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

RECIPE

WP4

D4.4 RECIPE Fault-tolerance and Quality of Service Support

http://www.recipe-project.eu

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: D4.4 RECIPE Fault-tolerance and Quality of Service Support

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

Deliverable No:	D4.4
WP No:	4
WP Leader:	R. Tornero
Due date:	31/10/2019
Delivery date:	31/10/2019

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

D4.4 RECIPE Fault-tolerance and Quality of Service Support

Work Package:	WP4
Deliverable number:	D4.4
Deliverable title:	RECIPE Fault-tolerance and Quality of Service Support
Due date:	31/10/2019
Actual submission date:	31/10/2019
Editor:	R. Tornero
Authors:	R. Tornero, A. Cilardo, C. Hernandez
Dissemination Level:	PU
No. pages:	17
Authorized (date):	31/10/2019
Responsible person:	W. Fornaciari
Status:	Draft Working **Final** Submitted Approved

Revision history:

Version	**Date**	**Author**	**Comment**
v.0.1	04/10/2019	R. Tornero	Create document structure
v.0.2	10/10/2019	C. Hernandez	Fill QoS section
v.0.3	15/10/2019	A. Cilardo	Fill Checkpointing section
v.0.4	18/10/2019	C. Hernandez	Finish first draft
v.0.5	28/10/2019	R. Tornero	Ready for internal revision
v.1.0	31/10/2019	R. Tornero	Final version after internal revision

Quality Control:

	Who	Date
Checked by internal reviewer	Marina Zapater	30/10/2019
Checked by WP Leader	R. Tornero (pro-tempore, G. Agosta)	31/10/2019
Checked by Project Technical Manager	G. Agosta	31/10/2019
Checked by Project Coordinator	W. Fornaciari	31/10/2019

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 Politecnico di Milano (POLIMI), Universitat Politècnica de València (UPV), Centro Regionale Information Communication Technology scrl (CeRICT), École Polytechnique Fédèrale 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

Executive Summary

This report documents the progress in the hardware support for Quality of Service (QoS) and fault-tolerance developments for the RECIPE prototype. This document presents the work carried out in T4.4 so far, and the resultant hardware mechanisms that will enable the software stack developed in WP2 to accomplish with its goals. We describe how to leverage Infiniband QoS features in the RECIPE prototype to allow critical application to have performance guarantees and the development of hardware support for checkpointing in FPGA devices and its integration in the current FPGA vendors design flow.

1 Introduction

In this deliverable we describe the developments carried out to provide the RECIPE prototype with hardware support QoS and fault-tolerance. In particular, we describe how Infiniband network features can be exploited to enforce performance guarantees at the network side and thus, avoiding the network background noise that can come from other co-running applications or by the maintenance operations required by a data-center/supercomputing facility. For fault-tolerance developments have focused on developing support for check-pointing in FPGA accelerators. Given that the interest for employing FPGAs in the context of HPC applications has significantly increased in the recent years having support for check-pointing in these devices is becoming a necessity.

The rest of the document is structured as follows. In Section 2, we describe and evaluate the QoS features of the Infiniband interconnect network deployed in the prototype. In Section 3, we introduce how leverage checkpointing techniques at FPGA level and how integrate them in current design flows of FPGA vendors. In Section 4, we propose a plan to integrate those developments with the RECIPE software stack. Finally, we end up with the conclusion of this work in Section 5.

2 Quality of Service over InfiniBand Network

Infiniband network architecture [4] provides QoS features at link and switch level that can be leveraged to improve the real-time guarantees of applications executing in the RECIPE prototype. In this deliverable we show how the Infiniband card adapters can be configured to ensure QoS in the recipe prototype.

2.1 Virtual Lanes and Service Levels

Infiniband uses a memory-based user level communication abstraction. Communication interface is the Queue Pair (QP) which is the end-poing of a communication link. QPs are implemented at the host side of the channel adapter (HCA). At the same time channel adapters implement Virtual Lanes (VLs). Virtual Lanes enable having data flows isolation withing a shared link. A VL arbiter is used to determine which flow will have access to the link.

Infiniband provides two fields for marking packets with a class of service: the service level (SL) field in the local route header and the traffic class field in the global route header. The SL field is a four-bit field that may be arbitrarily used to indicate a class of service. Infiniband does not define a specific relationship between SL value and forwarding behavior but there is a defined mechanism in the specification to specify a mapping between the SL values and the available forwarding behaviors in switches.

Infiniband switches may implement between one and 15 VLs (in the mellanox switches and adapters from Mellanox we have 8 VLs). A VL is an independent set of receive and transmit resources (i.e. packet buffers) associated with a port. In addition to SL, the local route header contains the VL field that indicates the virtual lane number from which the packet was transmitted. Once packets are received they are placed in the ports receive buffer corresponding to the virtual lane indicated by the VL field.

As a packet traverses the switch from input port to output port, the packet may transfer from one virtual lane to another. Each switch in the fabric contains a table (usually known as the SL to VL mapping table) that selects the output port virtual lane based on the packets SL, the port on which the packet was received, and the port to which the packet is destined. This mapping function permits interoperability on fabrics consisting of switches supporting various numbers of virtual lanes. This implies that, while the VL indication in a packet may change from hop-to-hop, the SL indication remains constant within a subnet. Note that packets within one virtual lane may pass packets in another virtual lane as they transit a switch.

2.2 Configuring Infiniband in the RECIPE prototype

The configuration of Infiniband can be made through OpenSm [3]. OpenSm is an InfiniBand compliant Subnet Manager and Subnet Administrator that allows handling Mellanox Infibiband components. OpenSm has to be running as a daemon for each Infiniband subnet and is the agent in charge of initializing the InfiniBand hardware.

OpenSM cached options file has a set of QoS related configuration parameters, that are used to configure SL2VL mapping and VL arbitration on IB ports. These parameters are:

- Max VLs: the maximum number of VLs that will be on the subnet.
- High limit: the amount of times a high priority virtual lane is prioritized over a low priority one.
- VLArb low table: Low priority VL Arbitration table that allocates weights from 0 to 255 to the differente VLs.
- VLArb high table: Low priority VL Arbitration table that allocates weights from 0 to 255 to the differente VLs.
- SL2VL: SL2VL Mapping table to associate defined VLs to the corresponding SLS to SLs (0-15)

The following text shows the part of the OpenSM configuration file in which the QoS features are configured.

```
qos_ca_max_vls 15
qos_ca_high_limit 0
qos_ca_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_ca_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 0
qos_swe_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_swe_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

In this particular example the systems is configured to use 15 VLs. SL to VL mapping is performed assigning a given SL to the VL matching its numbering (i.e VL=1/SL=1). Parameters including *ca* define the configuration of the channel adapters whereas the parameters including *swe* configure the switches in the network. In the high limit a value of 0 indicates that only a single packet from the high-priority table may be sent before an opportunity is given to the low-priority table. VL arbitration tables (both high and low) are lists of VL/Weight pairs. Each list entry contains a VL number and a weight (values 0-255), indicating the number of credits (packets of a given size) which may be transmitted from that VL when its turn in the arbitration occurs. A weight of 0 indicates that this entry should be skipped. A given VL may be listed multiple times in the High or Low priority arbitration tables. A high_limit value of 255 indicates that high priority VL will always be prioritized over low-priority VLs.

2.3 Testing Bandwidth Guarantees

To test how Infiniband network settings affect performance guarantees we have designed latency and bandwidth test applications based on Message Passing Interface (MPI) programming model. For that purpose, we have based on the Ohio State University Performance evaluation benchmarks available in `http://mvapich.cse.ohio-state.edu/benchmarks/]`. We have selected the benchmarks that measure MPI latency and bandwidth values and modified them to run them in an infinite loop so that we can test the impact of other applications in these metrics and understand to what extent the QoS features provided by Infiniband can be useful to provide performance guarantees to time-sensitive applications.

Figure 1 shows the experimental setup we have used to test Infiniband QoS features. This setup consists of two Intel manycore processors (that are part of the RECIPE prototype) connected through an Infiniband host-card adapter. In this hardware setup we deploy one or several instances of MPI kernel applications. In particular, we have executed two different kernel applications: (a) one to measure the maximum bandwidth by reading (or writing) data from one processor to the other, and (b) another application to test the latency of read and write transfers from one processor to the other. It is important to note that we use kernel applications to stress the throughput because real applications executed in this particular setup will not stress Infiniband link (one link is interconnecting just one processor). In a more realistic setup in which

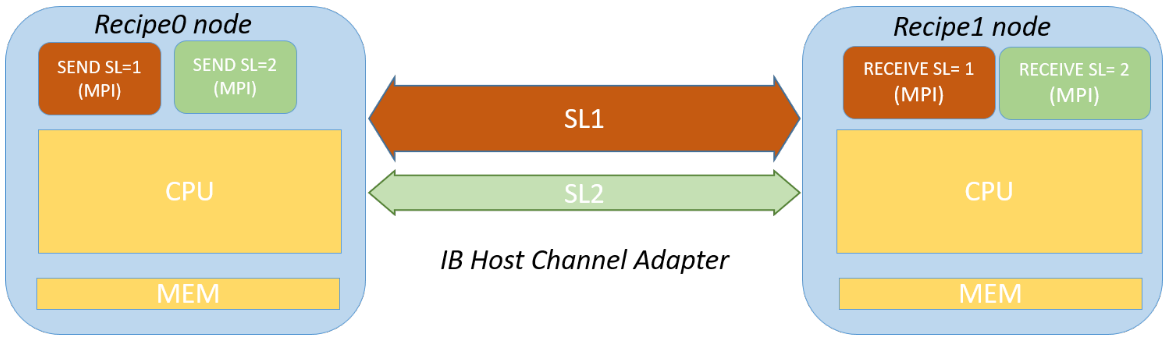

Figure 1: QoS experimental setup

a certain Infiniband link has to serve communications to a very high number of cores we expect regular applications to stress bandwidth requirements in a similar manner.

To achieve the best possible performance guarantees that applications with strict timing requirements may require we have configured OpenSM system with the following parameters:

```
qos_ca_max_vls 15
qos_ca_high_limit 255
qos_ca_vlarb_high 0:255
qos_ca_vlarb_low 0:0,1:64,2:191,3:0,4:0,5:0,6:0,7:0
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

As shown in the text above we assign to SL 0 the highest priority. In fact, as explained before, when the high limit is set to 255 there is no limitation impose in the amount of packets that SL 0 can prioritized. This means that applications executed using that service level will suffer very limited interference. For the rest of the service levels we assign a lower priority but we perform also an uneven distribution of bandwidth as reflected in the different weight values assigned to each VL. For instance, we assign 64 credits to VL 1 and 128 credits to VL 2 in the low priority level.

To launch applications assigned to a specific service level we use the following command:

```
mpiexec -n 2 -H recipe0,recipe1 --mca pml ucx -x UCX_NET_DEVICES=mlx5_0:1
        -x UCX_IB_SL=0 ./test_bw
```

With the command above we are executing the test_bw application with a service level equal to 0 and using the mlx5_0:1 Infiniband interface connecting recipe0 and recipe1 servers.

Table 2.3 shows bandwidth results achieved with the test applications when sending 4MB packets (i.e packets of large size). As shown in the table when only one test bandwidth application is executed the measured bandwidth is equal to 6.638GB/s in average and shows little variability. When two test applications are executed the test application executed with the highest priority SL (i.e SL=0) gets roughly the same bandwidth since all the packets of this application are prioritized. On the contrary, the same test application executed with SL 1 gets lower bandwidth since arbitration always prioritizes SL 0 packets. In particular, this application gets 3.163GB/s that correspond to a 47% of the maximum bandwidth requirements of the test application. Note that this reduction can be more significant in the context of high priority applications with higher bandwidth requirements. When three test applications are executed we observe that the maximum peak bandwidth is not guaranteed and the maximum peak bandwidth of the application is reduced to a 93% of the peak bandwidth required by the application. Note that this is explained by the fact that in a high contention scenario, in which bandwidth requirements are high, network links get congested and cause back-pressure effects (queuing delay) that cannot be removed by prioritizing packets at the arbitration. The other two application get even less bandwidth as they have no priority over SL 0. The performance they achieve comes from the fact that packets of these flows can get into the arbitration opportunistically, meanwhile packets of other higher priority flows are stalled at the end-node waiting the memory and CPU to serve their request.

Table 1: Bandwidth tests for different scenarios

	Bandwidth (MB/s)		
Scenario	SL=0	SL=1	SL=2
Interference free	6638	-	-
2 apps	6634	3163	-
3 apps	6223	2175	2167

Table 2.3 shows latency results for a test application that sends packets of 4MB. When the test application is executed alone the latency required to get the 4MB of data is 641us. When another application is co-running with the test application using the highest priority service level (SL=0), the latency of the high priority application is slightly increased (1%) while the latency of the application using SL 1 is increased by a factor of more than 20×. When three test applications are executed in the RECIPE cluster we observe an interesting effect that shows the limitations of guaranteeing performance just at the arbitration level. We observe that the latency of the highest priority application increases 1.45× and at the same time the latency of the applications using the other service levels is higher than the one experienced in the interference free scenario but lower than the latency values the application at SL 1 was having in the 2 apps scenario. While further experiments would be required to fully understand this behaviour our initial hypothesis is that this is explained by the fact that when the links get more congested packets of the high-priority flow can get stalled due to queuing delay. When packets of the high priority flow finally reach the destination node the low priority ones are granted access by the arbiter while this request is being served and at the same time create back-pressure effect at the destination node so increasing the queuing delay of the high-priority ones. That is performance guarantees cannot be fully preserved in a high congestion scenario.

Table 2: Latency tests for different scenarios

	Latency (us)		
Scenario	SL=0	SL=1	SL=2
Interference free	640.92	-	-
2 apps	649	13301	-
3 apps	943	3271	3385

2.4 Performance Isolation Limitations

While playing with the arbitration weights as Infiniband is very useful to achieve performance guarantees there are certain limitations in what we can achieve that must be considered. In particular we have found the two following aspects that can affect the performance guarantess provided by Infiniband:

- Low priority packets that arrived Infiniband card adapters and switches before high-priority packets and have been already arbitrated can create contention due to back-pressure effects to high priority flows. However, this impact is limited if the other applications do not saturate the network. We expect to be able to use the global and local resource managers to prevent that co-running applications saturate the network thus, allowing performance guarantees be respected during the execution of the time critical application.
- The state in which the network is found at the time a time-critical application arrives impacts performance guarantees and the initial stage since until in-flight packets are not absorbed they can create contention due to back pressure effects. However, we consider the impact of this effect is very limited (just hundred of milliseconds are required to get stable performance guarantees) provided HPC application timing requirements do not need to complete within such small time window.

3 Checkpointing at the FPGA Level

Because of the importance of Checkpoint/Restart in traditional HPC, RECIPE sets as one of its objectives exploring the extension of the C/R approach to Field Programmable Gate Array designs, providing all the key benefits of conventional C/R [2] to new classes of HPC applications inherently relying on deeply heterogeneous acceleration. A few crucial requirements were identified for such FPGA-augmented C/R techniques:

- transparency/automation;
- limited overhead;
- selectivity;
- portability.

Checkpoint/Restart techniques are normally classified in application-level, user-level, and system-level solutions [1], the latter being operated by the system hosting the job in a transparent way and in turn divided in solutions relying on Operating System support and hardware-level support. In line with the above key requirements, we based our FPGA-augmented C/R technique on a system-level approach, inherently needing a combined support from the underlying hardware and OS/RTMS. As implied by the above motivation and objectives, the first issue to address for extending C/R techniques to FPGA designs is the definition of the acceleration kernel state. From a physical perspective, considering the typical architecture of an FPGA acceleration card used in HPC, the overall device state is held by on-chip flip-flops; on-chip RAM, e.g. Xilinx BlockRAM components; and off-chip RAM modules, including standard DDR as well as new High-Bandwidth Memory technologies. While a coarse-grain approach to FPGA Checkpoint/Restart would retrieve and restore of the whole device state, this would incur large overheads, because not all the on-chip stateful resources are actually part of the application state and not the whole memory space is used by a given application. Furthermore, some of the application resources/memory areas could be used for holding temporary results that can be derived from the closest preceding checkpoint. In RECIPE we thus aim for a general mechanism allowing the designer to define what we named permanent state, indicating selectively what subset of the overall physical state is of actual relevance for checkpointing, involving both the FPGA device and the off-chip memory. Note that in this deliverable we are concerned with low-level enabling mechanisms, while future developments beyond RECIPE could cover tool-supported optimal identification of permanent state. In particular, in the implementation demonstrated in RECIPE, the expression of the state of an FPGA kernel is based on

- (off-chip state) the explicit indication of memory address windows that are designed to contain the permanent state (possibly, the whole allocated memory space, if no transient state can be identified).
- (on-chip state) the instantiation in the FPGA design of special library components, provided by RECIPE, featuring stateful behavior (registers, configurable memory modules, etc.). The explicit use of such components tells the RECIPE flow that the information held by the corresponding instantiated modules is part of the permanent state.

3.1 Internal configuration port and debug port

One crucial issue for the actual implementation of the proposed approach is the availability of low-level access mechanisms enabling all the components of the permanent state to be read/restored. The selected mechanism should enable the key properties of transparency/automation highlighted above as well as ensure limited overheads especially in terms of additional resources required and access times. In RECIPE we considered two alternatives providing complementary opportunities: internal reconfiguration ports (e.g. Xilinx ICAP) and JTAG debug port.

Internal ports, such as Xilinx *Internal Configuration Access Port* (ICAP), provide the ability of reconfiguring the device from the design itself, without any intervention from the outside. The key idea enabled by internal reconfiguration ports is to instrument automatically the user's FPGA design with additional logic in charge of controlling the Checkpoint/Restart process. This entails taking a snapshot of the overall design state through the corresponding commands provided by the internal reconfiguration port, reading back the contents of the FPGA configuration memory, which includes user elements such as flip-flops and on-chip block memory, accessing a DDR/HBM controller to read data from the off-chip memory, extracting the data subset corresponding to the permanent state, and writing it to an off-chip nonvolatile memory (NVM). The Restart process involves symmetric steps and requires freezing the design while the permanent data are retrieved from the NVM and expanded in a form that can be fed to the internal reconfiguration port. Crucial to this phase is the availability of a mode of operation enabling the successive load of configuration data to the reconfiguration port followed by a single, atomic command enforcing the new state. The devices used for experiments in the RECIPE prototype, namely a high-end Xilinx Ultrascale+ FPGA device mounted on an Alveo U280 HPC card, provide such a mode of operation with their Internal Configuration Access Port (ICAP). Of course, the additional C/R hardware used to instrument the user design must include the logic to address and decode the configuration *frames* that are retrieved from the configuration port as well as to extract/update the permanent state bits, which might be scattered throughout the frames with irregular patterns. The additional application-specific information required by the C/R hardware to locate the state bits in the configuration frames can in turn be stored in the NVM and retrieved by the hardware at each Checkpoint and each Restart operation. The C/R hardware also needs access to the off-chip memory, meaning that it needs to be connected as a master to the memory controller, concurrently with the main user design, while the information on the memory windows to be saved and restored is supposed to be stored in the NVM along with the on-chip state information. The saved state held in the NVM can be separately accessed by the host, both because the FPGA contents are part of a larger notion of application state, most likely involving the standard software part handled with usual C/R techniques, and in order to save the state on a persistent storage support besides the NVM.

The other access mechanism considered for use in RECIPE is the **JTAG debug port**. FPGAs normally provide a JTAG port for uploading the configuration image to the device memory as well as retrieving the memory contents from the device, in a similar way to the internal configuration port. However, unlike the internal port, JTAG must be operated from the outside and relies on a built-in controller, meaning that it does *not* require additional user logic resources. The port uses a serial interface and a dedicated connection, in most current board accessible through a USB port from a host system (possibly the server hosting the acceleration card itself, in case of a PCIe board). Hence, while the logic data exchanged through the JTAG port is essentially the same as with the internal port (in terms of configuration commands and memory frames), they are directly handled by the host-side C/R software and do not require any special infrastructure on the board, including the hardware-side C/R logic and the NVM. For accessing the off-chip memory, however, one must either require that the external memory device itself has a JTAG port accessible on the same scan chain or, as a more general solution, make the FPGA-side memory controller accessible from the JTAG logic and, hence, from the host-side software. In Xilinx devices, for example, this is possible by means of a special JTAG-to-AXI master component, which essentially allows JTAG to control an additional AXI master in the FPGA concurring with the user design for access to the memory controller. Therefore, based on this infrastructure, the JTAG access mechanism in the RECIPE assumes that all the C/R logic, including the location of permanent state bits in the configuration memory as well as the identification of off-chip memory address windows to be saved, is under the responsibility of the host-side software, while the FPGA user design is essentially unaware of it, except possibly for the additional JTAG-controlled master to be added to the internal bus for off-chip memory access.

3.2 Overall flow

In this section we outline the overall flow involving design-time and run-time operations. We refer the description to Xilinx technologies and software tools [5], since a few aspects of the flow are inherently specific to the particular FPGA family of choice. As already highlighted above, the initial step of the C/R methodology consists of specifying the application permanent state, involving both the on-chip state (registers and memory blocks) and the off-chip state (external memory modules). The specification of the on-chip state of relevance for checkpointing operation requires that the user explicitly instantiates in their design suitable components from a dedicated library. In the current implementation, the library provides parameterized register and memory blocks. The library implementation (opaque to the user) relies on direct instantiation of Xilinx primitives, i.e. flip-flops for register components and BRAMs for memory components, with a pre-defined naming convention, so that exact information can be kept about the physical implementation of those components throughout the design cycle. These special component names are looked up after synthesis and place-and-route in order to identify the user-instantiated modules. In fact, the names are kept by Xilinx tools in the form of design component properties, that can be queried through particular Tcl commands once the post-implementation design is generated. In the RECIPE prototype flow, this step is automated by means of Tcl scripts used to search for all instantiated components and, for each of them, retrieve the specific coordinates in the device physical layout. Once those coordinates are available, the next step is to locate the frame addresses and offsets in the FPGA configuration memory corresponding to the given coordinates. Again, this association is based on a by-product of the Xilinx toolflow, the logic allocation file (.ll), which keeps track of the correspondence between the physical primitives in the device layout and the frame addresses/offsets in the configuration memory. A second script was therefore developed to automate the inspection of the logic allocation file. The whole process yields a precise mapping of the original user-instantiated modules holding the permanent state and the locations in the configuration image that is readback/written through either the ICAP or the JTAG port. For external memory modules, the address windows to be checkpointed are listed in a separate input file, that is then read by the C/R software for run-time operation. Note that, apart from instantiating specific components from the C/R library and explicitly indicating address windows for external memory modules, the user is essentially unaware of the above flow. In the current prototype, the scripts are operated manually, but they could be easily automated and hidden in the overall toolchain, in case the C/R support would be fully integrated in the FPGA development tools. The design-time flow described above is summarized in Figure 2.

Regarding the run-time operation, the current prototype relies on the JTAG access mechanism, as explained above. The JTAG interface and the related readback/write commands can be accessed through a proprietary driver and Tcl command console provided by Xilinx, so the concept is currently demonstrated through Tcl scripts that reproduce the behavior of a C/R software layer, thus incorporating the low-level logic used to read in the permanent state information (configuration memory location and external memory address windows), the FPGA scan-chain operation and off-chip memory access through the JTAG-to-AXI component, the access to a persistent host-side storage where application state snapshots are saved, and any possible higher level logic, e.g. the RTMS controlling the C/R on the whole application- or system-level. As a final remark, while the operations required by the above flow are currently being carried out through the Xilinx Tcl console, we are developing a C/C++ wrapper, allowing the same operations to be performed programmatically for demonstration purposes in RECIPE. Again, the proof-of-concept could of course be easily re-engineered by the tool providers, in case the approach should be incorporated in a commercial FPGA toolchain.

4 Integration Plan

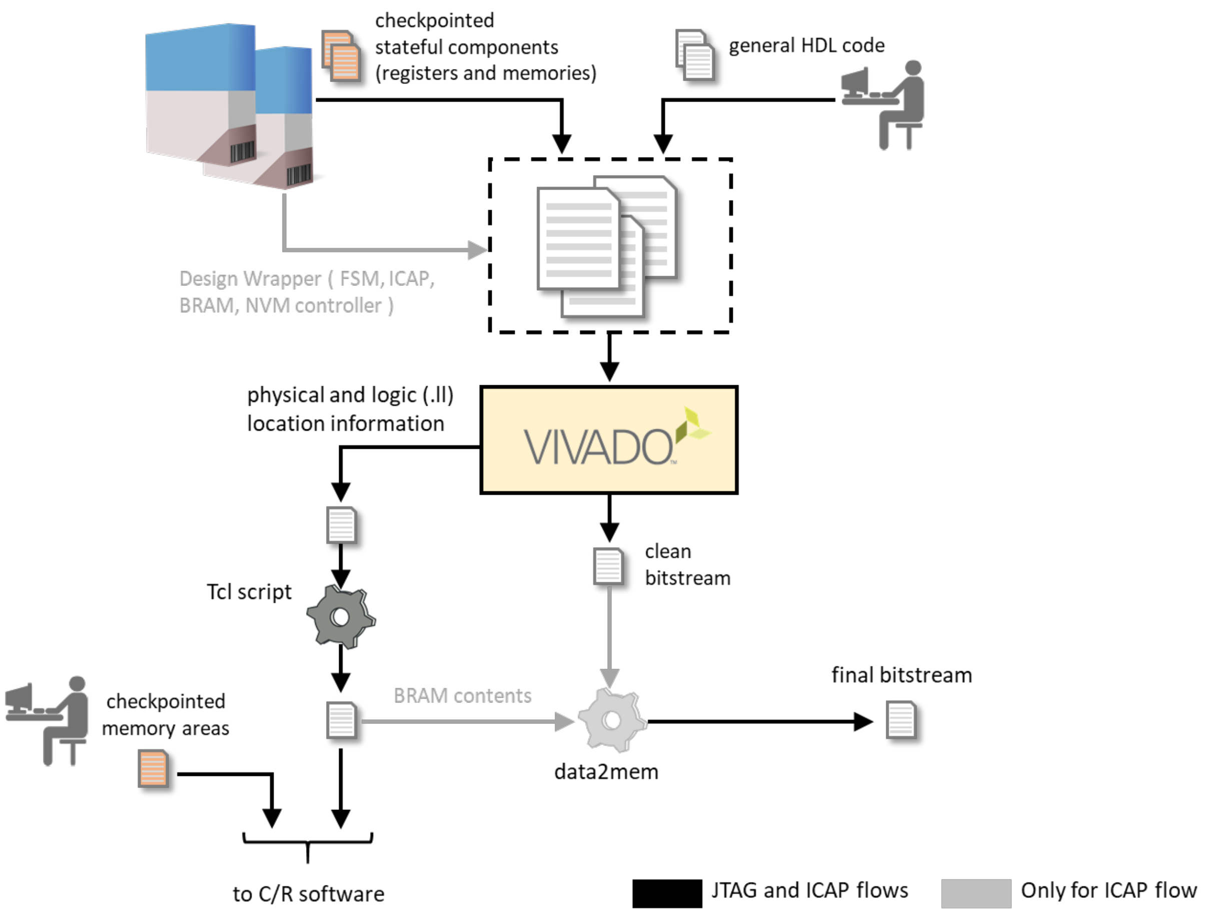

Figure 2: Overview of the overall FPGA C/R flow. The ICAP alternative would require an additional branch in the flow (grey lines), instrumenting the user design with additional C/R logic and initializing the on-chip BRAMs used by the C/R logic with some design-specific information.

This deliverable describes the QoS and check-pointing techniques that have been tested in controlled and limited scenarios. The next step will be integrating these features in the whole RECIPE stack so that use-cases and/or the resource manager are able to employ these features. To allow this the plan is to interface these techniques with the software stack developed in WP2 from M18 to M24 and to test its functioning and correct potential side-effects between M24 and M30. Finally, both techniques should be exercised and validated by the use-case applications between M30 and M36.

5 Conclusions

In this deliverable we have presented how to leverage Infiniand QoS features and how to deploy a checkpoiting system usable in the context of HPC applications.

In the context of QoS we have shown that Infiniband features are suitable to ensure performance guarantees to time-critical applications. We have also discussed about the potential limitations of these features and analysed that this limitations can be sorted out in the context of the time-critical applications targeted in RECIPE.

With respect to checkpointing in FPGA systems we have show how to use Vivado tools to provide the necessary support for checkpoiting. We have also analysed current limitations of ALVEO boards and proposed alternative paths to solve this potential limitations. Overall, we consider the proposed prototype solution can be deployed in a more realistic scenario allowing for the first time suppor for check-pointing in HPC applications using FPGA devices.

References

[1] Ifeanyi P. Egwutuoha, David Levy, Bran Selic, and Shiping Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65:1302–1326, 2013.

[2] I. Koren and C.M. Krishna. *Fault-Tolerant Systems*. Elsevier Science, 2010.

[3] Mellanox. Mellanox OFED for Linux User Manual Rev 1.5.3-1.0.0, 2011. http://www.mellanox.com/related-docs/prod_software/Mellanox%20OFED%20Linux%20User%20Manual%201_5_3-1_0_0.pdf.

[4] T. Shanley, J. Winkles, and Inc MindShare. *InfiniBand Network Architecture*. PC system architecture series. Addison-Wesley, 2003.

[5] Xilinx. Vivado Design tools, Accessed October 2019. https://www.xilinx.com/products/design-tools/vivado.html.