modules with multiple contexts, as well as the use of accurate pre-placement information to better direct the global placement. Such developments are however out of scope for RECIPE. The project will consider the integration of high-level library-based FPGA kernels which are statically included in the design, possibly in a pre-routed form. The control and support for data access made available to the library component will be provided by the surrounding FPGA-based communication infrastructure developed by RECIPE, possibly relying on alternative design styles, e.g. HLS or a C/C++ programmable soft core.

[1] see for example: https://www.xilinx.com/products/design-tools/acceleration-zone.html

2.4 Software programmable vector soft-core

RECIPE will explore an additional path towards the exploitation of heterogeneous accelerators, relying on the configurable GPU-like soft-core, called *nu+*, developed by the MANGO H2020-FETHPC project to support both the MANGO Emulation perspective and the Compute perspective. In the first scenario, the RTL core implemented on FPGA was used for the exploration of advanced architecture features deviating from current general-purpose heterogeneous architectures. In the Compute perspective, which is of interest for RECIPE, the nu+ core provides an effective solution as an FPGA *overlay*, offering HPC developers an approach to build tailored processing elements on reconfigurable hardware, reaching higher resource efficiency through customization, yet avoiding the development of a dedicated accelerator from scratch through the support for familiar programming toolchains.

The GPU-like model adopted by the nu+ soft-core exposes useful features for improved resource efficiency in that it provides an abundance of threads executing in a SIMD-like fashion, while reducing control overheads and hiding possibly long operation latencies. Accelerators based on nu+ can effectively exploit multithreading, SIMD/SIMT operation, and low-overhead control flow constructs, in addition to a range of advanced architecture customization capabilities, in order to enable a very high-level utilization of the underlying resources. RECIPE will import from MANGO a baseline implementation (shown in Figure 2), ensuring a set of minimum features including: support for hardware multithreading; data-level parallelism through large-size vector/SIMD/SIMT support; multiprocessor organization allowing non-SIMT execution; lightweight control flow constructs exposed to the programmer, such as predication and mechanisms for optimizing diverging threads and improving datapath utilization; hybrid memory hierarchy providing both coherent caches and non-coherent scratch-pad memory; on-tile performance counters, e.g. for utilization, stalls, instruction counters; facilities for fault/interrupt handling and debug.

Building on the baseline architecture imported from MANGO, RECIPE will explore techniques for customizing the GPU-like core to match the characteristics of the applications being accelerated, effectively exploiting:

- non-standard floating-point precision values as well as dedicated functions like fused operators;
- configurable mixes of instructions, e.g. integer, floating-point, special functions, memory operations, control-related instructions;
- design solutions and tradeoffs for critical components, such as a customizable Register File, along with solutions for building an effective memory hierarchy available to the GPU-like processor;
- new techniques for lightweight control flow and management of diverging threads;
- effective support for software programming models.

Regarding the last bullet above, an important aspect is that nu+ comes with a full LLVM-based compilation toolchain and low-level software libraries. On top of the basic platform, an OpenCL implementation targeted at nu+ can be built to expose a similar level of abstraction as current accelerators, particularly GPUs, without incurring the limitations of the FPGA-oriented

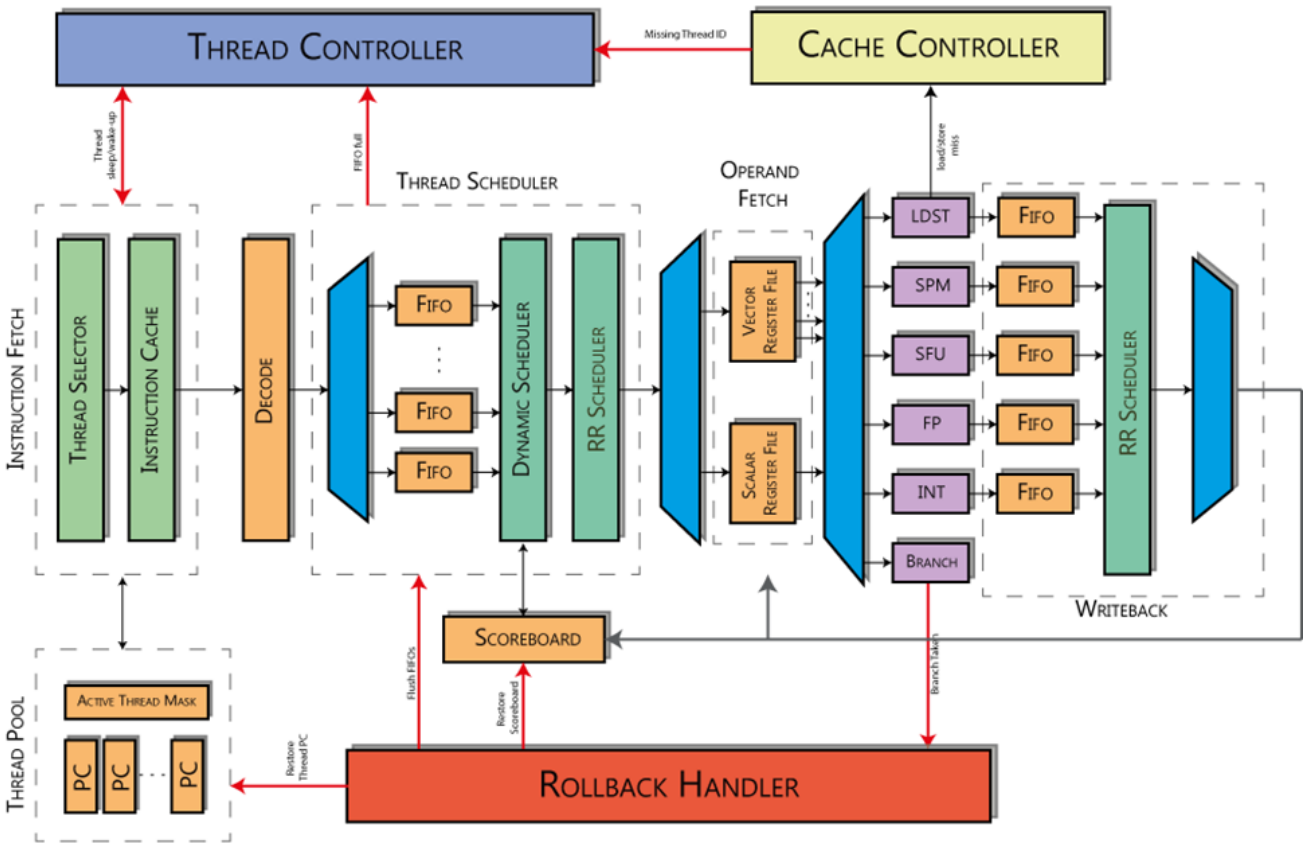


Figure 2: Baseline architecture of the GPU-like accelerator

model used by OpenCL FPGA design environments. Another important aspect is that the basic core in MANGO has been extended to a NoC-based multi-core setting, offering deep configuration capability at the system level. In particular, that means that the NoC-based system can be assembled by composing heterogeneous tiles, part of which can host customized accelerators (possibly developed through a full-custom, HLS-based, or library-based flow), the only requirement being that the specialized acceleration component be equipped with the nu+ NoC interface. As seen from the outside, the system will expose a single interface to the FPGA-based interconnection infrastructure, as described in the following sections, while offering to the accelerator designer the largest spectrum of possibilities in terms of hardware design styles and software programmability.

2.5 Interplay of different design styles

As implied by the above description of accelerator design styles pursued by RECIPE, the FPGA acceleration platform will provide a very large degree of flexibility while exploring the setups that best match the application requirements. The final setting is likely to rely on a hybrid design style, where performance-critical portions of the applications are accelerated by means of a library-based or a full-custom approach, while the control-intensive higher-level parts of the accelerated kernels, as well as the infrastructure-level communication functionality is developed through more abstract approaches, e.g. relying on HLS and/or C/C++ programmable soft cores.

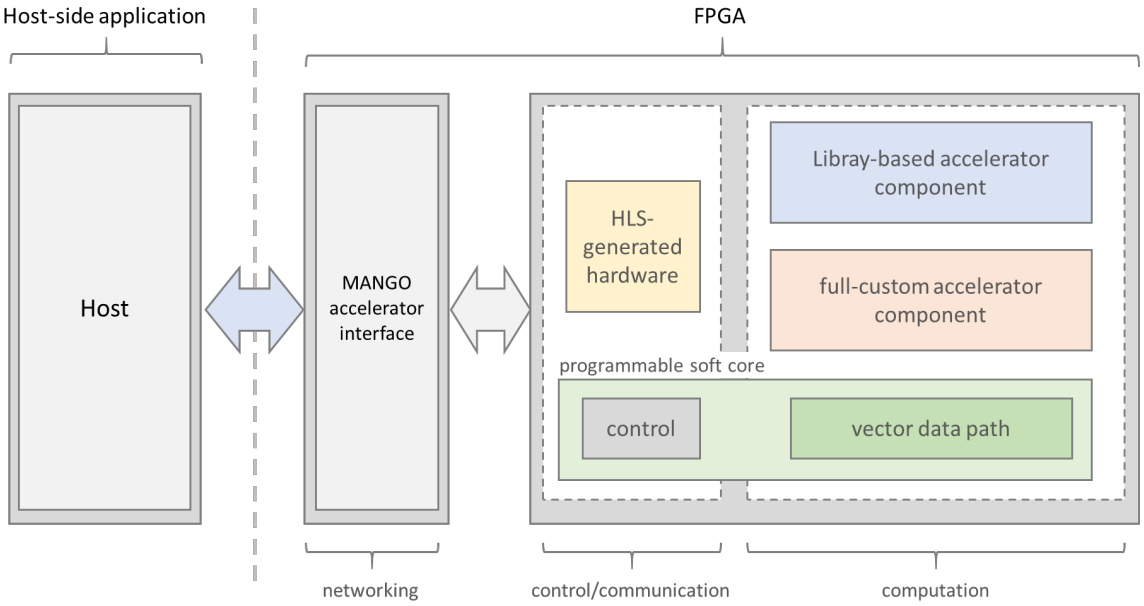


Figure 3: FPGA design styles and their role in the RECIPE platform

The vector processing capability of the programmable nu+ soft core can also be considered for acceleration of performance critical operations. On top of the hardware acceleration infrastructure, the low-level support for communication and interfacing with the CPU-based host system will support the definition of a software API to be exposes to HPC developers, in such a way that the actual design styles and details of the hardware architecture are made mostly transparent. The overall scenario is summarized in Figure 3. Section 4.5 provides the specific details of the interfacing between hardware components within a single FPGA-implemented accelerator, while Sections 3-5 describe the interfacing of the whole FPGA platform to the MANGO host as well as the RECIPE system.

3 Communication between the Mango host and the FPGA platform

At the heterogeneous node level, a high-speed communication interface is required between the host system and the FPGA devices. The main means of communication will be based on PCI express. In particular, PCI express will be the default communication interface between the server and FPGA device. For single-FPGA accelerators, direct PCIe connectivity will be provided. In the RECIPE prototype a number of PCIe direct lanes will be available in the server to allow direct connectivity to FPGA devices. However, due to physical constraints, the interconnection of all FPGA devices cannot be carried out directly with these interfaces. This requires implementing additional interconnection capabilities within the FPGA devices. We use the physical PCIe interfaces to build different communication architectures and topologies to suit the different acceleration needs.

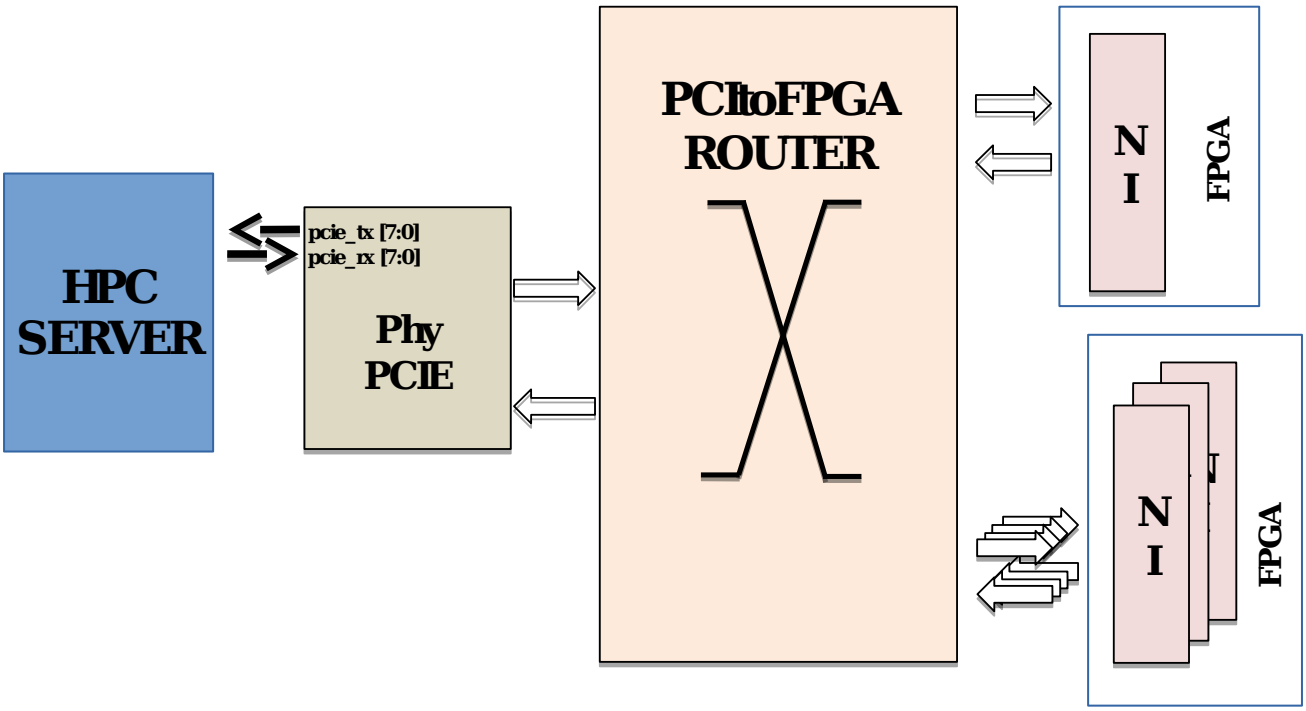


Figure 4: PCIe router and FPGA interconnection

To allow building different communication architectures we added to the FPGA devices a network router acting as the PCIe entry point router. This router allows implementing several receiving and injection ports to read from/write into the interface. The router is attached to the physical interface of PCIe via two unidirectional 64-bit ports. These ports are used to receive/inject traffic from/to the server and distribute it to the appropriate FPGA devices. Figure 4 shows the PCIe entry point router and how it is interconnected with the server and FPGA devices in the RECIPE prototype. Network interfaces (NIs) are implemented in the FPGA devices to receive and send data from/to the PCIe to the rest of FPGA devices.

Additionally, we added the possibility of using DMAs as an efficient transfer mechanism between memories and FPGA accelerators. However, the traffic between the server and the acceleration system must be properly designed in order not to become a bottleneck. Care must be taken in its design in order to keep the premises set within the project for QoS guarantees. We also support the utilization of virtual networks to ensure bandwidth guarantees.

4 Node-level communication

The node-level communication support that is provided in the context of RECIPE is based on the specific needs of the different FPGA acceleration design styles described in Section 2, here referred to as acceleration *modes*. To cover the intra-node needs we envision two scenarios: (1) Intra FPGA communication and (2) FPGA-to-FPGA communication. Note that these two communication scenarios are not required in every acceleration mode. For instance, full-custom RTL designs and HLS-based designs might not require intra-FPGA communication in situations

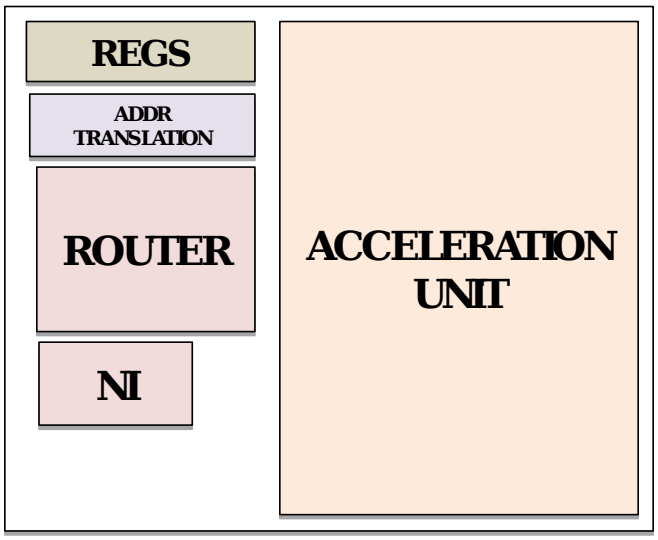


Figure 5: Tile Building blocks.

where an isolated acceleration component is fitted to a whole FPGA device. However, for these acceleration modes we may require support for FPGA to FGPA communication if the resulting acceleration implementation does not fit in a single FPGA device. In this latter case, specific means for partitioning the algorithm are required. On the contrary, acceleration modes using arrays of smaller acceleration units require having efficient intra-FPGA communication means to fetch and write data from/to the different memory controllers.

Node-level communications rely on the utilization of the network infrastructure developed in H2020 MANGO. Hereafter, we describe the main components used in this network and how they relate to each other.

Tile-based architecture. In order to simplify the communication infrastructure required for the FPGA nodes, we have opted for a tile-based design where accelerators and memory modules are mapped to. In this approach, the resulting architecture reverts to a homogeneous set of tiles laid out in a 2D mesh configuration. Each tile offers the same functionality for communication but within each tile a different accelerator and memory configuration may be placed. This means that heterogeneity is possible at the tile level. Figure 5 shows the main building blocks of the tile.

Mango Router. The router that is used to perform node-level communications implements up to five identical bidirectional interfaces. Each of those interfaces may connect to another router located on a neighbouring tile or to the local NI. The router architecture is especially suited to implement a 2D mesh topology. The router implements connectivity towards North, East, West, and South. Each interface implements two unidirectional ports with the same signals. The router includes buffers and a stall-and-go link level flow control to avoid overflowing buffers at the input ports of the router. The router also allows the possibility of including one or several virtual networks (VNs) and virtual channels (VCs). Programmable weighted arbitration is also provided in this router to improve performance guarantees for applications with strong real-time requirements.

Network Interface. The network interface (NI) module acts as the interface between the accelerator unit and the network and/or the rest of the components. The network interface

allows both local and remote traffic. Traffic generated by the accelerator and targeting nonlocal entities (for instance, a memory located on another tile) is encapsulated in a message, pipelined through the NI module, and injected into the router. Local traffic accessing local configuration register or a memory module in the same tile is also encapsulated and transferred using the network interfaces.

Memory Controller (MC). While the deployed network allows exchanging data between acceleration units, we expect communications from accelerators to off-chip memory to involve a significant fraction of the overall communication. We incorporated memory controllers attached to specific locations in the network to allow accessing the different memory modules provided in the FPGA boards. Memory controllers deal with heterogeneous memory interfaces. In the current version of the RECIPE prototype we have DDR3 and DDR4 memory modules but we may also incorporate high-bandwidth memory (HBM) interfaces once they become available in the market. Using HBM allows effective data movement from the host to the FPGA memories. Another important aspect of the memory controller is its placement in the FPGA cluster. The location of the memory controller can affect significantly the performance for latency sensitive applications so finding the appropriate location is crucial to ensure an effective utilization of resources. Note that in the multi-FPGA cluster, memory modules are shared between the different FPGA devices and only one MC can have access to the memory module at a time.

4.1 Intra-FPGA communication

For the intra-FPGA communication infrastructure we rely on the network infrastructure developed in MANGO. This network infrastructure is based on a 2D mesh network on-chip that implements one or several physical and virtual networks using wormhole routers to interconnect the different tiles of the system. In RECIPE we use tiles to wrap acceleration units in order to allow accelerators to communicate with each other and with memory when the adopted acceleration design style demands such functionality. The architecture of the tiles and the NoC infrastructure that is deployed within the FPGA to accommodate more than one acceleration unit in a single FPGA device are depicted in Figure 6.

Note that, as mentioned before, intra-FPGA communication is only required for acceleration modes including more than one acceleration unit in the FPGA.

4.2 FPGA-to-FPGA communication

Having the ability to exchange data across different FPGA units is required to support acceleration modes in which accelerators span several FGPA devices. To this end, we rely on the same network-on-chip architecture defined before. However, when dealing with FPGA-to-FPGA communication the most important limitation comes from the reduced number of pins available on the FPGA devices. To alleviate this problem, we incorporated pin multiplexing techniques to allow maximizing the communication throughput across different FPGA devices. Pin multiplexing can be implemented relying on specific IP cores offered by FPGA providers and requires careful design of the timing aspects of designs implemented in the FPGAs to achieve high-speed communications.

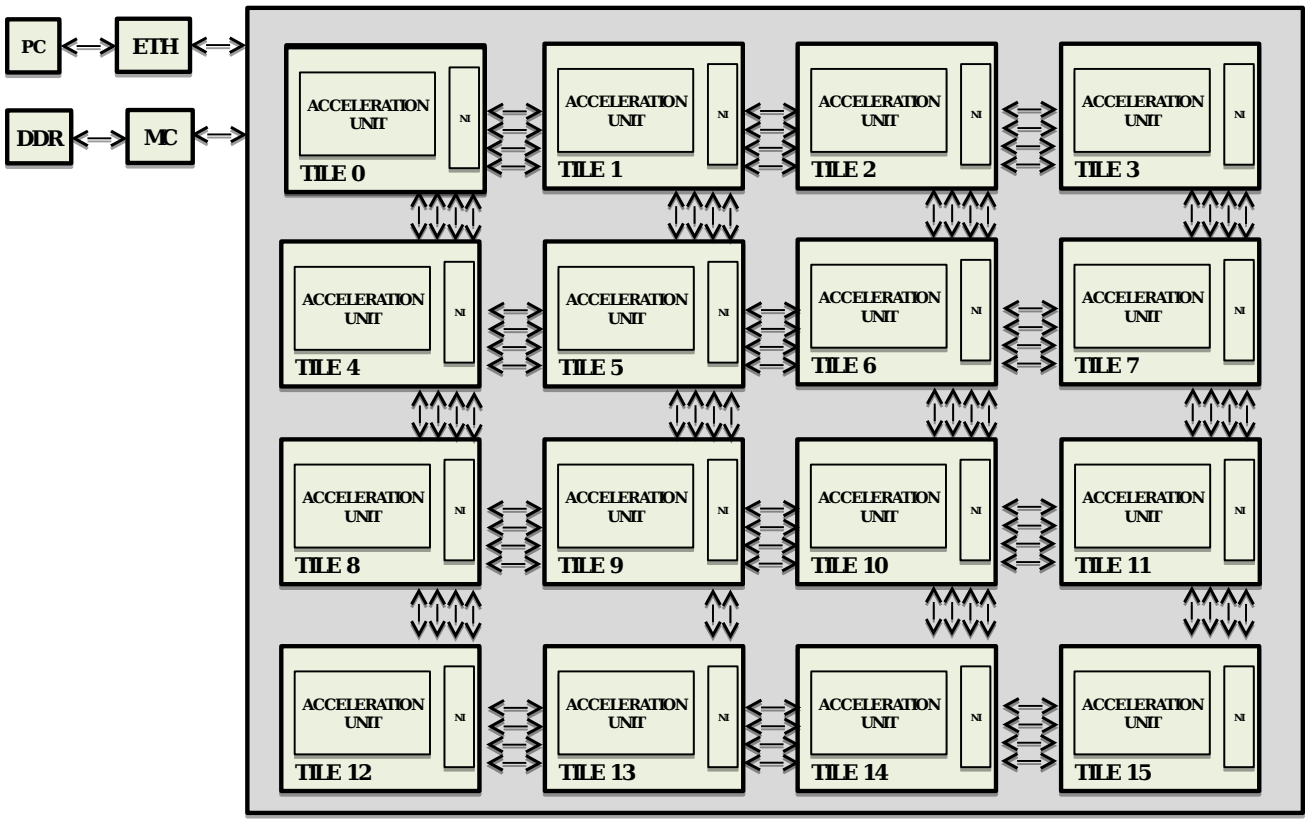


Figure 6: Logical view of an array of accelerators included in one or more FPGA devices using the tile-based approach.

4.3 Efficient Memory Transfers

To transfer data from the memory on the FPGA acceleration nodes to the memory on the Server, DMA engines are used. Data is routed to specific Input/Output devices and then sent to the Host. These transfers are especially useful to transfer the output data generated by the accelerators to the Host. In the other direction, from the memory on the Server to the memory on the FPGA system, data is routed following the same approach. Data is read from Server memory and directly sent to the communication interface, and once it gets to the acceleration system, data is routed through the same data channels as the local DMA transfers. These transfers are useful to send the data to be processed by the accelerators.

The DMA can transfer data from the local memory to other memories within the HPC system, or to the application memory space. This flow of data goes through the NI attached to the memory controller. After analyzing the message format, the NI forwards the data through the network to reach another tile, or to a local IO device on a specific data channel. At the network level, a one-to-one association exists between the virtual networks, the DMA channels, and the data channels.

The DMA can also be programmed to transfer data directly to accelerators (the UNIT). This is especially important for acceleration modes that can take as input large amounts of data for their processing. This is the case, for instance of acceleration modes using full-custom RTL and software programmable vector soft cores.

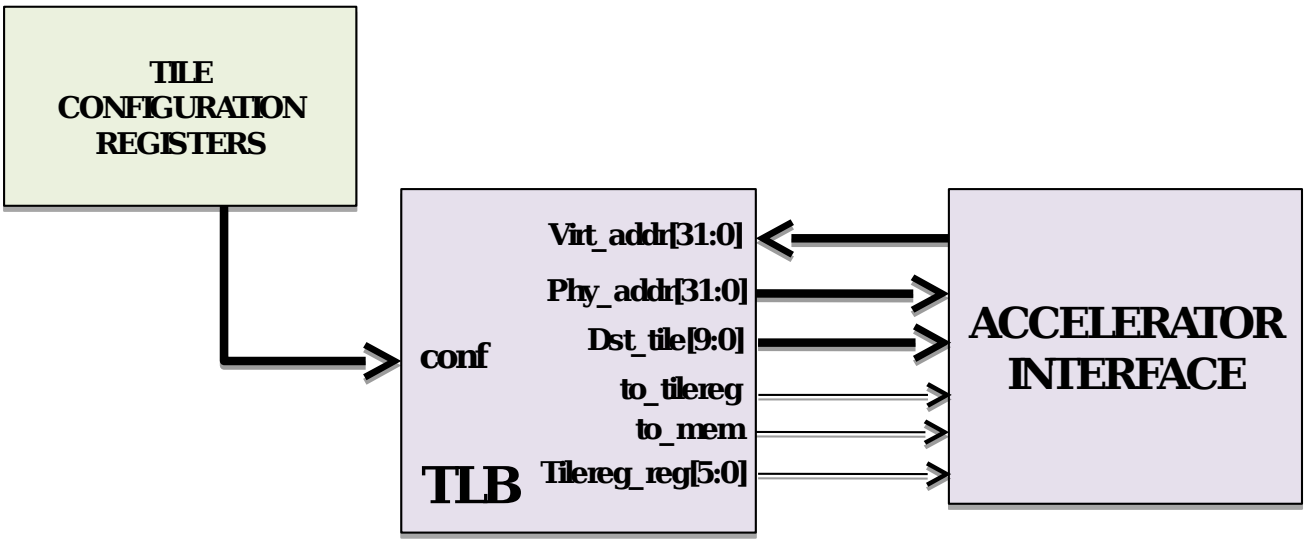


Figure 7: Address translation mechanism.

4.4 Address Translation

One important aspect of the system is how memory is addressed by the accelerators. In RECIPE we support different acceleration schemes, and each of them may need to access memory in a specific manner (e.g locally or remotely, enforcing coherence or not). In addition, the memory will be shared also by the server as the applications will interoperate with their kernels running on the accelerators and will share memory. The approach we follow for memory management in RECIPE is to offer accelerators the possibility to have their own virtual address space (bounded to 4GB memory space) and the resource manager takes control of the physical memory available on the cluster. Specific registers within the tile have been implemented to map accelerator's virtual addresses into the final physical memory resources. For acceleration modes where this address virtualization scheme is required, a simple address translation module (a.k.a TLB) has been implemented in the FPGA. The address translation module allows the resource manager to map virtual addresses into physical ones. This in practice allows a proper communication of the accelerator with memory resources.

The TLB receives a virtual address from the acceleration unit. With this address, the TLB internally computes the physical address and provides back the data for the access to the acceleration module. For regular memory accesses, the accelerator interface contains the physical address to access and the location (tile) where the target memory is located. A regular memory access is identified by setting a specific signal.

Besides this, the FPGA network architecture inherited from MANGO enables accelerators to synchronize with the host system or with other accelerators by letting accelerators access memory-mapped registers. To do this, the TLB can be configured to map those registers to specific virtual memory addresses of the accelerator. Thus, from the standpoint of the accelerator, a read or write operation to a synchronization resource will be translated into a regular read or write operation to a virtual address. However, the virtual address, when translated, will be redirected to the specific register. Figure 7 illustrates the interconnection of the TLB and the acceleration units.

4.5 Accelerator interfacing

This subsection deals with low-level hardware component interfaces, i.e. those adopted *within* the boundary of the heterogeneous Acceleration Unit as called in the schemes shown in the previous sections. Such interfaces are relevant just for the hardware accelerator designer. The Mango tile containing the accelerator itself will in turn be connected to the rest of the system through the communication system described in the previous sections.

Low-level interfaces will in turn work at three different levels:

- Basic hardware components, possibly developed through HLS or custom design, will be equipped with standard-compliant interfaces. Depending on the FPGA technology of choice, this might be one of two standard interconnects, i.e. AXI or Avalon, although the broader adoption of the former makes AXI the preferred option. Two different variants of the interconnect will be considered, suitable for two design approaches: streaming interfaces, made of simple, point-to-point, unidirectional ports (in the simplest instance, just data plus ready/valid signals) that are suitable for situations where the high-level program data flow is directly mapped to a corresponding hardware structure, and memory mapped interfaces (address-based read/write interfaces), allowing more complex interaction patterns based on shared memory access. Note that both AXI and Avalon provide stream and memory-mapped versions of the protocol. The standard compliance will improve the interoperability of low-level components, including the case of modules imported from existing IP catalogs, and will be possibly exploited with different design styles. This choice will mostly benefit the low-level hardware accelerator designer. Regarding the specific case of memory interfaces, only local on-chip memory will be directly supported at this level, including the cases where HLS is used (i.e., coding patterns inferring off-chip memory will not be supported), because access to off-chip memory will be mediated by the external interfaces described in the previous sections.
- Cache-coherent custom accelerators can be integrated in the nu+ many-core infrastructure as heterogeneous cores, as depicted in Figure 8. Importantly, the whole system will appear as a single MANGO tile, or Acceleration Unit. This allows the tight coupling of complex accelerator components, e.g. library-based optimized implementations of some kernel function, and the programmable soft core, enabling C/C++ programming of the whole infrastructure, the possible exploitation of the vector instructions in the soft core, as well as a direct and transparent (shared memory based) interaction with optimized acceleration hardware.
- The whole FPGA-side acceleration system will be interfaced with the outside through the accelerator interface adapted from MANGO, described in the previous sections. The hardware interface is readily available on the nu+ manycore system, as it has already been integrated in the MANGO infrastructure, avoiding any additional design overheads for the developer. For cases where the nu+ soft core is not instantiated and direct use of dedicated accelerators is planned, those accelerators need to be equipped with an interface wrapper, providing an AXI or Avalon interface on the internal side, which will be made available by the RECIPE technology partners.

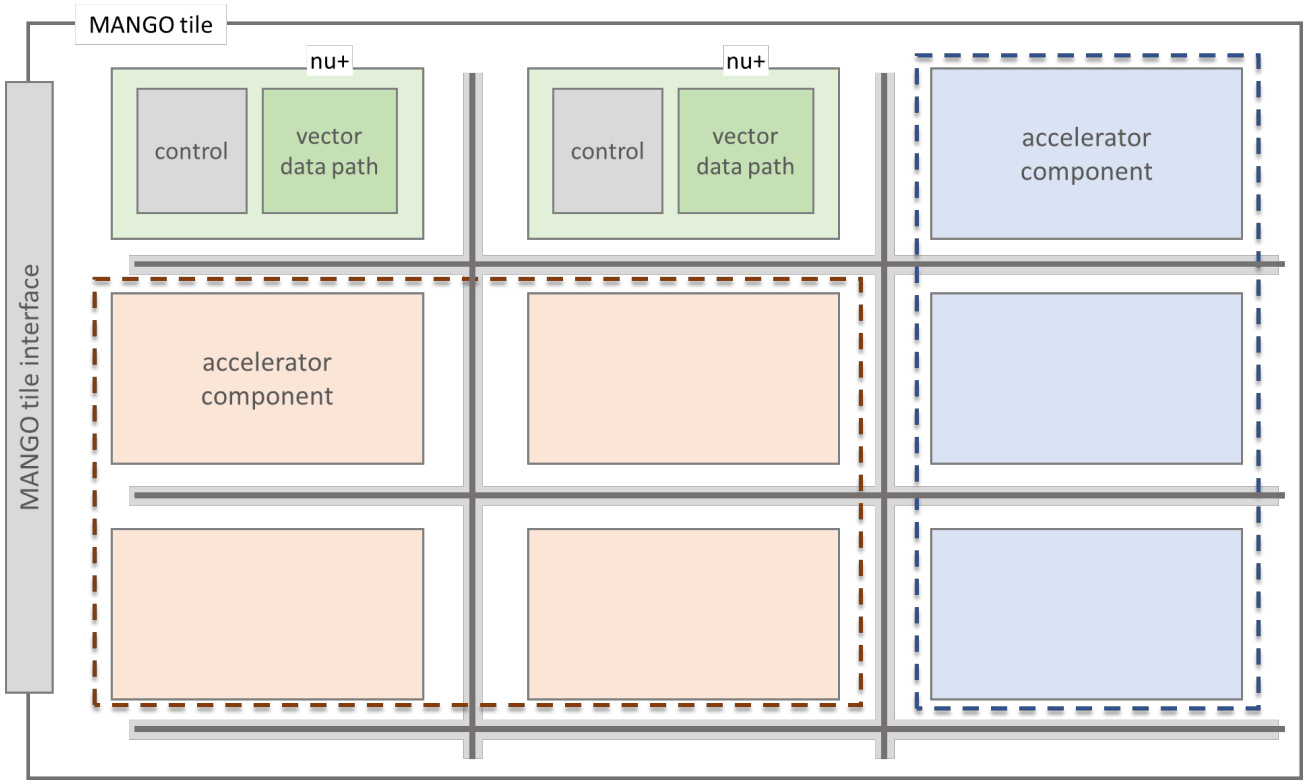


Figure 8: A MANGO tile, corresponding to a single Acceleration Unit, including a heterogeneous nu+ manycore system

5 System-level communication

System level communications in the RECIPE prototype correspond to those carried out between the different nodes in the HPC system. The main communications involving different nodes will correspond to data transfers between the nodes of applications which require to communicate, and will be mainly carried out by both point-to-point and collective communication primitives implemented in the message-passing application programmer interface (MPI). In this section we elaborate on the capabilities of the InfiniBand network substrate to allow the efficient implementation of the aforementioned MPI primitives and on the potential extensions required to add the support for node-to-remote accelerator communications as required to implement the concept of disaggregated acceleration.

5.1 InfiniBand network capabilities

For the implementation of the concept of disaggregation we will rely on the ultra-low latency and high throughput offered by InfiniBand networks. Unlike Ethernet, in InfiniBand networks packets are never lost ensuring that a higher level of quality of service can be offered to the different applications.

The main quality of service (QoS) capabilities of InfiniBand rely on the link layer architecture [7].

The link layer encompasses packet layout, point-to-point link operations, and switching within a local subnet. QoS is supported by InfiniBand through Virtual Lanes (VLs) a.k.a virtual channels in interconnection networks' terminology. VLs provide separate logical communication links which share a single physical lane. Each link can support up to 15 VLs (plus one that is reserved for management). Service levels (SLs) can be defined to match a certain QoS level. Each link can have a different VL, and SL can be used to provide each link with a given priority. The way this can be configure in InfiniBand is through the utilization of SL-to-VL mapping tables.

In addition to the definition of service levels, performance guarantees of real-time HPC applications can be improved by leveraging the weighted arbitration implemented by Mellanox InfiniBand switches. In these switches, arbitration between traffic of different VLs is performed by a two-priority-level weighted round robin arbiter. To make weighted arbitration effective the arbiter has to be programmed with a sequence of (VL, weight) pairs and one should define the maximum number of credits that can be processed before low priority ones are served.

5.2 Disaggregated FPGA acceleration

In RECIPE we take advantage of the low latency and high-bandwidth capabilities provided by InfiniBand to effectively implement disaggregated acceleration support. Disaggregated accelerators can be accessed from any server in the HPC infrastructure as if they were virtually attached to the server where the application is being executed. The idea is to allow one application to potentially utilize all acceleration units available in the HPC facility accessing them in a transparent way through the interconnection network.

The way the offloading process (involving the data transfers from the host to the accelerators) is carried out depends on the particular programming model employed by the application developers. For instance, specific MPI primitives can be implemented to allow send and receive information using modified MPI Send and MPI Recv primitives to a computing element in the FPGA. The way the FPGA handles such MPI primitives is implementation dependent. For instance, one possibility is to centralize and handle message exchanges at the server side which on the one hand will reduce the specific support for MPI in the FPGA device, at the expense of increasing the latency of communications. The other alternative will consist in implementing a subset of the MPI standard either with a software library for tasks implemented on programmable soft/embedded cores or by directly implementing a hardware Message Passing Engine in the FPGA.

As an alternative to the implementation of the concept of disaggregation using MPI, we can consider in the context of RECIPE the utilization of OpenCL-based disaggregation by adapting the OpenCL runtime to have the desired functionality. This option does not seem to be a solution as clean as the ones based on MPI, but might fit better the requirements of FPGA accelerated applications. There exist already implementations of such OpenCL functionality, like SnuCL [6].

The decision of which particular model for disaggregated acceleration will be employed in RECIPE is not yet taken and will depend on how the different applications match the different options and what is the performance that one or another approach is expected to achieve in the context of the specific RECIPE use-cases.

References

[1] Accelize, Retrieved Nov 2018. https://www.accelize.com.

[2] Accelogic, Retrieved Nov 2018. http://accelogic.com.

[3] Intel FPGA Deep Learning (DL) Acceleration Suite, Retrieved Nov 2018. https://www.intel.com/content/www/us/en/programmable/solutions/technology/artificial-intelligence/solutions.html.

[4] LabVIEW 2013 FPGA Module Help, Retrieved Nov 2018. http://zone.ni.com/reference/en-XX/help/371599J-01.

[5] MANGO: exploring Manycore Architectures for Next-GeneratiOn HPC systems, Retrieved Nov 2018. http://www.mango-project.eu/overview.

[6] Center for Manycore Programming. University of Seoul. SnuCL Suite. OpenCL Frameworks and Tools for Heterogeneous Clusters, Retrieved Nov 2018. http://snucl.snu.ac.kr/snucl.html.

[7] Mellanox. Real Solutions for the Challenges of the Post-Petascale Era, Retrieved Nov 2018. http://www.mellanox.com/page/white_papers.

[8] Xilinx. Xilinx BLAS libraries, Retrieved Nov 2018. https://github.com/Xilinx/gemx.