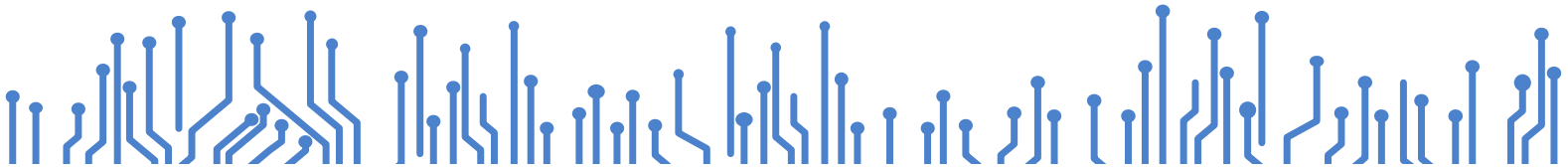
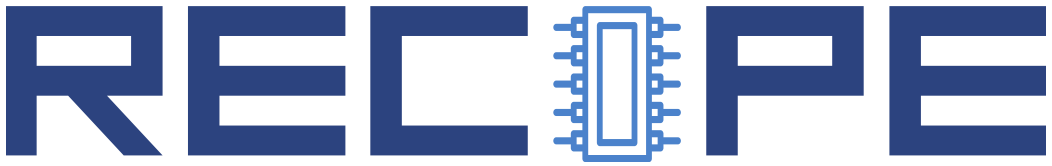
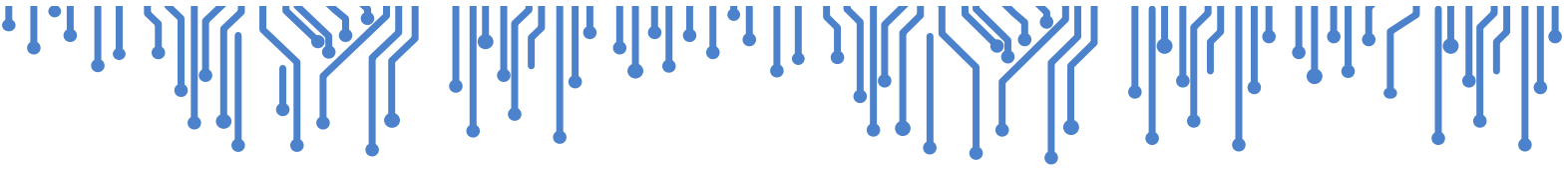
REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems





WP3 Predictive Reliability and QoS Enforcing Methodologies

D3.3 RECIPE Fault Prediction Tools



http://www.recipe-project.eu

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

Grant Agreement No.: 801137
Deliverable: D3.3 RECIPE Fault Prediction Tools

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

Deliverable No:	D3.3
WP No:	3
WP Leader:	R. Canal
Due date:	30/04/2020
Delivery date:	07/05/2020

Dissemination Level:

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

DOCUMENT SUMMARY INFORMATION

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

D3.3 RECIPE Fault Prediction Tools

Work Package:	WP3 Predictive Reliability and QoS Enforcing Methodologies
Deliverable number:	D3.3
Deliverable title:	RECIPE Fault Prediction Tools
Due date:	30/04/2020
Actual submission date:	07/05/2020
Editor:	R. Canal
Authors:	F. Mazzocchetti, R. Canal, M. Fusi, L. Kosmidis, F.J. Cazorla, J. Abella, C. Hernandez, R. Tornero, J. Flich, Giuseppe Massari, Federico Reghenzani
Dissemination Level:	PU
No. pages:	15
Authorized (date):	07/05/2020
Responsible person:	W. Fornaciari
Status:	Submitted

Revision history:

Version	Date	Author	Comment
v.0.1	13/04/2020		Outline and contributions identified
v.1.0	27/04/2020		Final version after internal review

Quality Control:

	Who	Date
Checked by internal reviewer	J. Abella	27/04/2020
Checked by WP Leader	R. Canal	27/04/2020
Checked by Project Technical Manager	G. Agosta	07/05/2020
Checked by Project Coordinator	W. Fornaciari	07/05/2020

COPYRIGHT

ACKNOWLEDGEMENTS

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

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

DISCLAIMER

Contents

1 Introduction **6**

2 Summary of the Reliability Methodology **7**

3 Reliability Tool **9**
3.1 CPU and memory monitoring 9
3.1.1 Structure of the Hardware Reliability Tool 9
3.1.2 How to use the library 10
3.2 Library for FPGA monitoring 11
3.2.1 How to use the library 12
3.3 GPU monitoring and the Local Resource Manager 12

4 Summary **14**

1 Introduction

High-Performance Computing (HPC) systems have become ubiquitous and are no longer concentrated in supercomputing facilities and data centers. While these facilities still exist and grow, a plethora of HPC systems building on large multicores and accelerators (e.g. GPUs, FPGA-based) are nowadays deployed for a variety of applications of interest, not only for large enterprises and public institutions, but also for small and medium enterprises as well as small public and private bodies.

The proliferation of HPC systems and applications in new domains has led to new requirements related to non-functional requirements (time, power, reliability, temperature, etc) and the implementation of platforms to satisfy them [1, 5, 4, 7, 10]. In this deliverable, we provide a description of the reliability tool (fault prediction tool) developed in this project.This prediction of the aging of the different hardware components of the platform will -later on- guide application deployment and resource management, leveraging aspects related to the lifetime of the platform as a whole. The aging monitoring and prediction is based on two main input parameters:

- Aggregate block usage: from functional units and memories to chip level and system level aggregates.
- Connection to the thermal tools (T3.3 and D3.5) for more accurate prediction (as temperature has a key influence in degradation).

The rest of the deliverable provides details on the usage and capabilities of the software developed. The specific underlying techniques implemented are described in the previous deliverable D3.2.

The tool developed is available at: RECIPE's GIT repository and they are being integrated in the run-time manager (Task 3.5)

2 Summary of the Reliability Methodology

Reliability at the system level needs to take into consideration the different -heterogeneous- blocks that build it. In RECIPE, this means a system composed of a multicore CPU and several accelerators (GPU and FPGA). Overall, the system should provide information (on the reliability status of each block). For some blocks where the hardware provides more information, the system can leverage more detailed information (e.g. for the CPU, we may distinguish between the memory hierarchy and the processor core).

System reliability measurements will be done through the computation of the FIT rate (number of failures in 10^9 hours of operation). System reliability may be computed as an aggregate of multiple blocks (e.g. CPUs, caches, memories, accelerators ...). Individual –independent- FIT rates can be combined to a system by:

$$FIT = \frac{number\,of\,failures}{t_{op}}$$

$$FITsystem = \sum_{i=1}^{N}(FIT_i)$$

Similarly, the FIT rate depends on the fundamental vulnerability of the block (also known as raw FIT rate). From the raw FIT rate, there are several derrating factors in the system [11]. This effectively means that any fault in the circuitry, may not be visible to the architectural state (i.e. on-chip code and values of the application being run). To take this into account, we will use the Architecture Vulnerability Factor -AVF- which measures block occupancy. In short, if there is a fault in a block we are not using (i.e. it does not hold the architectural state), then it will not affect the system (i.e. it will be masked). AVF can be factored in in the FIT rate computation as follows:

$$FITsystem = \sum_{i=1}^{N}(AVFblock_i * FIT_i)$$

Finally, electronic circuits degrade over time. Among other causes, temperature is a key contributor. As degradation increases the FIT rate of components, we will factor it in as an acceleration factor (Af) of the "operation" time (HMOL model [3]). Thus:

$$top + d = top * Af$$

and Af is defined as:

$$Af = e^{\frac{Ea}{k}*(\frac{1}{Tuse}-\frac{1}{Tstress})}$$

where:

Ea is the activation energy (eV) of the failure mode ($Ea = 0.7eV$ for transistors)

k is the Boltzmann's Constant ($k = 8.617x10^{-5}eV/K$)

Tuse is use temperature (standardized at 55°C or 328°K)

Tstress is the stress temperature (higher than T_{use} means stress, lower means recovery) (in ^{o}K)

This model has been shown to adapt to different sources of acceleration (NBTI, HCI, TDDB, electromigration) [2, 8, 9, 12]. If so, individual FIT rates for each source should be computed and then aggregated. This is a powerful model that will let us incorporate different sources of degradation into our system-wide model.

Implementation details can be found in deliverable D3.2. In this report, we describe the software delivered.

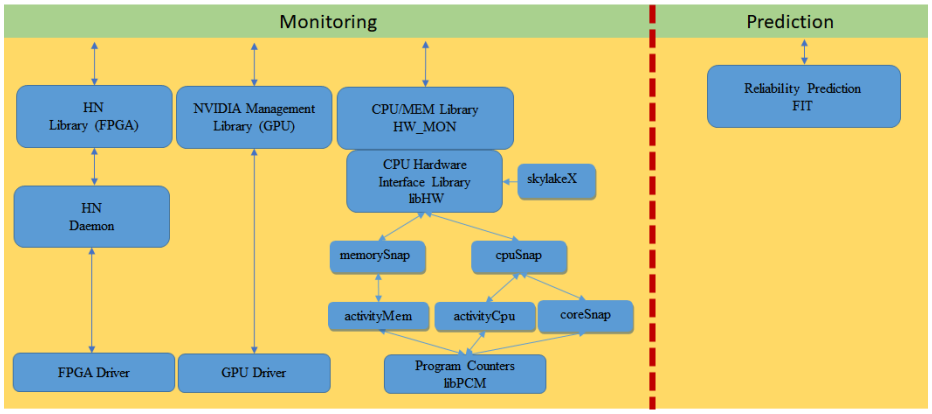


Figure 1: Reliability tool breakdown into components

3 Reliability Tool

The main goal of the tool is to provide a Failure in Rate (FIT) estimation of the different system blocks (i.e. CPU, memory and accelerators) under the execution of a program in the system. The FIT rate will be computed as determined in the previous section. To archive this goal, we monitor the activity of the system is monitored during the program execution. This is performed using the monitoring capabilities of the hardware devices as descrived in D3.2.

3.1 CPU and memory monitoring

Accessing the monitoring capabilities requires certain privileges for the access to the specific information. Similarly, certain conditions need to be enforced so that the readings are significant.

- Sudo privileges. They are necessary because the code can access to PMC using specific x86 instructions, which are under sudo privileges domain.
- On the target machine all cores need to be a fixed frequency. This trick is necessary for avoid cycles different in term of times. For example, 1 cycles in core A is 10ns, in B core is 20ns. For simplicity you can set the governor to performance in all cores under Linux domain. There are other ways to do it.
- No interference in the machine.

For the prediction part, there is no specific requirement. As long as the data supplied is significant, the prediction performed will be accurate -within the model's capabilities.

3.1.1 Structure of the Hardware Reliability Tool

The monitoring tool (integrated in a single library) is composed by two libraries . One of them is the Intel PCM library which is used to handle the access to PMC and retrieve CPU and memory topology information (libPCM) . The second library (libHW) is used to handle the data, perform the modeling computation and provide the results and provide an easy interface to use. Finally, we have a third library (FIT) that provides the accounting of the failure rate (in FIT units). This is separated from the hardware interface (libPCM) and the model computation

(libHW) so that the software can instantiate it anywhere, without needing to instantiate the specific modelling and/or hardware interface.

The structure of the library is depicted in Figure 1. The concept of Snap is an important concept if the user wants to modify something. In this library, a Snap is photography of the object. For example, if we have 10 cpuSnap object, it means we have 10 shot photography of the CPU state. This because the 'int loop' of the entry function is set to 10.

File Name	Description
fit.cpp	FIT class
`hw_mon.cpp/h`	Main part of the library. It is like the main of the library
coreSnap.cpp/h	Information of each core
cpuSnap.cpp/h	Information of the CPU, include information about sockets, cores, topology. Activity
memorySnap.cpp/h	Information of the Memory, abstract layer of all memory. Activity
activityCpu.cpp/h	Activity for CPU
activityMemory.cpp/h	Activity for Memory
skylakeX.h	Parameter of skylake architecture.

3.1.2 How to use the library

In the caller code (e.g. the runtime manager), we have to first include the headers of the libraries. For instance, if they are located in the "lib/libHW" folder:

```
#include "lib/libHW/hw_mon.h"
#include "lib/libHW/fit.h"
```

The first header contains the only function to use under the caller perspective. It is detailed below. The second header is for reading and writing the current and computed FIT values. The FIT object offers Getter functions to retrieve the FIT results. If the system has several socket and/or many memory nodes it will report an average for the whole system. For example, if my system has 4 sockets, the FIT CPU will be an average of these 4 CPU sockets. At the start of the system, the user (or system administrator) needs to insert the manufacturers' FIT values (or an estimation of those). The tool allows the user to define every CPU block FIT value (e.g. ALU, caches, etc.), Everytime the reliability model is called, these FIT values are updated accordingly.

Then, we have to compile the caller program adding the library. If the cpp file is called "example.cpp" just type in the command line (The libPCM uses the pthread library, for that reason we need to add this library at compile time):

```
g++ example.cpp -L./lib -lHW_MON -lpthread -o example
```

The entry point of the library is a function. The definition is in the `HW_MON` header and it is as follows:

```
void entry(char * program[], fit &fit, int t_use_standardized,
           int loop, int fix_temp);
```

Where,

Parameter Name	Description
char * program[]	It is the program which you want to execute under the system monitoring. You need pass the entirely path of the program and its parameter. The last parameter need to be a NULL
fit &fit	Object which holds fit input parameter and offer fit results values.
int t_use_standardized	It is the temperature for the Acceleration Factor calculation.
int loop	Default value to set is 1. If you want calculate the fit using an average of sub data just put the iteration number here. The program will make 'loop' iteration and will calculate the fit values with these averages.

3.2 Library for FPGA monitoring

The access to FPGA and GPU heterogeneous resources in the RECIPE prototype is provided by the HN Library and HN Daemon. These two components expose a pletora of features for user applications to work with these type of resources, independently of their physical location in the final system. With regards to the activities of WP3, the most important functionalities exposed to upper software layers consist of enabling the reading of temperature and occupation counters of the heterogeneous resources.

In RECIPE, we have adopted OpenCL language for programming GPUs and FPGAs. Thus, the HN library implements the most common functions of the OpenCL specification. One of these function is *clGetDeviceInfo*. We have extended the behavior of this function for getting temperature reads and kernel occupation information for FPGA devices. The prototype of the function, got from Khronos OpenCL specification[6], is as follows:

```
cl_int clGetDeviceInfo(cl_device_id device, cl_device_info param_name,
size_t param_value_size, void *param_value, size_t *param_value_size_ret)
```

The parameters of this function are summarized next.

Parameter Name	Description
cl_device_id device	OpenCL resource returned by *clGetDeviceIDs*.
cl_device_info param_name	An enumeration that identifies the device information being queried. It can be one of the values as specified in the table below for reading temperature or occupation.
size_t param_value_size	Specifies the size in bytes of memory pointed to by param_value. This size in bytes must be greater than or equal to size of return type specified in the table below.
void *param_value	A pointer to memory location where appropriate values for a given param_name as specified in the table below will be returned. If param_value is NULL, it is ignored.
size_t *param_value_size_t	Returns the actual size in bytes of data being queried by param_value. If param_value_size_ret is NULL, it is ignored .

Apart from the enumeration values described in the Khronos OpenCL specification for the *cl_device_info* data type, we have provided two extension for letting applications to read temperature and occupation information for FPGA resources, as next table specifies.

cl_device_info	Description
CL_DEVICE_CORE_TEMPERATURE_RECIPE	Return type: cl_int. Temperature value.
CL_DEVICE_CORE_OCCUPATION_RECIPE	Return type: cl_int. Metric of the device occupation for the reliability model.

3.2.1 How to use the library

We rely on OpenCL programming model for both host and kernel applications. Thus, the header files that should be included by the host part source code of an application are as follows:

```
#include <CL/opencl.h>
#include <CL/cl_ext_upv.h>
```

Let's assume that the HN library and headers files has been installed to "./lib/libHW". Then, the application should be compiled and linked basically in the same way as compiling an application for using a standard OpenCL library.

```
g++ example.cpp -I./lib/libHW -L./lib -lHW_MON -lOpenCL -lpthread -o example
```

Once the application has been compiled and previously to its execution, the user should add the location of the OpenCL library wrapper that comes with HN Library to the `LD_LIBRARY_PATH` environment variable, when it is not installed in the default system location.

3.3 GPU monitoring and the Local Resource Manager

For the monitoring of the GPUs instead, vendors like NVIDIA and AMD already provide suitable run-time libraries (NVIDIA Management Library and AMD Display Library). These libraries, allow us to observe the current load (activity) of the processing units, their temperature, and dynamically change the clock frequency, whenever requested by the run-time management policy.

These libraries sit next to the FPGA and CPU monitoring library, previously described, as shown in Figure 1. Together, they provide the status of the different hardware pieces to the run-time manager. The run-time manager, subsequently, can trigger the reliability prediction library for accurate estimate computation not only for each piece, but for the whole heterogeneous system as a whole.

The integration of these low-level HW libraries into a single component of the RECIPE software stack, allows the local resource manage to represent a unified abstraction layer, on top of a deeply heterogeneous node. Moreover, since the BarbequeRTRM performs the monitoring of the hardware status, it is the sole data provider for the reliability prediction library.

As already explained in other deliverables, the BarbequeRTRM will benefit from the integration with the Reliability tools, since the data collected will be used to provide better reliability status

of the underlying system; as well as, it enables the implementation of reliability-aware resource management policies.

4 Summary

The proliferation of heterogeneous HPC systems and applications in new domains has led to new requirements related to non-functional requirements (time, power, reliability, temperature, etc) and the need for platforms to satisfy them. In this deliverable, we provided a summary of the progress achieved in the following activities related to different QoS aspects, as part of WP3:

- Reliability tool. We reported the tool created. We showed that it is aplicable to any heterogeneous system and we showed how we can integrate all reliability and degradation measurements into a single system value. We described the predictive mechanism we will integrate in the run-time manager.

This deliverable has described the fault prediction tool developed as part of WP3. The tasks has progressed according to the plan and we look forward to the use of the novel approaches for deriving better run time manager policies in the remaining tasks of the WP.

References

[1] 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*, pages 7–12. ACM, 2018.

[2] DES. Constant temperature accelerated life testing using the arrhenius relationship, 2013. https://www.desolutions.com/blog/2013/08/constant-temperature-accelerated-life-testing-using-the-arrhenius-relationship/.

[3] P. Ellerman. Calculating reliability using fit & mttf: Arrhenius htol model.

[4] J. Flich, G. Agosta, et al. Exploring manycore architectures for next-generation HPC systems through the MANGO approach. *Microprocessors and Microsystems*, 61:154 – 170, 2018.

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

[6] Khronos Group Inc. OpenCL specification. https://www.khronos.org/opencl/.

[7] Giuseppe Massari, Anna Pupykina, Giovanni Agosta, and William 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.

[8] H. Miyamoto. Semiconductor reliability and quality handbook.

[9] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 29–40, Dec 2003.

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

[11] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. Di Natale, D. Gizopoulos, and S. Di Carlo. Cross-layer reliability evaluation, moving from the hardware architecture to the system level. *Microprocess. Microsyst.*, 39(8):1204–1214, November 2015.

[12] P Vassiliou and A. Mettas. Understanding accelerated life-testing analysis. In *2002 Annual Reliability and Maintainability Symposium.*, Dec 2002.