Trading-off reliability and performance in FPGA-based reconfigurable heterogeneous systems

Alessandro Vallero, Alberto Carelli and Stefano Di Carlo
Politecnico di Torino, Control and Computing Departments,
{alessandro.vallero | alberto.carelli | stefano.dicarlo}@polito.it

***Abstract*—Recent years have witnessed the rapid growth of heterogeneous systems, composed of CPUs and hardware accelerators, to face up the constant increase of computational performance demand of digital systems. In this scenario, FPGAs offer the possibility to implement high performance reconfigurable accelerators, able to speed up the intrinsically parallel portions of applications. The study of reconfigurable heterogeneous systems is still maturing and, while some contributions about performance and power consumption are available, in literature there are few works addressing reliability. This paper analyzes reconfigurable heterogeneous systems in presence of permanent faults occurring in the FPGA. In this context, a reconfigurable heterogeneous architecture, including a Run Time Manager responsible for the communication of software tasks and the FPGA, the scheduling and the placement of hardware tasks, is presented. In addition, the paper introduces a reconfigurable heterogeneous system simulator for the proposed architecture. This simulator is able to evaluate during the design phase the degradation of the system performance due to permanent faults and allows to explore the design space dimensions efficiently.**

I. Introduction

In recent years, the performance demand and complexity of applications have constantly increased alongside with the development of new technologies. As a result, the computational paradigm is moving from CPU based computing platforms to heterogeneous systems composed of CPUs and several classes of dedicated hardware accelerators. In this context, this paper addresses heterogeneous systems composed of a CPU and a Field Programmable Gate Array (FPGA). FPGAs are among the most promising hardware accelerators [1], [2]. This trend is confirmed by the recent launch of the Intel® Programmable Acceleration Cards enabling to build FPGA heterogeneous systems based on the the Intel Xeon CPU and the Intel Arria® 10 GX FPGA [3]. FPGAs enable to dynamically reconfigure the same hardware to perform different tasks. The introduction of Dynamic Partial Reconfiguration (DPR) capabilities has further increased this trend. DPR allows for the configuration of a portion of the FPGA at run-time to implement a given functionality. This happens without interrupting the tasks running on the remaining parts of the accelerator, thus achieving high flexibility and responsiveness.

As in all deep sub-micron technologies, permanent faults due to fabrication defects and/or aging of the device are a significant concern in FPGA-based heterogeneous systems. They may significantly impact the reliability and performance of the final application. In literature, the effect of permanent faults in standalone FPGA systems for mission critical applications has been previously analyzed. This includes techniques to detect permanent faults [4], [5] and techniques to tolerate or recover from the effect of these faults [6], [7], [8], [9], [10]. In [6], [7], the authors present a detection and recovery methodology for permanent faults in FPGA systems. Whenever a permanent fault is detected, the fault is recovered by reconfiguring the whole FPGA with a precomputed configuration, thus excluding the faulty area. This approach is however not feasible in case of a large number of hardware tasks, since the number of possible precomputed configurations grows exponentially. A good strategy consists of dividing the reconfigurable area into tiles as proposed in [8]. Recovery from a permanent fault is achieved by moving the configuration of a faulty tile into a spare one. In this way, the number of configurations becomes proportional to the number of hardware tasks at the cost of a more complex reconfiguration hardware architecture.

Reconfigurable architectures and reconfiguration strategies have already been addressed in literature [11]. The most important limitation of these reconfigurable architectures is that they are designed for standalone FPGA systems. In these systems, the hardware functionality configured in the FPGA is constant and does not change according to the run-time needs of the application. General purpose reconfigurable heterogeneous systems including both FPGAs and CPUs introduce a higher level of complexity. The execution of the tasks must be properly orchestrated between the CPUs and the hardware accelerators implemented in the FPGAs. This is required to exploit all the computational resources efficiently, thus reaching the best possible performance. To achieve this goal, Burns et al. [12] introduced the concept of Run-Time Manager (RTM), a component able to coordinate and manage a reconfigurable hardware architecture, thus enabling self-awareness and self-adaptivity. The RTM is usually part of an Operating System (OS). It is responsible for the scheduling and the placement of hardware tasks in the underlying reconfigurable hardware architecture. The RTM monitors the load of the FPGA and keeps track of the parts of the FPGA that are involved in the computation at a given time. On the basis of these parameters, it decides the execution order of the hardware tasks and the portion of reconfigurable area where they must be accommodated. An RTM able to schedule hardware tasks for 2D reconfigurable areas, taking into account a reuse approach, was proposed in [13]. However, the impact of the occurrence of permanent faults is not considered. An optimized RTM for heterogeneous systems was presented in [14], [15]. However, these works assume that the execution order of the hardware tasks is known in advance, thus limiting the application of this methodology in case of general purpose applications.

This paper analyzes FPGA based reconfigurable heterogeneous systems in presence of permanent faults occurring in the FPGA. For this purpose we introduce a reconfigurable heterogeneous architecture able to tolerate permanent faults, in which software applications can issue requests for acceleration via reconfigurable hardware to an RTM. In addition, the paper introduces a reconfigurable heterogeneous system simulator for

the proposed architecture. This simulator is able to evaluate during the design phase the degradation of the system performance due to permanent faults. It therefore allows us to explore the design space dimensions efficiently.

The reminder of the paper is organized as follows: Section II describes the proposed reconfigurable heterogeneous system framework while Section III shows the experimental results analyzing the performance degradation due to permanent faults for different configurations of the proposed system architecture. Finally Section IV concludes the paper.

II. The reconfigurable heterogeneous system framework

The proposed FPGA-based reconfigurable heterogeneous system framework is designed to allow hardware acceleration, while taking into account the occurrence of permanent faults occurring in the reconfigurable hardware. The framework is designed to operate in a context in which the order and the starting time of applications is not predictable. It allows to accelerate application tasks by executing them on the reconfigurable hardware. In details, every time a task is mapped to the reconfigurable hardware a new configuration of a portion of the FPGA is issued. Multiple tasks can be accommodated concurrently into the FPGA, depending on the area they require. By exploiting DPR, it is possible to configure the FPGA without stopping the execution of the tasks not involved in the configuration process. Whenever hardware faults are detected, the hardware is reconfigured in order to avoid the use of the faulty portion of the FPGA.

A. System architecture

At the software layer, two main actors play a role into the proposed system architecture: (i) the software applications that require hardware resources to be executed, and (ii) the RTM that is in charge of orchestrating the execution of the applications. These two players are organized into a client/server architecture as depicted in Figure 1. The RTM acts as a server able to receive requests for computing resources from the applications. Every time a software task requires to be accelerated by reconfigurable hardware, the application sends a request to the RTM. When the task is completed, the server informs the application that the output of the accelerated computation is available.

To support the high-level architecture presented in Figure 1, the reconfigurable hardware available in the heterogeneous systems must be properly integrated with the CPU executing the software applications. Figure 2 illustrates the proposed architecture that partitions the FPGA in two parts: the static area depicted in green, which is never modified, and the reconfigurable area depicted in orange. The reconfigurable area is partitioned into several reconfigurable tiles, named reconfigurable slots. The reconfigurable slots can be configured to accomodate the soft-cores required to accelerate the computation. A single soft-core can occupy more than one reconfigurable slot, depending on its size. The static area hosts instead three main blocks. The communication manager is responsible for moving data among reconfigurable slots and to the external world (main memory). The configuration manager is in charge of configuring the FPGA, based on the commands

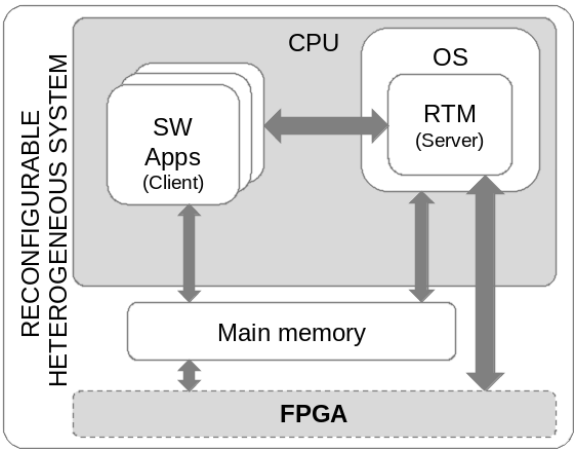

Fig. 1. Software-layer view of the proposed heterogeneous architecture.

issued by the RTM. When requested, it moves the proper configuration files (*bitstreams*), from a dedicated configuration memory into the internal configuration port of the FPGA. A dedicated configuration memory is adopted to achieve better performance and to avoid congestion of interconnections. Finally, the fault manager monitors permanent faults occurring in the reconfigurable area of the FPGA, transmitting information about faulty slots to the RTM that can take proper actions to reconfigure the FPGA. This task can be accomplished by running periodic tests on the FPGA [4], [5], or simply collecting error signals generated by fault detection hardware embedded in the soft-cores mapped in the FPGA [10].

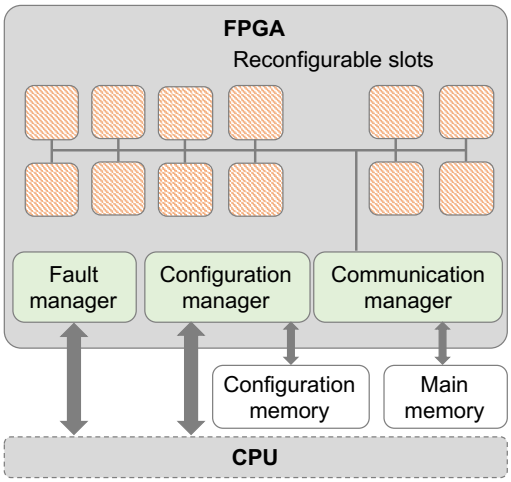

Fig. 2. The target FPGA architecture

Based on this hardware architecture, the main modules composing the RTM can be summarized as follows:

- *Requests manager*: it accepts and decodes requests for hardware computing resources. Once the requests are accepted, they are communicated to the scheduler. Finally, when a request is served, it notifies the requesting software applications that the associated task is completed.
- *Scheduler*: it manages the assignment of the FPGA reconfigurable slots to the requests issued by the

software applications. This operation is performed according to a particular scheduling policy.

- *Hardware configuration dispatcher*: it provides to the FPGA configuration manager the information required to configure and execute hardware tasks. This includes: information about the involved reconfigurable slots, the bitstreams to be configured and the address space in main memory associated with the software application are communicated.
- *Committer*: it receives notifications from the FPGA configuration manager about hardware tasks completion and results.
- *Fault manager*: it tracks the list of faulty slots mapping computations to fault-free slots.

The scheduler decides where and when a hardware task is executed. Each hardware task requires to configure and use a specific set of soft-cores. In the proposed architecture, several implementations of the same soft-core are considered. In particular, each implementation is characterized according to the configurable slots that are involved and must be stored in the configuration memory. When a task is scheduled, one of the available implementations is chosen and its associated bitstreams are configured. The RTM has a database of all bitstreams available for each soft-core. This feature allows the RTM to access directly to the information of soft-cores as soon as a new request arrives from a software applications. For each bitstream, several properties are available including: the configuration memory address of the bitstream, the involved FPGA reconfigurable slots, an estimate of the configuration time and of the execution time. The scheduler selects one of the available configurations for each soft-core on the basis of these parameters.

To achieve high performance, the scheduler keeps track of the current status of the system. In particular, each reconfigurable slot is monitored in order to know which soft-core and its associated implementation are currently configured. The status of the execution of each reconfigurable slot is taken into account as well. The status can be executing, when computation is running, cached, when the task is finished and the soft-core is ready to begin a new task, or faulty, when a permanent fault has occurred in the reconfigurable slot. This information is particularly important since when a slot is cached, it can be employed again without the need of the configuration process. This saves time during the configuration process.

The life-cycle of a software request consists of several steps. During the steps, the request is moved into different lists depending on its status. When the request is received by the RTM, it checks if the associated hardware task can be executed on the reconfigurable hardware. In case there is no bitstream available for the required soft-core, or all the bitstreams associated with the requested soft-core involve faulty reconfigurable slots, the software request is refused. Otherwise, it is pushed into a waiting list. The waiting list is a list ordered according to the arrival time of the requests. It contains all the requests that have already been accepted but that have not been sent by the dispatcher yet. Every time a new request arrives or a request completes, the scheduler checks if a new request can be dispatched. When a request is processed by the dispatcher, it is then moved to the executing list. When the committer is notified that the computation related to a request finishes, the request is moved to the done list. Eventually, the RTM sends a message to the application to notify that the hardware task execution is complete.

From the reliability standpoint, the static area represents a critical element of the proposed architecture. Fault tolerant techniques such as Triple Modular Redundancy (TMR) must be implemented to protect this area.

B. System simulator

To analyze the performance of the proposed reconfigurable heterogeneous system architecture for a given set of applications, a system simulator has been developed. Using the simulator, the system can be stressed by profiled and synthetic software requests. For synthetic software requests several parameters can be chosen: the number of requests sent to the RTM; the time period between two requests; the number of implemented soft-cores; the number of possible configurations per soft-core; the number of reconfigurable slots per each configuration of every soft-core; the execution time per each configuration of every soft-core; and the number of faulty slots.

All parameters can be fixed or distributed according to a uniform or a Gaussian probability density function. For the reconfigurable architecture, the number of reconfigurable slots and the reconfiguration time of each slot can be configured.

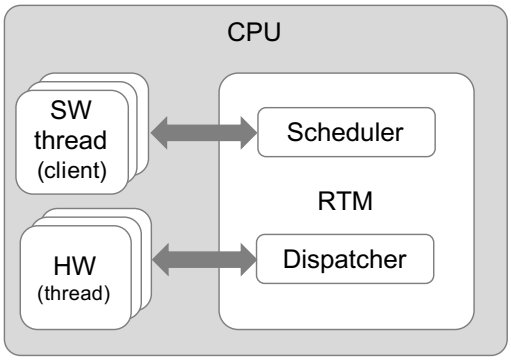

Fig. 3. The proposed system simulator

The simulator runs on the CPU only (Figure 3). Operations on the FPGA are emulated via software. More specifically, the reconfiguration process managed by the dispatcher consists of making the dispatcher sleep for a time equal to the one required by the configuration manager of the FPGA to load the proper bitstream. Instead, the execution of the hardware tasks is implemented by the creation of a new software thread executing a sleep for an amount of time equal to its execution time. As soon as an emulated hardware task is over, it notifies its end to the committer.

To evaluate the performance of the system, the simulator is able to provide information about the percentage of refused/accepted client requests, the *load* of the FPGA, that is the average ratio between the number of reconfigurable slots involved in the requested computation and the total number of

reconfigurable slots in a unit of time, the *turnaround time*, that is the global time to complete a client request, the *waiting time*, that is the time a client request spends waiting before being executed, including the time needed for its configuration, and the *scheduling time*, that is the time employed by the scheduler. Time measurement have an uncertainty of 1 micro second.

III. Experimental Results

The proposed Run-Time Manager is implemented as a multi-threaded user application in a Linux environment. We tested the performance of the proposed reconfigurable heterogeneous system framework by means of the simulator presented in section II under different scenarios. The performed analysis focuses on the impact of the effect of permanent faults on the performance of the system. In order to reproduce a real-world scenario, we run the simulation on top of the Xilinx ZYNQ™-7020 SoC. This SoC is composed of a dual-core ARM Cortex-A9 MPCore microprocessor whose maximum clock frequency is 667 MHz. The simulator was configured reflecting the parameters of the FPGA of this SoC.

For each experiment, we generated 100 synthetic client requests separated by a time interval of 1 ms. We considered a target FPGA composed of 100 reconfigurable slots. The reconfiguration time required by a single slot was set equal to 3*ms*. We hypothesized 100 different soft-cores available to satisfy client requests. Each soft-core implementation was characterized by a number of reconfigurable slots according to a Gaussian distribution with mean and standard deviation respectively equal to 8 and 2. In this way we are able to model the area required by every soft-core. The execution time of each soft-core was assumed as uniformly distributed between 3 and 40 *ms*.

To evaluate the trade-off between reliability and performance, we run several simulations changing the number of faulty slots (i.e., slots affected by at least a permanent fault) from 0 to 50 and the number of available configurations per soft-core from 1 to 100. For each combination of number of faulty slots and number of available configurations per soft-core, we repeated the simulation 10 times for statistical significance. We evaluated the average value of: the load of the FPGA, the turnaround time, the waiting time and the number of accepted client requests. Statistics about rejected client requests are omitted from these measurement. Results are shown in the form of heatmaps, where the x axis is the number of available configurations per soft-core and the y axis is the number of faulty tiles. In the reminder of this section, the points of each heatmap are referred through their coordinates (x,y).

A. The load

Results reporting the load of the reconfigurable slots are presented in Figure 4. The first thing that can be noticed is that there is not a linear relation among the load, the number of faulty slots and the number of available configurations. In fact, there is a yellow line, indicating high loads, ranging from $(20,0)$ to $(81,13)$.

To better understand the impact of the number of faulty slots and the number of available configurations per soft-core we can analyze them independently. Figure 5 reports the load

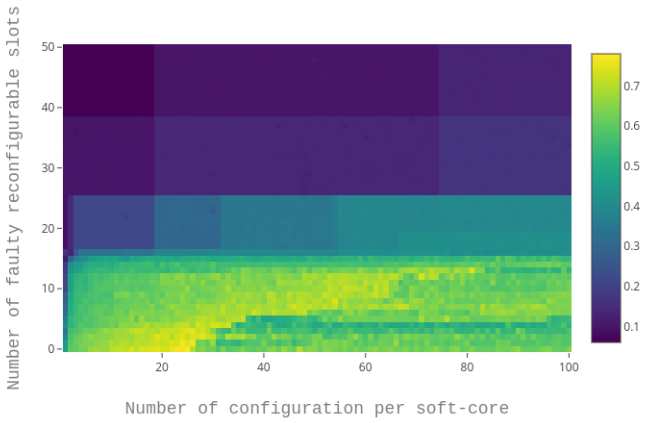

Fig. 4. The load of the reconfigurable slots.

of each row normalized with respect to the case in which there is only one configuration available per soft-core (first column). This plot highlights how the load of the FPGA improves increasing the number of available configurations (moving horizontally from left to right). However, this is not always true. Below 15 faulty slots, the highest load (for a fixed number of faulty slots) is in between 20 and 80 number of configurations. Moreover, in this particular region, the load increases with the number of faulty slots (moving vertically). This behavior is the result of the used scheduling policy, which is not able to fully load the FPGA even if there is a large number of configurations available. Further explanations about this hypothesis are provided in the following subsection.

Figure 5 also reveals a dramatic break down close to 26 faulty slots. This means that the improvements due to higher number of configurations in terms of load are not relevant above this threshold since the load is kept very low (Figure 4). This break down is also evident in Figure 6, where the load of each column is normalized with respect to the load of the case in which there are no faults (first row).

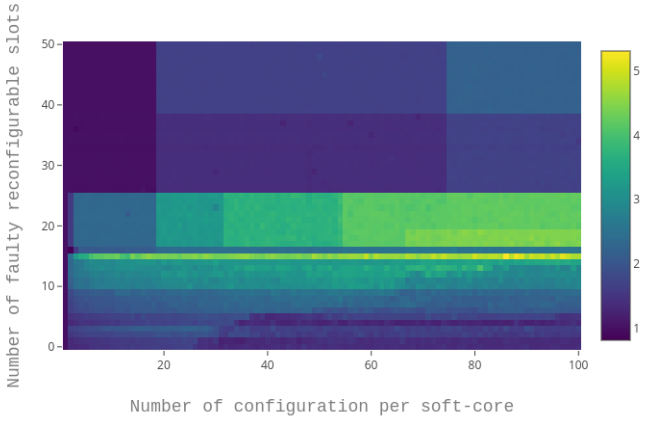

Fig. 5. The load of the reconfigurable slots with results normalized with respect to the case there is only one configuration available per soft-core.

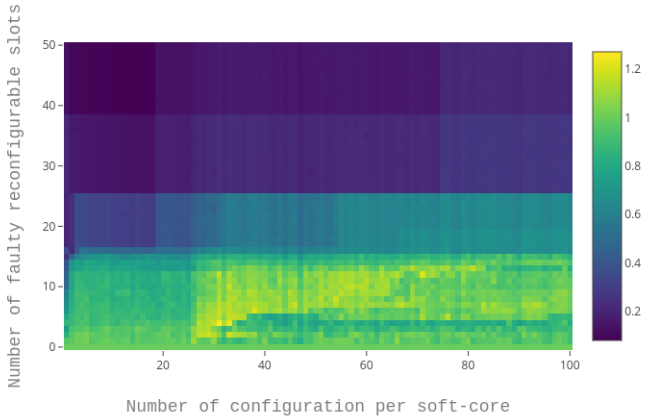

Fig. 6. The load of the reconfigurable slots with results normalized with respect to the case there are not any faulty reconfigurable slots.

Figure 6 reveals that, in most of the cases, the load decreases as the number of faulty slots increases (moving vertically), apart from the rectangular region described by the two points with coordinates *(26,1)* and *(82,13)*. This anomaly is caused by the adopted scheduling policy as well. Finally, looking at the heatmap horizontally, from left to right, it can be noticed that the degradation due to the increase of the number of faulty slots in terms of load is similar regardless the number of configurations available per soft-core, with the exception of the case in which only a configuration is available.

B. Timing performance

When evaluating the timing performance of a reconfigurable heterogeneous system, we are mainly concerned by three parameters: the *turnaround time*, the *waiting time* and the *scheduling time* of the client requests. We found that scheduling time is negligible with respect to the other two timing metrics, regardless the number of available configurations and the number of faulty slots. For this reason, results concerning scheduling time are omitted.

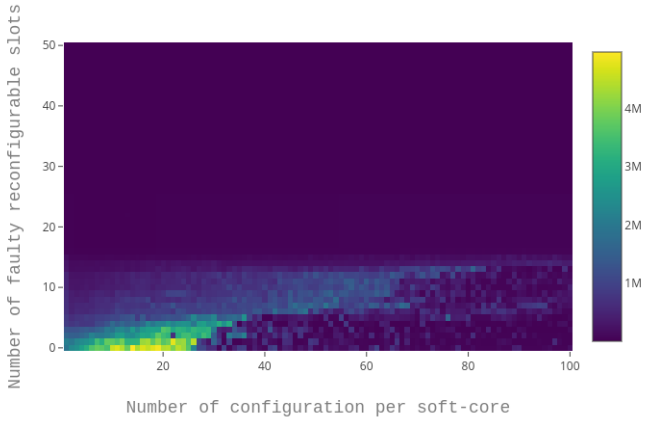

Fig. 7. The average turnaround time (*ms*).

The average turnaround time of client requests for each experiment is reported in the heatmap of Figure 7. The region with the highest turnaround time is also the region with the highest load (Figure 4). This gives more strength to the hypothesis that the scheduling policy is responsible for such a strange behavior. In fact, in this specific case, the order in which the hardware tasks are scheduled depends on two factors: the available reconfigurable slots and the available configurations for the soft-cores. We believe that for experiments with high load and high turnaround time, large and long tasks are executed before small and short ones, thus resulting in high occupancy of the FPGA and high waiting time for small and short client requests. In fact, when analyzing the average waiting time normalized with respect to the average execution time (Figure 8), these experiments are characterized by high waiting time.

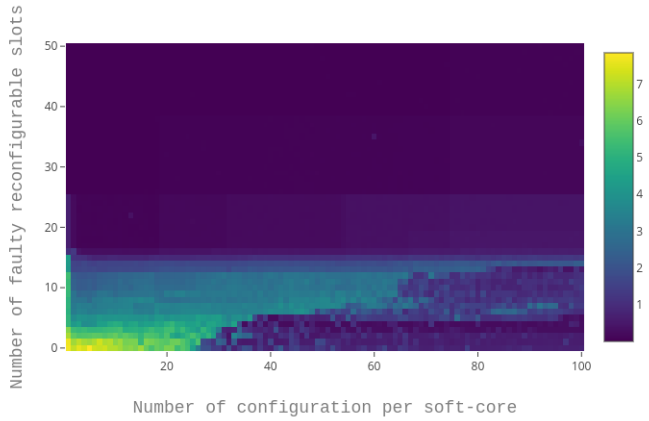

Fig. 8. The average waiting time normalized with respect to the average execution time

The average waiting time is proportional to the probability that the same reconfigurable slot is required by more than a hardware task. As the number of available configurations per soft-core increases, this probability and the waiting time decrease. The opposite scenario happens when the number of faulty slots grows: this probability and the waiting time increase accordingly. This trend is respected until 16 faulty slots. After this threshold, the waiting time drops since only a small fraction of client requests can be accepted, as reported in the following subsection.

C. Accepted client request ratio

Next questions to be answered are about the impact of the number of faulty reconfigurable slots and the number of configurations per soft-core on the acceptance of the client requests. First, Figure 9 highlights the fact that increasing the number of faulty slots reduces the number of accepted requests. The reduction is not linear. All client requests are accepted when there are no faulty slots, half of the requests are accepted for about 10 faulty slots and just about a quarter for 15 faulty slots. As anticipated in the previous subsection, we observed a break down in the region between 14 and 16 faulty slots, in which the client requests are almost halved.

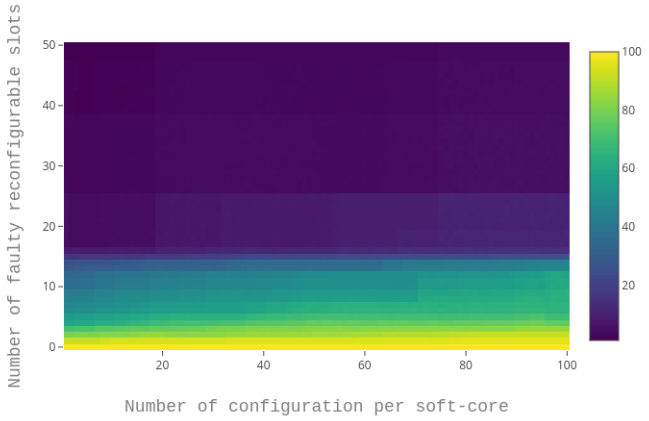

Fig. 9. The number of accepted client requests (100 client requests in total).

The advantages introduced by increasing the number of configurations per soft-core in terms of accepted client requests are presented in Figure 10, in which results of each row are normalized with respect to the case in which there is only one configuration available per soft-core (first column). This heatmap highlights two distinct behaviors depending on the region taken into account. More specifically, below the aforementioned break down at about 16 faulty slots, the benefits of having more configurations per soft-core are limited and a maximum improvement equal to 1.79 is achieved for 100 configurations and 14 faulty slots. Conversely, above the break down, the improvements increase linearly with respect to both the number of faulty reconfigurable slots and the number of available configurations per soft-core.

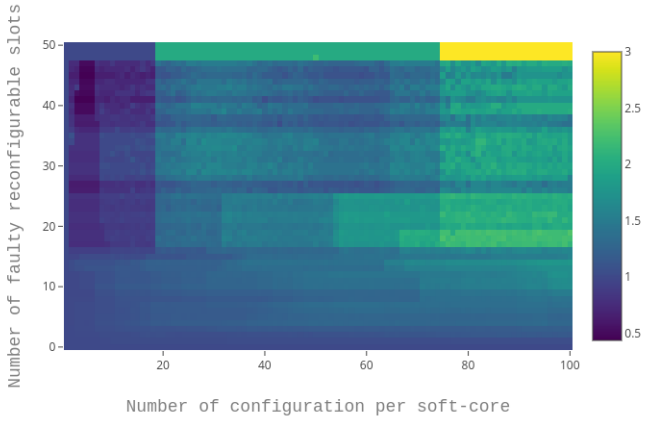

Fig. 10. The number of accepted client requests normalized with respect to the case there is only one configuration available per soft-core.

IV. Conclusion

This works addressed the design of an RTM for general purpose FPGA based heterogenous systems. Thanks to its server/client architecture, it introduces the possibility of accelerating tasks coming from other systems opening new scenarios to distributed computing. Alongside illustrating a new RTM, this paper also provides an efficient tool for the estimation of the performance of reconfigurable systems. In fact the simulator is able to reproduce a huge variety of scenarios by tuning some parameters.

Future works can address the study of new scheduling algorithms in a multi-fpga scenario, exploiting the possibility to have concurrent reconfiguration. Moreover further investigation can be devoted to the employment of the simulator during the design phase. Consequently the designed systems achieve best possible performance and the costs of unexploited reconfigurable hardware resources are reduced.

References

[1] M. Weinhardt, A. Krieger, and T. Kinder, "A framework for pc applications with portable and scalable fpga accelerators," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec 2013, pp. 1–6.

[2] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized fpga accelerators for efficient cloud computing," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 430–435.

[3] Intel, "Intel® programmable acceleration card with intel arria® 10 gx fpga," [Online] https://www.altera.com/pac, 2017.

[4] L. Cassano, D. Cozzi, S. Korf, J. Hagemeyer, M. Porrmann, and L. Sterpone, "On-line testing of permanent radiation effects in reconfigurable systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 717–720.

[5] M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Test strategies for reliable runtime reconfigurable architectures," *IEEE Transactions on Computers*, vol. 62, no. 8, 2013.

[6] C. Bolchini, A. Miele, and C. Sandionigi, "Autonomous fault-tolerant systems onto sram-based fpga platforms," *Journal of Electronic Testing*, vol. 29, no. 6, pp. 779–793, 2013.

[7] ——, "Increasing autonomous fault-tolerant fpga-based systems' lifetime," in *Test Symposium (ETS), 2012 17th IEEE European*, May 2012, pp. 1–6.

[8] S. D. Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and A. Vallero, "A novel methodology to increase fault tolerance in autonomous fpga-based systems," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, July 2014, pp. 87–92.

[9] S. D. Carlo, A. Miele, P. Prinetto, and A. Trapanese, "Microprocessor fault-tolerance via on-the-fly partial reconfiguration," in *2010 15th IEEE European Test Symposium*, May 2010, pp. 201–206.

[10] S. D. Carlo, P. Prinetto, and A. Scionti, "A fpga-based reconfigurable software architecture for highly dependable systems," in *2009 Asian Test Symposium*, Nov 2009, pp. 125–130.

[11] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/508352.508353

[12] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. De Wit, "A dynamic reconfiguration run-time system," in *Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*. IEEE, 1997, pp. 66–75.

[13] Y. Lu, T. Marconi, K. Bertels, and G. Gaydadjiev, "Online task scheduling for the FPGA-based partially reconfigurable systems," in *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2009, pp. 216–230.

[14] G. Mariani, V. M. Sima, G. Palermo, V. Zaccaria, C. Silvano, and K. Bertels, "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures." in *DATE*, W. Rosenstiel and L. Thiele, Eds. IEEE, 2012, pp. 1379–1384.

[15] G. Durelli, C. Pilato, A. Cazzaniga, D. Sciuto, and M. Santambrogio, "Automatic run-time manager generation for reconfigurable MPSoC architectures," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*. IEEE, 2012, pp. 1–8.