The RECIPE Approach to Challenges in Deeply Heterogeneous High Performance Systems

Giovanni Agosta[a], William Fornaciari[a], David Atienza[b], Ramon Canal[c], Alessandro Cilardo[d], José Flich Cardo[e], Carles Hernandez Luz[e], Michal Kulczewski[f], Giuseppe Massari[a], Rafael Tornero Gavilá[e], Marina Zapater[b]

[a]*Politecnico di Milano Milano, Italy*
[b]*École Polytechnique Fédérale de Lausanne Lausanne, Switzerland*
[c]*Barcelona Supercomputing Center and Universitat Politècnica de Catalunya Barcelona, Spain*
[d]*University of Naples Federico II / CeRICT Napoli, Italy*
[e]*Universitat Politècnica de València València, Spain*
[f]*Poznań Supercomputing and Networking Center Poznań, Poland*

Abstract

RECIPE (REliable power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems) is a recently started project funded within the H2020 FETHPC programme, which is expressly targeted at exploring new High-Performance Computing (HPC) technologies. RECIPE aims at introducing a hierarchical runtime resource management infrastructure to optimize energy efficiency and minimize the occurrence of thermal hotspots, while enforcing the time constraints imposed by the applications and ensuring reliability for both time-critical and throughput-oriented computation that run on deeply heterogeneous accelerator-based systems. This paper presents a detailed overview of RECIPE, identifying the fundamental challenges as well as the key innovations addressed by the project. In particular, the need for predictive reliability approaches to maximize hardware lifetime and guarantee application performance is identified as the key concern for RECIPE, and is addressed via hierarchical resource management of the heterogeneous architectural components of the system, driven by estimates of the application latency and hardware reliability obtained respectively through timing analysis and modelling thermal properties, mean-time-to-failure of subsystems. We show the impact of prediction accuracy on the overheads imposed by the checkpointing policy, as well as a possible application to a weather forecasting use case.

This manuscript has been submitted to the *Microprocessors and Microsystems*

Email addresses: agosta@acm.org (Giovanni Agosta), william.fornaciari@polimi.it (William Fornaciari), david.atienza@epfl.ch (David Atienza), ramon.canal@bsc.es (Ramon Canal), acilardo@unina.it (Alessandro Cilardo), jflich@disca.upv.es (José Flich Cardo), carherlu@upv.es (Carles Hernandez Luz), kulka@man.poznan.pl (Michal Kulczewski), giuseppe.massari@polimi.it (Giuseppe Massari), ratorga@disca.upv.es (Rafael Tornero Gavilá), marina.zapater@epfl.ch (Marina Zapater)

Special Issue on **European Projects in Embedded Systems Design**

Keywords: HPC; heterogeneous computing; run-time management;

1. Introduction and long-term objectives

The current increase in available data from a variety of sources and the emergence of new application domains that require massive amount of computational power is transforming the landscape of High-Performance Computing (HPC). HPC has always been an important infrastructure for several industrial fields (e.g., oil & gas, finance) as well as for weather forecasting and for the advancement of physics and chemistry. Nowadays, it can be considered a vital infrastructure for both nations and industrial players, as testifies by the massive investments that are leading towards the construction of Exascale systems.

Emerging application domains such as computational biochemistry and high-performance machine learning, coupled with the increasing heterogeneity of the underlying hardware architectures, require new technologies able to deal with increasingly complex systems, while preserving the performance levels that HPC users can currently leverage.

In sight of the Quality of Service (QoS) needs of these emerging application domains, as well as the need to manage computational resources in a more flexible way to address the partitioning of heterogeneous resources among applications exhibiting widely different ability to exploit specific accelerators, tightly coupled approaches for application development and management of HPC resources and energy are deemed necessary [1]. Other key challenges include the management of thermal issues through a balanced approach involving both hardware and software support, and the capability to predict failure occurrences.

The "REliable power and time-ConstraInts-aware Predictive management of heterogenous Exascale systems" (RECIPE) project targets such emerging HPC scenarios, characterized by new types of requirements, including time criticality and reliability, as well as new delivery models, e.g. cloud-based solutions [2], potentially open to a large audience of HPC users. These emerging scenarios require that new HPC platforms are able to support multiple applications running concurrently, possibly with conflicting Quality of Service requirements [3–7]. In addition, at the technology level, RECIPE addresses deep heterogeneity, based on dedicated accelerators like GPUs and FPGAs, as an enabling factor for improved energy efficiency, building on the results collected from previous research projects [8, 9].

The main focus in RECIPE is on resource management [10], targeted at performance predictability and reliability for HPC applications. To address current limitations, RECIPE will take a hierarchical approach to develop a Run-Time Management System (RTMS) partitioned in two layers, a *global* resource manager providing proactive fault tolerance and coarse-grained resource allocation, as well as a *local* resource manager, ensuring reactive fault tolerance and fine-grained resource allocation. The fine-grained resource management and predictive strategies will support the combined optimization of reliability and timing.

The interface between the runtime management system and the application will be mediated by the programming model, supporting the expression of application expectations. Furthermore, the runtime management system will rely on hardware monitors and a hardware abstraction layer, providing a uniform view of the available accelerators, including GPU, manycore, and FPGA devices, and exposing partitioning mechanisms used to ensure scalability, system-wide reconfigurability, and dedicated resource provisioning.

The RECIPE project aims at providing two key contributions.

A global manager and proactive strategies for reliability, energy efficiency and QoS. The *global resource manager* operates at the *system level*, and aims at increasing the system efficiency in terms of performance/watt, due to power / performance / thermal aware allocation; reducing the mean time to failure (MTTF) due to proactive strategies; and enforcing timing guarantees for critical applications.

A local manager and reactive strategies for reliability, energy efficiency and QoS. The *local manager* operates at the level of the *individual heterogeneous node* (typically composed of one or more CPUs as well as heterogeneous accelerators) aims at improving the overall utilization of non-CPU computing devices under heavy condition of massively parallel workload; improving the Energy-Delay product with respect to current state-of-the-art HPC infrastructures; reducing the probability of faulty executions; reducing the average recovery time with respect to classical checkpoint-restart approaches; and enforcing timing guarantees for critical applications in cooperation with the *global resource manager*.

To achieve the aforementioned goals RECIPE is developing technologies focusing on:

Fault prediction, prevention and recovery Through predictive techniques, we aim at improving the error detection coverage to provide more precise estimates of MTTF to the resource management infrastructure.

Thermal modelling technique By developing thermal models of the system, we aim at reducing thermal gradients, thus improving system MTTF due to thermal stress.

Timing analysis and enforcement Through more precise timing analysis and enhanced QoS enforcing at the run-time manager, we aim at achieving zero deadline misses for critical applications under the targeted failure rates.

Runtime manager integration with hybrid programming models By coupling the resource manager with hybrid programming models, we aim at radically reducing the integration cost, as well as simplifying the programming in such hybrid models.

Hardware abstraction layer for resource disaggregation Through an appropriate disaggregation layer allowing remote access to heterogeneous accelerators, we aim at increasing the effective usage of available resources under non-saturated conditions as well as identifying optimal matches of application requests and available resources.

Organization of the paper. The rest of this paper is organized as follows. In section 2 we review the technological background, while in section 3 we provide an overview of the deeply heterogeneous architecture upon which the RECIPE technologies are built, as well as the hardware abstraction layer that provides resource disaggregation. In section 4 we describe the key innovations provided by the RECIPE project in terms of predictive resource management and its integration with the programming model, while in section 5 we discuss the application scenario, with an in-depth analysis of one of them. In section 6 we draw some conclusions and highlight future research directions.

2. Background and open issues

This section describes the current technological background which is directly relevant for RECIPE. This spans several areas including run-time management, heterogeneous computing architectures, HPC memory/interconnection infrastructures, thermal modelling, reliability, programming models, and timing analysis.

2.1. Runtime resource management and reliability

The problem of unexpected hardware failures is rapidly growing, due to the increasing thermal stress sensitivity of hardware components in modern HPC systems. For some applications, reliability issues may become critical, because faults lead to silent data corruptions passing undetected by the hardware, or otherwise because mere re-execution upon a crash may lead to violating the expected execution time. In other words, for time-aware applications, the occurrence of some runtime error may lead to deadline misses (timing violations), even when appropriate error recovery mechanisms, e.g. checkpointing, are put in place. In the latest years, several *Dynamic Reliability Management* solutions have been introduced [11–14]. Most operate on a *reactive* basis, by exploiting checkpoint-restore protocols, to resume corrupted data or faulty application executions. *Proactive* approaches, based on suitable predictive models, have been recently proposed. However, such models are often platform-specific and based on single-node computing systems. Furthermore, they do not usually consider the application timing requirements. The goal is typically to maximize the lifetime of the hardware components.

2.2. Programming models

HPC applications are usually programmed by using *hybrid* programming models, combining a distributed memory model, often MPI, to manage the application parallelism at global level, and a shared memory model, to manage parallelism at the single-node level. In heterogeneous systems, the node-level programming model needs specialized support for heterogeneity, such as those provided by OpenCL or CUDA. In such heterogeneous programming models, the programmer is in charge of managing explicitly the required resources, which demands a large amount of boilerplate code. When resource management is available, it is generally placed at the top level (in the application dispatcher) or it operates under the assumption that an entire node is allocated to a single application. Such coarse approaches pose an inherent limitation, that will be addressed by RECIPE to enable application-driven management strategies.

2.3. Heterogeneity

Current trends in HPC are increasingly moving towards heterogeneous platforms, i.e. systems made of different computational units, ranging from general-purpose processors to graphics processing units (GPUs) and even special-purpose units made of custom acceleration logic, often implemented on field-programmable gate arrays (FPGAs) [15–18]. In fact, several players have recently introduced FPGA-based heterogeneous platforms used in a large range of high-performance computing applications [19], e.g. multimedia, machine learning, bioinformatics, etc. [19, 20], with speedups in the range of 10x to 100x. However, the effective exploitation of combined heterogeneous resources still poses an open issue in current HPC, and has only been demonstrated for very specific workloads.

2.4. HPC interconnect

Current HPC interconnect technologies include Ethernet [21], InfiniBand [22], as well as vendor specific interconnects, particularly Intel Omni-Path technology [23]. Ethernet, a dominant standard for mainstream commercial computing requirements, has continued to evolve, reaching performance levels of 400 Gbps in 2017. InfiniBand is designed for scalability, relying on a switched fabric network topology together with remote direct memory access (RDMA) to reduce CPU overhead which enables maintaining a performance and latency edge in comparison to Ethernet in many high performance workloads. The InfiniBand roadmap [24] details bandwidths reaching 600Gb/s data rate HDR in the middle of 2018 and 1.2Tb/s data rate NDR in 2020. Introduced in 2015, Intel's end-to-end Omni-Path Architecture (OPA) claims higher messaging rates and lower latency than InfiniBand, in addition to advanced features such as traffic flow optimization, packet integrity protection and dynamic lane scaling, although it is not currently a recognized standard. In RECIPE, InfiniBand will be the HPC interconnect of choice, because of the higher performance versus Ethernet as well as its role as a de-facto standard, unlike Intel OPA. However, the project will need to address crucial issues related to the interplay with the deeply heterogeneous acceleration fabric, for which no solutions exist exposing direct access to HPC interconnect technologies, as well as the interplay between the QoS mechanisms provided at RTMS and interconnect level.

2.5. Thermal modelling

Efficient thermal management requires accurate knowledge about the thermal profile of the chip in both steady and transient states. In conventional

Multi-Processor Systems on Chip (MPSoCs), the knowledge of the chip floorplan allows the design-time identification of the hot-spot locations. However, this approach is no longer possible in heterogeneous MPSoCs equipped with a reconfigurable fabric, as the thermal distribution depends on the accelerators implemented, which are unknown during the design phase. As a result, energy efficiency comes at the cost of shifting thermal evaluation from the chip design phase to the run-time management, affecting also runtime reliability [25]. This may be particularly relevant to RECIPE, because of the adoption of FPGA-based accelerators. The project will thus aim at leveraging accurate, fast, and flexible thermal simulation to demonstrate novel efficient thermal management strategies for heterogeneous HPC platforms.

2.6. Timing analysis

The large amount of computation and storage nodes integrated in HPC systems makes applications suffer from considerable performance variability, which will inevitably increase with the advent of exascale HPC systems. The uncertainty in the timing behavior of HPC applications is a consequence of either the internals of the application (intrinsic variability) or the interactions between different applications or with the system itself (extrinsic variability). Timing variability makes it difficult to achieve tight execution time bounds for the applications running in the computing infrastructure [26]. For applications with tight real-time requirements, like those addressed by RECIPE, standard practices of performance analysis are not enough and new methods are required to derive trustworthy and tight upperbounds to application execution times [27–30]

3. Architectural Concept

The RECIPE hardware architecture builds on the outcome of the MANGO H2020 project [5]. While the host-side part of the prototype includes commercial architectures like Intel Xeon [31, 32] as well as GPUs [33], heterogeneous acceleration relies on the MANGO Field Programmable Gate Array (FPGA) fabric [17] in addition to commercial HPC FPGA cards. The MANGO prototype, originally aimed at manycore architecture exploration, contains two main components, the General-purpose Node (GN), which includes commercial CPUs and GPUs, and the Heterogeneous Node (HN), made of customized motherboards mounting dedicated FPGA daughterboards developed by ProDesign GmbH. HNs offer a large degree of flexibility for exploring and instantiating customized accelerators and various heterogeneous system configurations. Among other features, the key architectural aspects of MANGO inherited in RECIPE include the communication infrastructure between the GN and HN, synchronization mechanisms between the host and the hardware application kernels, QoS guarantees, burst transfers, unified HN memory address space, and a basic HN hardware abstraction library.

An important challenge in RECIPE relates to the increased scale of the target HPC system. While MANGO focuses on the concept of an HN node made of several interconnected FPGAs, RECIPE targets a more ambitious scenario where several HN nodes coexist in the system, each defining its own set of accelerators. While the HNs are physically distributed throughout the HPC system, they are part of a single pool of accelerators. As an example, in MANGO an application can obtain a set of resources, like memory or accelerators, only from its corresponding HN node. In RECIPE we extend the application capabilities to use accelerators from different, efficiently interconnected HN nodes at the same time.

In order to fully match the potential of the RTMS, the RECIPE heterogeneous accelerators also expose hardware monitors and a suitable hardware abstraction layer, described in Section 3.2. The hardware abstraction layer provides a uniform view of the available accelerators, including GPU, manycore, and FPGA devices, exposing fine-grain partitioning mechanisms used to ensure scalability, system-wide reconfigurability, and dedicated resource provisioning.

Similar to the multi-accelerator compute platform, the memory/interconnect architecture also needs to scale up to the system level. The MANGO solution allows some basic bandwidth allocation strategies *within* the HN, in that HN accelerators and the connected server can obtain reserved interconnect bandwidth and memory resources, although this feature is limited to the applications running on the particular server connected to the HN. These capabilities are expanded in RECIPE from the node level to the system level, totally decoupling memory and interconnect management from the node boundary and exposing such functionality to the complete system. In order to achieve that purpose, we augment the MANGO prototype with a standard InfiniBand adapter, plugged into the GN part of the MANGO cluster, which in turn enables the integration of the reconfigurable accelerator fabric within the HN in the RECIPE prototype, such that the FPGAs inside of the HN can be seen as other heterogeneous devices in the system. Unlike QoS, reliability-related mechanisms are not directly supported by the MANGO technology. However, MANGO allows multiple accelerator *replicas* to be instantiated and used on the same HN or on different HNs. This feature paves the way to new flexible mechanisms for enhancing fault-tolerance in RECIPE. In fact, in the project we provide node-level (within the HN node) and system-level (within the Resource Manager) reliability mechanisms able to detect and isolate malfunctioning devices located in the HN node, then exploiting the underlying reconfigurability to enforce proactive and reactive fault-tolerance strategies.

3.1. The RECIPE prototype

Figure 1 illustrates the prototype built in the context of RECIPE. It is composed of a set of heterogeneous nodes interconnected through a high-performance InfiniBand network. The prototype includes commercial CPUs and GPUs, plus commercial of-the-shelf (COTS) FPGAs and the MANGO HN cluster. This assembly results in a deep heterogeneous system that incorporates the most common state-of-the-art heterogeneous resources for HPC systems.

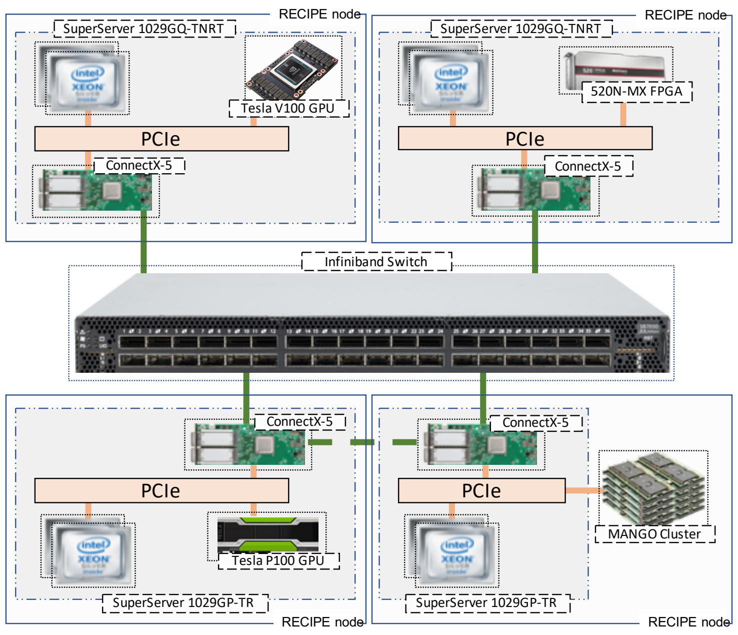

Figure 1: RECIPE prototype

3.2. Hardware abstraction layer and low-level runtime support

One of the challenges that need to be addressed when designing highly heterogeneous systems lies on keeping the programability of such systems under control, meanwhile diminishing any sort of performance degradation. In fact, different compute architectures mean distinct instruction sets (ISAs) and/or programming flows, thus making the programming of these systems complex. In RECIPE, we rely on the a local resource manager, a hardware abstraction layer (HAL) and a custom low-level runtime environment to deal with this demanding task.

The HAL developed for the RECIPE project provides a unique C-based programming application interface to: I) access the different type of heterogeneous resources targeted, II) support the different acceleration modes supported by RECIPE heterogeneous accelerators, and III) expose to the applications and the to the local resource manager, a global and unified view of these resources, which can be accessed independently of the physical location they are located, mimicking the concept of resource dissagregation already explored in datacenters for both CPU and GPU devices.

To satisfy its goals, the HAL relies on the distributed low-level runtime environment that is shown on Figure 2. It consists of two main components: the HN library and the HN daemon. On the one hand, the HN library is the part that an application process, that wants to access the RECIPE heterogeneous resources, has to be linked to. This library includes monitoring, synchronization

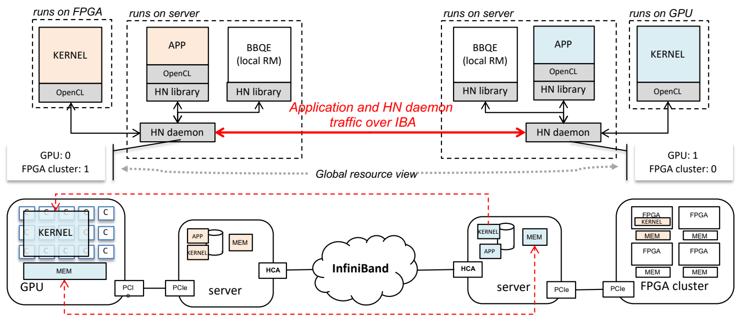

Figure 2: Low-level runtime architecture.

and communication support, among other features. In general, it establishes communication with the HN daemon for: obtaining the different resources the system is composed of, together with status information of them; setting QoS application requirements; allocating memory and computing resources; transferring data and launching kernels.

The HN daemon is a distributed user space process that captures the HN library requests and it executes them on the correct accelerator or resource device. As a distributed process, there is an instance of it running in each system node. These instances discover themselves by means of a hierarchical discovery service which allows every instance to have the same global view of resources, plus the communication cost for accessing each device, that depends on the node that is used to access them. In the figure, this process is illustrated with the *box calls* connected to the HN daemon. As it can be seen, both processes can manage the same heterogeneous resources, but the cost to access them differs depending on the number of network hops required to reach them. In that way, the daemon on the left of the figure has direct access to the GPU, but a cost of one hop to access the remote FPGA cluster, since we are assuming that the servers in the figure are exactly one host distance away each other for this example. Thus, the communication cost is a useful value that should be taking into account for the scheduling and mapping strategy implemented by the resource manager.

To offer the low-level runtime services to the applications through the HAL in a simple manner the HN library provides a library wrapper to capture OpenCL function calls, so that OpenCL-based applications can run in the system without modifications. In addition, these applications can access every resource in the system in a location-independent way. This is shown in the figure with the dash red lines. As it can be seen, an OpenCL-based application (marked in blue color) running on the server of the right is using the remote GPU to execute its OpenCL kernel.

4. Runtime management system

RECIPE aims at addressing the reliability problem by operating both at the node and at the infrastructure level, thanks to a *hierarchical* resource management approach. Our objective is to integrate reactive and proactive solutions, while maximizing the lifetime of the system, as well as providing reliability guarantees to time-sensitive applications. For the latter, we need to minimize the use of reactive mechanisms and maximize the effectiveness of predictive resource management, thus supporting the combined optimization of reliability and timing.

Based on this approach, the RECIPE runtime management system will be partitioned in two layers, a *global* resource manager providing proactive fault tolerance and coarse-grained resource allocation, as well as a *local* resource manager, ensuring reactive fault tolerance and fine-grained resource allocation. In particular, RECIPE will build on the integration of BarbequeRTRM [34] and the SLURM-based Global Resource Management (GRM) framework. These will cooperate on a hierarchical basis, with GRM being in charge of dispatching the workload at the global infrastructure level, and BarbequeRTRM mapping the application's tasks and memory requests at the local node level. Each instance of BarbequeRTRM (one per node) will take into account the timing requirements of the application dispatched by the GRM layer, the current status of the node, i.e. load and power consumption when available, and the performance profile of the available (heterogeneous) computing resources. To this aim, the BarbequeRTRM relies on the hardware abstraction layer described in Section 3.2, to get knowledge of the heterogeneous set of processing units and memory nodes installed into the system, and enforce the resource allocation decisions of the management policies. The specific case of processing units of type CPU instead, is based on the abstraction and the control capabilities offered by the Linux frameworks *cgroups* and *cpufreq*, to isolate the "host-side" application threads and perform frequency scaling.

As specific technical actions planned by RECIPE, we will extend the BarbequeRTRM capabilities by: 1. feeding the policies with data coming from suitable models, in order to implement *reliability-aware* and *timing-driven* resource allocation policies; 2. providing a local checkpoint-restore mechanism, transparent with respect to the application, to resume the execution of application's tasks and optionally redefine the set of assigned resources. On the other hand, the GRM capabilities will also be extended by: 1. adding a proactive reliability module that will communicate with the 3D-ICE [35] temperature modelling module, enabling reliability modelling and prediction; 2. proposing new reliability-aware runtime global management strategies; and 3. enhancing the capabilities of distributed computing of the GRM, enabling the application execution to span across multiple nodes.

In order to support the above predictive strategies, RECIPE will develop novel models integrated with run-time management policies, running and cooperating at both the node and the infrastructure level. Furthermore, to assess the impact of reliability and thermal issues on the overall system architecture, we will also rely on simulation, which will be possible thanks to the recent gem5-X framework. In RECIPE, gem5-X [36] will be equipped with both thermal and reliability simulation to evaluate the impact of architectural choices and resource management strategies in short and long term reliability. We will investigate state-of-the-art techniques in fault prediction technologies [37]. Existing fault-prediction techniques are generally based on the existence of a model of the system that has the ability to predict the occurrence of faults when the real-time performance, power, and temperature measurements of the different system components indicate that faulty situations are likely to occur. These models can be built by using fault statistics of systems based on similar technologies, when available, or by using information provided by the component manufacturer which allows creating simulation models of the system. When both approaches are combined, synthetic models can be enriched or calibrated at runtime with information collected during the system lifetime. Given that the system temperature is a very important contributor to the component reliability, fault prediction models will also incorporate the thermal behavior. The fault/performance prediction, based on application statistics, will then drive the runtime resource manager during scheduling and task mapping, in order to minimize missed deadlines and adapt the checkpoint rates to various fault recovery strategies, on a given HPC system configuration. In the experimental section, we carried out a preliminary analysis of three first possible policies for tuning the checkpoint rate, observing the expected impact on the application execution time.

4.1. Reliability

A reliable HPC infrastructure must include *Checkpoint/Restore (C/R)* mechanisms, in order to enable the possibility of recovering the execution of applications experiencing faults at run-time. This is a typical *reactive* approach to fault recovery. In RECIPE, we aim at minimizing the probability of triggering reactive actions, due to the costs of the restore procedures. However, our runtime management software also implements these mechanisms in order to improve system robustness by covering cases of misprediction of the reliability models. Checkpoint/Restore techniques are normally classified in application-level, user-level, and system-level solutions [38]. The latter is operated by the system hosting the job in a transparent way and in turn is divided in solutions relying on Operating System support and hardware-level support. We base our C/R technique on a system-level approach, inherently needing a combined support from the underlying hardware and OS/RTMS. This means that, from the programming model perspective, the involvement of the application developer is not required. Checkpoints are periodically performed, by the resource managers, which are therefore in charge of handling the trade-off between reliability and overhead, according to the application and the overall system requirements.

In addition, as one of its main objectives, RECIPE directly addresses the heterogeneous nature of modern HPC systems, particularly *deeply* heterogeneous architectures based on FPGAs. In that respect, given the importance of Checkpoint/Restore in traditional HPC, RECIPE aims at extending the C/R approach to FPGA accelerators, providing a new reliability mechanism for emerging classes of HPC applications. We highlight that the proposed mechanism is not alternative to the range of low-level techniques for FPGA reliability [39–41], which can coexist with the RECIPE solution since they involve either the physical device or the particular user design. Rather, the FPGA-extended C/R techniques explored in RECIPE provide a way of effectively controlling the FPGA-side execution state from the host system, so as to checkpoint and restore it as part of a larger application state. As such, the proposed approach is orthogonal to the type of faults, e.g. aging or single event upsets, which might affect the FPGA device and, indeed, it is instrumental to preserving the integrity of the system-wide application state upon faults involving any part of the HPC infrastructure.

More specifically, the FPGA C/R techniques explored in RECIPE can either take a coarse-grain approach, where the whole state of the acceleration card (the FPGA device chip as well as the off-chip memory) is saved and restored, or a fine-grain approach, where a *permanent state* can optionally be defined by the user, 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. For *off-chip* state, i.e. DDR or HBM memory available in the card, the user is required to explicitly indicate the memory address windows that contain the permanent state, while for *on-chip* state, the user is required to instantiate in the FPGA design 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. Regarding the access mechanism, RECIPE considered a range of different solutions, whose actual availability might depend on the specific FPGA device. In particular, the checkpointing and restoring of the FPGA execution state may be supported through dynamic partial reconfiguration by means of internal ports, such as Xilinx Internal Configuration Access Port (ICAP), which provide the ability of reconfiguring the device from the design itself, requiring however a suitable instrumentation of the hardware design. Alternatively, device state readback and restore operations, in either a global or a partial form, can take place through JTAG, a well-established industry standard for verifying and debugging integrated circuits and printed circuit boards, which all FPGA devices normally provide in order to upload the configuration image to the device as well as retrieve data from the device. In the case of the JTAG access mechanism, for accessing the off-chip memory one must either require that the external memory device (a DDR device or a HBM chiplet) has itself a JTAG port accessible on the same scan chain or, as an alternative solution, make the FPGA-side memory controller accessible from the JTAG logic and, hence, from the host-side software. In the Xilinx devices used in RECIPE, for example, this is possible by means of a special JTAG-to-AXI master component, which essentially allows the FPGA JTAG port to control an additional AXI master in the FPGA, competing with the user design for access to the memory controller and hence to the off-chip memory.

4.2. Interplay with programming models

As highlighted above, most RTMS-level choices will be driven by application-specific QoS requirements and behavior profiles. This is particularly important because RECIPE envisions scenarios where multiple unrelated applications will co-exist on the same nodes, thus requiring system-wide resource management in order to maximize utilization of (possibly heterogeneous) compute resources. To support resource management at the node level, the application needs to be able to interact with the resource manager in a way that is as much transparent as possible to the programmer, so as to provide the resource manager with the appropriate information about the required Quality of Service and the type of resources that the application can effectively exploit. The interface between the runtime management system and the application in RECIPE will be mediated by the programming model, supporting the expression of such application expectations. Existing experimental models, however, do not fully match the requirements of typical industry-grade implementations. In particular, there is not support for dynamic compilation, nor for any source language except C/C++. In RECIPE, we will build on the *mangolib* model and implementation developed in the MANGO FETHPC project [5] to integrate system-wide resource management within the programming model, and will extend it to reach industry-grade maturity. The current *mangolib* model allows the application to specify a different "recipe" for each kernel, as well as multiple implementations of the same kernel, targeting different accelerators. The runtime manager then automatically manages the selection of the best configuration according to the application priority level, quality of service requirements, and resource availability, while optimizing system-wide metrics of utilization and energy-efficiency.

In order to minimize the effort in terms of application porting, we recently built an OpenCL wrapper layer on top of *mangolib*. This layer comes with a very minimal set of changes in the common OpenCL API specification. In Listing 1, we show a classical example of OpenCL sample application: *matrix multiplication*, implemented by using the custom OpenCL API. The functions introduced or modified have been highlighted in red. Here below, we can summarize the porting steps that a developer should carry out to adapt an OpenCL standard compliant implementation to the run-time managed version:

1. Replace the original OpenCL header file `CL/cl.h`, with the `mango_cl.h` header, coming with the *mangolib* installation (line 1).

2. Instantiate a `mango_data` data structure, and fill it with information regarding the application name and the "recipe" file name, including the per-kernel requirements (lines 17-19).

3. Pass the `mango_data` structure to the `clCreateCommandQueue` function as 5th argument (line 20).

4. Once the buffers have been created and populated (lines 53-58), pass them to the `clEnqueueNDRangeKernel`, in order to bind the kernel to their input and output buffers (lines 79-83).

5. Call the `clStartComputation` to trigger the resource mapping and, afterwards, the kernel execution (line 88).

Listing 1: Matrix multiplication: OpenCL implementation on top of *mangolib*

```
1  #include <mango/mango_cl.h>
2
3  int main()
4  {
5     int err;
6
7     // Platform and device retrieval are kept for code portability
8     cl_uint dev_cnt = 0;
9     clGetPlatformIDs(0, 0, &dev_cnt);
10    cl_platform_id platform_ids[10];
11    clGetPlatformIDs(dev_cnt, platform_ids, NULL);
12
13    cl_device_id device_id;
14    clGetDeviceIDs(platform_ids[0],
15                   CL_DEVICE_TYPE_ALL, 1, &device_id, NULL);
16
17    // Add MANGO library specific data to the OpenCL context
18    struct cl_mango_data mango_data;
19    mango_data.application_name = "matrix_multiplication";
20    mango_data.recipe           = "mm";
21    cl_context context = clCreateContext(0, 1,
22                                 &device_id,
23                                 NULL,
24                                 &mango_data,
25                                 &err);
26
27    cl_command_queue commands = clCreateCommandQueue(context,
28                                 device_id,
29                                 0,
30                                 &err);
31    cl_program program;
32 #ifdef BUILDKERNEL
33    char *KernelSource;
34    // Load the kernel source for online compilation
35    program = clCreateProgramWithSource(context,
36                   1, (const char **) &KernelSource, NULL, &err);
37 #else
38    char* KernelBinaryPath = "yourpathtobin";
39    program = clCreateProgramWithBinary(context,
40                         1,
41                         &device_id,
42                         NULL,
43                         (const unsigned char **) & KernelBinaryPath,
44                         NULL,
45                         &err);
46 #endif
47
48    err = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
49
50    // Creation of the OpenCL kernel object
51    cl_kernel kernel = clCreateKernel(program, "matriMul", &err);
52
53    // Host-side allocation of buffers for the matrices
54    int* h_A = (int*) malloc(mem_size_A);
55    int* h_A = (int*) malloc(mem_size_B);
56    int* h_C = (int*) malloc(mem_size_C);
57
58    // Creation of the OpenCL buffer objects
59    cl_mem d_A = clCreateBuffer(context,
60                         CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
61                         mem_size_A,
62                         h_A,
63                                 &err);
64      cl_mem d_B = clCreateBuffer(context,
65                                 CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
66                                 mem_size_B,
67                                 h_B,
68                                 &err);
69      cl_mem d_C = clCreateBuffer(context,
70                                 CL_MEM_READ_WRITE,
71                                 mem_size_C,
72                                 NULL,
73                                 &err);
74
75      // Kernel arguments
76      cl_mem in_buffers[2], out_buffers[1];
77      in_buffers[0]  = d_A;
78      in_buffers[1]  = d_B;
79      out_buffers[0] = d_C;
80      int wA = NR_ROWS_MATRIX_A;
81      int wC = NR_ROWS_MATRIX_C;
82      err = clSetKernelArg(kernel,  0, sizeof(cl_mem), &d_B);
83      err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_A);
84      err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_C);
85      err |= clSetKernelArg(kernel, 3, sizeof(int),    &wA);
86      err |= clSetKernelArg(kernel, 4, sizeof(int),    &wC);
87
88      // Put kernels and buffers together
89      err = clEnqueueNDRangeKernel(commands,
90                                 kernel,
91                                 &in_buffers, 2,
92                                 &out_buffers, 1,
93                                 0,
94                                 NULL, NULL);
95
96      // Asynchronous wait for the kernel to complete the
97      // writing of the output buffer
98      err = clEnqueueReadBuffer(commands,
99                                 d_C,
100                                CL_TRUE,
101                                0,
102                                mem_size_C,
103                                h_C,
104                                0,
105                                NULL, NULL);
106
107     // MANGO-specific application execution trigger
108     clStartComputation(commands);
109
110     free(h_A);
111     free(h_B);
112     free(h_C);
113     clReleaseMemObject(d_A);
114     clReleaseMemObject(d_C);
115     clReleaseMemObject(d_B);
116     clReleaseProgram(program);
117     clReleaseKernel(kernel);
118     clReleaseCommandQueue(commands);
119     clReleaseContext(context);
120
121     return 0;
122 }
```

With this implementation, the kernel-to-device mapping is tranparently managed by the local resource manager, as mentioned above. This means that the platform and device selection calls (lines 7-15) do not have any real effect on the kernel compilation and offloading, until the function `clStartComputation` is called. Here the programming library waits for the resource manager to provide all the mapping information, before launching the actual kernel execution. The remaining part of the application (host-side) code is perfectly OpenCL standard compliant.

4.3. Proactive/reactive thermal management

To enable effective thermal management, RECIPE develops a holistic simulation tool for a wide range of cooling techniques and settings, building on 3D-ICE and using the most recent pluggable heatsink models that enable simulation of arbitrary cooling models [42], and using the gem5-X framework [43]. The goal of these tools is to model and simulate temperature in a spatio-temporal way, feeding the data to the reliability models to estimate MTTF.

For this to be a reality, the advances and enhancements undertaken in RECIPE are as follows:

- Enabling the simulation of arbitrary state-of-the-art cooling mechanisms, such as the ones found in current servers. In particular, within RECIPE we need to enable the simulation of both natural convection mechanisms (i.e., heatsinks), and forced convection cooling (i.e., heatsinks plus fans). This is done by incorporating a pluggable heatsink model into the 3D-ICE thermal simulator, which we have released as version 2.2.7 of 3D-ICE.
- Proposal of a methodology that will allow us to assess the impact of the main control knobs related to temperature in today's servers, which range from workload allocation, DVFS setting and fan speed control policies [44].
- Proposal of a methodology to adequately link the thermal aspects to the reliability of the system. For this purpose, we will use the MTTF and reliability models proposed, which will be incorporated into the resource management policies.

To accomplish this goal, we need ways of validating temperature against real devices. For this purpose, within the RECIPE project we have also created a platform which comprises a real test chip [45] for accurate thermal characterization. In particular, this platform is based on a Thermal Test Chip (TTC), an integrated circuit containing an array of power dissipating elements and an array of temperature sensors. Our thermal platform is capable of applying a generic power dissipation pattern to the thermal test chip and measuring the corresponding temperature map, at a rate up to 1 kHz. This capability allows to measure the temperature map of an integrated circuit subject to reference power dissipation maps, and thus to design and validate thermal models. In our particular case, the chip is organised as a 4 by 4 array of individual cells, each capable of temperature sensing and power generation through a resistive element. The heating element in each cell is capable of dissipating up to 12 W. This allows to simulate the equivalent of a 16-core chip with a total power dissipation of 192 W.

The methodology we envision for enabling DTM and reliability management is the one depicted in Figure 3 and described in [44]. The simulator will enable

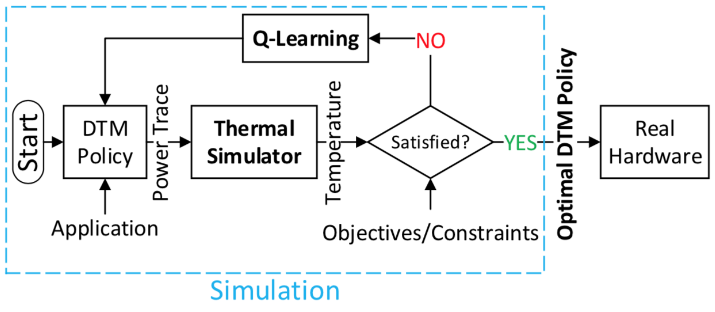

Figure 3: Thermal modelling methodology to enable DTM and reliability management [44]

us perform an exploration of the impact of aggressive DTM management policies which may not only cause performance degradation and additional power consumption, but more importantly, jeopardizes lifetime reliability of the whole system. In fact, one of the main reasons that makes researchers reluctant to consider fan speed control as a key DTM approach is the lack of a transient thermal simulator for MPSoCs with proper integration of fans. The incorporation of fan models in 3D-ICE, and the use of this methodology provides a comprehensive framework for exploring thermal effects of DTM policies in a safe way.

Workload allocation, DVFS and fan speed altogether drastically increase the number of runtime design parameters to be decided by a DTM and reliability-aware policy, which leads to additional challenges to find the best values for optimal behavior of the whole system. Our methodology enables the exploration of this design space in an automated way. In particular, we envision the use of Reinforcement Learning (RL) techniques. The RL agents can explore the design space using 3D-ICE and combine that with the impact of reliability. Initial experiments [45], prove the feasibility of our approach.

4.4. Timing constraints management

RECIPE has developed a methodology for HPC application analysis of timing behavior for driving the decisions made by the RTMS. Our approach builds upon techniques based on memory layout randomization for increasing test coverage [46, 47], as well as measurement-based probabilistic timing analysis (MBPTA) [29, 30] to predict rare (high) timing behavior beyond those values observed in applications with a continuous flow of (big) data.

This means that the application can come with timing requirements in terms of tasks completion times (deadlines) and related probability value p. Such a value represents the required probability of meeting the deadline specified. In other terms, $1 - p$ is the acceptable probability of experiencing deadline misses. Accordingly, a suitable characterization of the timing responses of the application must be performed, considering the heterogeneous set of resource allocation options that the RTMS must take into account. Then, at run-time, the RTMS can exploit this characterization to drive the actual process of resource mapping, thus selecting the combination of processing units and the memory resources that should provide the required timing guarantees.

In this paper, we report the characterization of a geophysical exploration application which builds upon Full Waveform Inversion (FWI), a cutting edge technique that aims to acquire the physical properties of the subsoil from a set of seismic measurements [48, 49].

The complete tool flow for probabilistic worst-case execution time provides:

1. Exploration of execution conditions. We integrate a software randomization layer in the geophysical exploration application to test its susceptibility to memory layouts caused by different code, heap and stack allocations.

2. Worst-Case Execution Time (WCET) analysis. We analyze and fit an MBPTA technique for WCET prediction so that it can be used in the context of HPC applications running on high-performance systems.

In the remaining of this section, we evaluate those techniques on the geophysical exploration application, proving their viability to study its (high) execution time behavior, and we show that appropriate integration of those techniques allows scaling the application to the use of parallel paradigms, thus beyond the execution conditions considered in embedded systems -where WCET and pWCET estimates were originally proposed and used.

In particular, MBPTA has been applied at the application level. To measure the accuracy of our approach, we compute the MET (*Maximum Execution Time*) of the application in 1000 independent runs. We also compute the MET when using randomization to measure the accuracy of the pWCET for each of the randomization techniques. The application was run in isolation in the system (a common setup for critical or large applications in HPC infrastructures). More details can be found in [50].

Finally, we show the absolute maximum execution time (MET) and pWCET estimates for different problem sizes at an exceedance threshold of 10^{-6} per run in Figure 4. We show results for -memory layout- randomization of code (C), stack (S) and heap (H). We also show no-randomization for comparison purposes. Overall, absolute differences are small, thus showing that pWCET estimates are very close to the MET out of 1,000 runs in absolute terms, which indicates that unobserved execution times can only be slightly higher than observed ones. Moreover, results show that stack and heap randomization cause little performance variation. However, code randomization causes a noticeable impact in MET and pWCET estimates, thus showing that code memory placement is particularly critical for this application. Relative results on the increase of the pWCET estimates w.r.t. MET are shown in Figure 5. As we increase problem size, execution time variability decreases in relative terms, which leads to sharper pWCET distributions and hence, smaller increase in relative terms of the pWCET estimates w.r.t. MET. Hence, the larger the problem size is, the more stable the execution time is. The use of SWrand [47] along with MBPTA-CV provides strong guarantees on these observations, thus allowing an efficient

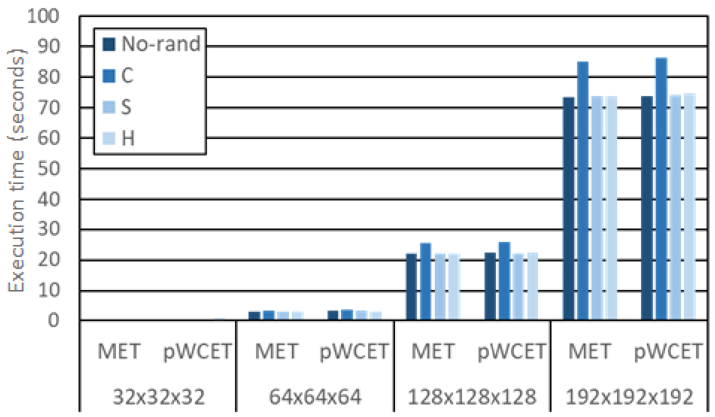

Figure 4: Absolute maximum execution time (MET) and pWCET estimate (in seconds) at an exceedance probability of 10^{-6} for the different configurations of the application (FWI).

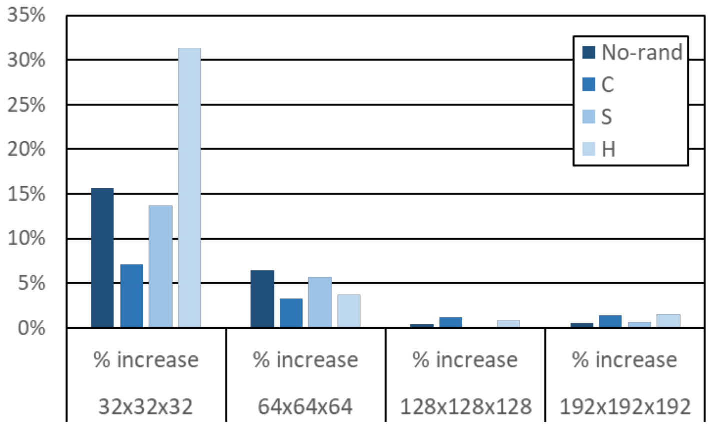

Figure 5: Relative increase of the pWCET estimate at an exceedance probability of 10^{-6} w.r.t. maximum execution times (MET) for the different configurations.

use of resources scheduling HPC applications accounting for their unobserved high execution times. Further analysis can be found in [50].

To the best of our knowledge, no specific methods have been devised to predict reliably high execution times of HPC applications. Hence, our adaptation of a mechanism –MBPTA-CV – used in another domain to fit the needs and constraints of HPC systems is a pioneering attempt to reach the goal of reliable and tight estimation of execution time bounds for HPC applications.

4.5. Impact of prediction-based checkpointing policy

In this subsection, we focus on how the run-time resource management infrastructure can minimize the checkpoint overhead, which introduces some penalty in the workload execution times, by exploiting the aforementioned models. The goal is therefore, to minimize the checkpoint rate, controlled by the local resource manager, according tothe expected failure probability.

To assess the impact of prediction-based checkpointing, we performed a preliminary analysis, by comparing three possible policies. As a baseline, we considered a standard reactive *restart* mechanism – when the application fails, the system restarts it. No checkpointing is needed, but for many use cases the restart cost is not acceptable. With the periodical checkpointing feature enabled instead, we considered different possible policies to control the checkpoint rate:

- *fixed-rate*: periodical checkpoints of managed applications performed at regular intervals;
- *prediction-based*: checkpoint the application in the immediate proximity of a predicted failure;
- *error-tolerant*: considering the predicted MTTF, attempts to checkpoint at 90% of the predicted MTTF, to account for the inherent inaccuracy of the prediction models.

Figure 6 shows the results of a simulated experiment, carried out under the following assumptions: the failure rate is set so that the baseline (not reported) has an average slowdown of 1; checkpoint time is around 1.5-2% of the application (ideal) execution time; restart overhead is about 15-20% of the application (ideal) execution time. Figure 6 reports, the average overhead due to reliability (i.e., $O = T_{exe} - T_{ideal})/T_{ideal})$, as a function of the MTTF prediction error. For the execution of the experiments, we generated random workloads over a long time window (around 200× the execution time of a single application). Experiments have been repeated 20 times, taking median values. The fixed-rate policy was configured with a checkpointing interval set at 20× the checkpointing time. As shown in the figure, prediction-based policies are more effective when an effective prediction (relative prediction error is shown on the y axis) is available. The error-tolerant policy instead, can come into play in case the reliability models cannot guarantee prediction errors lower than 2.5%.

5. Weather forecasting use-case

RECIPE identified a number of real-world high performance applications demonstrating the emerging class of requirements addressed by the project in terms of heterogeneity, fault-tolerance, and time constraints. In particular, the

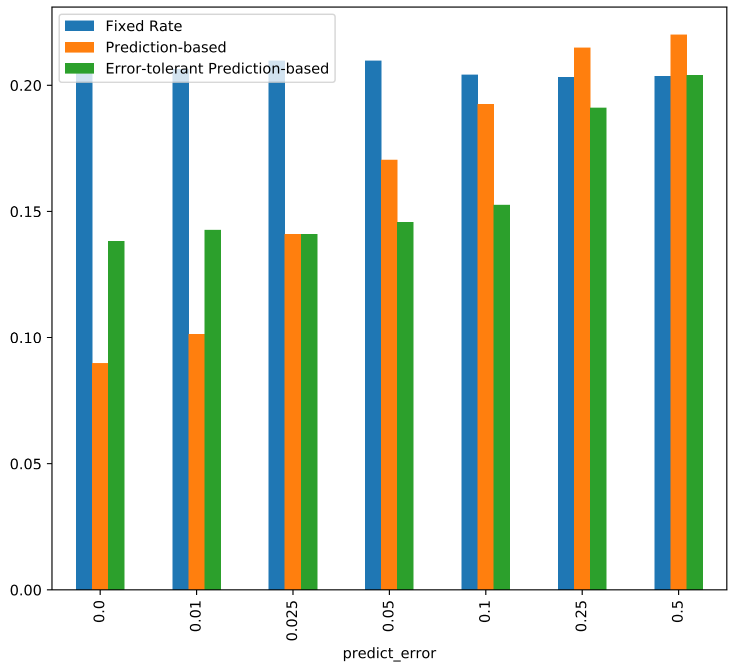

Figure 6: Performance impact (slowdown over a failure-free execution) of different prediction models. Y-axis: $T_{exe} - T_{ideal})/T_{ideal}$, X-axis: MTTF prediction error.

RECIPE use cases include applications in the areas of weather forecasting, subsoil properties identification, and bio-medical big-data applications. The ultimate objective in RECIPE is to perform a demonstration on both industry-grade, pre-Exascale systems and on emerging deeply heterogeneous technologies, in order to measure the degree of success of the project.

In the weather forecasting area, two application are addressed: the first one, UrbanAir, is a multiscale model that allows for weather prediction, wind farm optimizations and prediction of air quality over complex urban areas [51], see Figure 7 for example. The second one is a flood prediction model, supported by on-field devices.

The UrbanAir uses two models - WRF for weather prediction at mesoscale to local level, and EULAG for detailed wind prediction and air quality modelling at local to street level. Both models are HPC application written in FORTRAN that uses MPI parallel programming paradigm. Both models have been coupled to each other so that prediction results of the former are used as input parameters to the latter. As for the EULAG model, there are some very small parts of code, called kernels, available for the GPU accelerators, though they are not part of the production code yet.

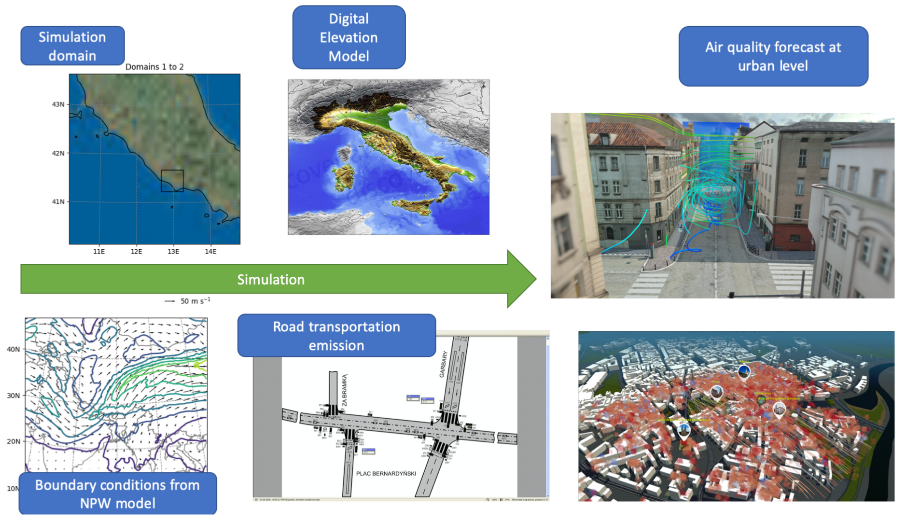
Simulation domain
Domains 1 to 2
Digital Elevation Model
Air quality forecast at urban level
Simulation
Road transportation emission
Boundary conditions from NPW model
ZA BRAMKA
GARBARY
PLAC BERNARDYŃSKI

Figure 7: High level representation of the UrbanAir application.
Different input parameters, e.g. digital elevation model, road transporation emission, mesoscale weather prediction are used to model air quality over urban areas, which are then visualized at city and street scale.

Within the RECIPE context, the UrbanAir application will benefit from fault tolerance and time constraints features in particular. Weather prediction for emergency scenarios, such as flood prediction, requires timely analysis. This will be achieved by combining different methods of speeding up simulations (e.g. coarser domain, longer time-step, different physics and parametrization schemes) and RECIPE tools to obey required time regime. For the long-running scenarios, e.g. weather prediction for wind farm optimization, the key requirements is to provide reliable forecast under given time constraint and hardware/-software failures. The UrbanAir application already supports restarting simulation from a given point in time, which will be coupled to fault tolerance solution provided within the project.

In the emergency scenarios, such as flood prediction, 15 minutes regime has been defined within which NWP prediction has to be delivered. To support DSOs operations, NWP prediction has to be provided within 24 hours. The application performance may be sensitive to memory layout or task placement across HPC machine. In such situation, the MBPTA helps to analyze the worst case execution, so that proper parameters values of the simulation may be selected, as well as the highest possible resolution of the simulation domain. In this way, NWP prediction is delivered with the highest quality of results possible under given time constraints.

6. Conclusions

In this paper, we presented an overview of the technological challenges emerging from the coupling of deeply heterogeneous HPC architectures in Exascale systems with emerging application domains characterised by QoS requirements. We identified proactive and reactive reliability mechanisms as necessary to support this scenario, and introduced the key innovations brought by the RECIPE H2020 FETHPC project to address is. Such technologies span runtime management, heterogeneous computing architectures, HPC memory/interconnection infrastructures, thermal modelling, reliability, programming models, and timing analysis.

We provided an initial assessment of the impact of reliability techniques such as checkpoint/restore mechanisms when coupled with fault prediction models. During the second half of the RECIPE project, we will work towards the integration of such mechanisms in the context of the disaggregated management of the heterogeneous computing resources, to provide a flexible environment to run applications with widely different requirements.

Acknowledgement

The activities described in this article received funding from the European Union's Horizon 2020 research and innovation programme under the FETHPC grant agreement no. 801137 RECIPE: REliable power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems.

References

[1] J.-P. Panziera *et al.*, "Strategic Research Agenda 2017," ETP4HPC, Tech. Rep., Nov 2017.

[2] B. Koller *et al.*, "Towards an environment to deliver high performance computing to small and medium enterprises," in *Sustained Simulation Performance 2015.* Cham: Springer International Publishing, 2015, pp. 41–50.

[3] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella, "Managing Heterogeneous Resources in HPC Systems," in *Proc. of PARMA-DITAM '18.* ACM, 2018, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/3183767.3183769

[4] J. Flich *et al.*, "Enabling HPC for QoS-sensitive applications: The MANGO approach," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 702–707.

[5] ——, "Exploring manycore architectures for next-generation HPC systems through the MANGO approach," *Microprocessors and Microsystems*, vol. 61, pp. 154 – 170, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933118300243

[6] G. Massari, A. Pupykina, G. Agosta, and W. Fornaciari, "Predictive resource management for next-generation high-performance computing heterogeneous platforms," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'19)*, Jul 2019.

[7] A. Pupykina and G. Agosta, "Optimizing Memory Management in Deeply Heterogeneous HPC Accelerators," in *2017 46th Int'l Conf on Parallel Processing Workshops (ICPPW)*, Aug 2017, pp. 291–300.

[8] J. Flich *et al.*, "The MANGO FET-HPC Project: An overview," in *IEEE 18th Int'l Conf on Computational Science and Engineering (CSE)*. IEEE, 2015, pp. 351–354.

[9] M. Zanella, G. Massari, A. Galimberti, and W. Fornaciari, "Back to the future: Resource management in post-cloud solutions," in *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications*, ser. INTESA '18. New York, NY, USA: ACM, 2018, pp. 33–38. [Online]. Available: http://doi.acm.org/10.1145/3285017.3285028

[10] W. Fornaciari, G. Agosta, D. Atienza, C. Brandolese, L. Cammoun, L. Cremona, A. Cilardo, A. Farres, J. Flich, C. Hernandez, M. Kulchewski, S. Libutti, J. M. Martínez, G. Massari, A. Oleksiak, A. Pupykina, F. Reghenzani, R. Tornero, M. Zanella, M. Zapater, and D. Zoni, "Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18. New York, NY, USA: ACM, 2018, pp. 187–194. [Online]. Available: http://doi.acm.org/10.1145/3229631.3239368

[11] C. L. Chou and R. Marculescu, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

[12] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *2016 Design, Automation Test in Europe Conference (DATE)*, March 2016, pp. 854–857.

[13] L. Huang and Q. Xu, "Characterizing the lifetime reliability of manycore processors with core-level redundancy," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2010, pp. 680–685.

[14] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. Rosing, "WARM: Workload-aware reliability management in Linux/Android," *IEEE Trans on CAD of Integrated Circuits and Systems*, 2016.

[15] "https://euroexa.eu."

[16] "https://www.altera.com/products/sip/memory/stratix-10-mx/overview.html."

[17] "http://www.mango-project.eu."

[18] A. Cilardo and L. Gallo, "Interplay of loop unrolling and multidimensional memory partitioning in HLS," in *Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 163–168.

[19] K. Paranjape, S. Hebert, and B. Masson, "Heterogeneous computing in the cloud: Crunching big data and democratizing HPC access for the life sciences," Intel Corporation, Tech. Rep., 2010.

[20] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. Pande, "Hardware accelerators for biocomputing: A survey," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, May 2010, pp. 3789–3792.

[21] "IEEE standard for Ethernet - Amendment 10," *IEEE Std 802.3bs-2017*, pp. 1–372, Dec 2017.

[22] G. F. Pfister, "Aspects of the InfiniBand architecture," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, Oct 2001, pp. 369–371.

[23] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel® Omni-Path architecture: Enabling scalable, high performance fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, Aug 2015, pp. 1–9.

[24] "https://www.infinibandta.org/infiniband-roadmap/."

[25] D. Zoni, L. Cremona, and W. Fornaciari, "All-digital energy-constrained controller for general-purpose accelerators and CPUs," *IEEE Embedded Systems Letters*, 2019, doi:10.1109/LES.2019.2914136.

[26] F. Reghenzani, G. Massari, W. Fornaciari, and A. Galimberti, "Probabilistic-WCET reliability: On the experimental validation of EVT hypotheses," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, ser. COINS '19. New York, NY, USA: ACM, 2019, pp. 229–234. [Online]. Available: http://doi.acm.org/10.1145/3312614.3312660

[27] F. Reghenzani, G. Massari, and W. Fornaciari, "chronovise: Measurement-based probabilistic timing analysis framework," *J. Open Source Software*, vol. 3, no. 28, p. 711, 2018.

[28] F. Reghenzani, G. Massari, and W. Fornaciari, "The misconception of exponential tail upper-bounding in probabilistic real-time," *IEEE Embedded Systems Letters*, 2018, doi:10.1109/LES.2018.2889114.

[29] L. Cucu-Grosjean et al., “Measurement-based probabilistic timing analysis for multi-path programs,” in *ECRTS*, 2012.

[30] J. Abella, M. Padilla, J. D. Castillo, and F. J. Cazorla, “Measurement-based worst-case execution time estimation using the coefficient of variation,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 4, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3065924

[31] “https://lanl.gov/projects/trinity/specifications.php.”

[32] “https://www.bsc.es/marenostrum/marenostrum/technical-information.”

[33] “https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/.”

[34] P. Bellasi, G. Massari, and W. Fornaciari, “Effective runtime resource management using linux control groups with the BarbequeRTRM framework,” *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 2, pp. 39:1–39:17, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2658990

[35] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, “3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling,” in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2010, pp. 463–470.

[36] Y. M. Qureshi, W. A. Simon, M. Zapater Sancho, K. Olcoz, and D. Atienza Alonso, “Gem5-X: A Gem5-based system level simulation framework to optimize many-core platforms,” *Proceedings of the 27th High Performance Computing Symposium (HPC 2019)*, p. 12, 2019.

[37] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, “Fault prediction under the microscope: A closer look into HPC systems,” in *SC ’12: Proc. of the Int. Conference on High Performance Computing, Networking, Storage and Analysis*, Nov 2012, pp. 1–11.

[38] I. Egwutuoha, D. Levy, B. Selic, and S. Chen, “A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems,” *The Journal of Supercomputing*, vol. 65, 09 2013.

[39] K. Lee and S. S. Wong, “Fault-Tolerant FPGA with Column-Based Redundancy and Power Gating Using RRAM,” *IEEE Transactions on Computers*, vol. 66, no. 6, pp. 946–956, 2017.

[40] J. A. Cheatham, J. M. Emmert, and S. Baumgart, “A survey of fault tolerant methodologies for FPGAs,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 2, pp. 501–533, 2006.

[41] M. G. Parris, C. A. Sharma, and R. F. Demara, “Progress in autonomous fault recovery of Field Programmable Gate Arrays,” *ACM Computing Surveys*, vol. 43, no. 4, 2011.

[42] A. Iranfar, F. Terraneo, W. A. Simon, L. Dragic, I. Pilji, M. Zapater Sancho, W. Fornaciari, M. Kovac, and D. Atienza Alonso, "Thermal characterization of next-generation workloads on heterogeneous MPSoCs," 2017.

[43] D. Zoni and W. Fornaciari, "Modeling DVFS and power-gating actuators for cycle-accurate NoC-based simulators," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 3, pp. 27:1–27:24, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2751561

[44] A. Iranfar, F. Terraneo, G. Csordas, M. Zapater, W. Fornaciari, and D. Atienza, "Dynamic thermal management with proactive fan speed control through reinforcement learning," *[Proceedings Design, Automation and Test in Europe Conference and Exhibition (DATE)]*, p. 6, 2020.

[45] F. Terraneo, A. Leva, and W. Fornaciari, "An open-hardware platform for mpsoc thermal modeling," in *International Conference on Embedded Computer Systems.* Springer, 2019.

[46] C. Curtsinger and E. D. Berger, "STABILIZER: Statistically sound performance evaluation," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 219–228, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2490301.2451141

[47] L. Kosmidis, C. Curtsinger, E. Quiones, J. Abella, E. Berger, and F. J. Cazorla, "Probabilistic timing analysis on conventional cache designs," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 603–606.

[48] J. Kormann, J. Rodríguez, N. Gutiérrez, M. Ferrer, O. Rojas, J. De la Puente, M. Hanzich, and J. María Cela, "Toward an automatic full-wave inversion: Synthetic study cases," *The Leading Edge*, vol. 35, pp. 1047–1052, 12 2016.

[49] M. Hanzich, J. Kormann, N. Gutiérrez, J. Rodríguez, J. De la Puente, and J. María Cela, "Developing full waveform inversion using hpc frameworks: Bsit," in *EAGE Workshop on High Performance Computing for Upstream*, 09 2014.

[50] M. Fusi, F. Mazzocchetti, A. Farres, L. Kosmidis, R. Canal, F. J. Cazorla, and J. Abella, "On the use of probabilistic worst-case execution time estimation for parallel applications in high performance systems," *Mathematics*, vol. 8, no. 3, pp. 314–335, Mar 2020. [Online]. Available: http://dx.doi.org/10.3390/math8030314

[51] D. W. Wright, R. A. Richardson, W. Edeling, J. Lakhlili, R. C. Sinclair, V. Jacauskas, D. Suleimenova, B. Bosak, M. Kulczewski, T. Piontek, P. Kopta, I. Chirca, H. Arabnejad, O. O. Luk, O. Hoenen, J. Weglarz, D. Crommelin, and D. Groen, "Building confidence in simulation: Application of easyvvuq," submitted to Journal of Advanced Theory and Simulations on 12/12/2019.