Approche logicielle pour améliorer la fiabilité d'applications parallèles implémentées dans des processeurs multi-cœur et many-cœur

Vanessa Carolina Vargas Vallejo

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **NANO ÉLECTRONIQUE ET NANO TECHNOLOGIES**

Arrêté ministérial : 25 mai 2016

Présentée par

Vanessa Carolina VARGAS VALLEJO

Thèse dirigée par **Raoul VELAZCO, Directeur de Recherche, CNRS**
et codirigée par **Jean-François MÉHAUT, Professeur, Grenoble UJF**

préparée au sein **Laboratoire Techniques de l'informatique et de la Microélectronique pour l'Architecture des Systèmes Intégrées**
et de **l'École Doctorale Électronique, Électrotechnique, Automatique, Traitement du Signal (EEATS)**

Approche logicielle pour améliorer la fiabilité d'applications parallèles implémentées sur des processeurs multi-cœur et many-cœur

Thèse soutenue publiquement le **28 avril 2017**,
devant le jury composé de :

Monsieur Fréderic PÉTROT
Professeur, Président, Grenoble INP

Monsieur Luc GIRAUD
Directeur de recherche, Rapporteur, INRIA Délegation aquitaine

Monsieur Olivier ROMAIN
Professeur, Rapporteur, Université de Cergy-Pontoise

Monsieur Nacer-Eddine ZERGAINOH
Maître de conférences, Examinateur, Université Grenoble Alpes

Acknowledgement

First and all above, I would like to thank ***God***, the almighty, for all the blessings He shed in my life, without Him and his support this work would not be possible.

Special thanks to the Ecuadorian government through the Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador and Universidad de las Fuerzas Armadas ESPE for the confidence placed in me and my abilities to fulfill this challenge.

I would like to express my deep thankfulness and appreciation to my academic advisor, Dr. Raoul Velazco for his helpful guidance, valuable availability and precious optimism throughout the development of this research.

I want to express my deep gratitude to my academic co-advisor, Prof. Jean -François Méhaut for his constant involvement, constructive advices, and priceless readiness during this thesis.

Special thanks to my thesis reviewers Prof. Luc Giraud et Prof. Olivier Romain, for their invaluable comments and relevant questions that help me to improve this document.

I acknowledge all the staff of TIMA and LIG laboratories for their hospitality and cooperation, special thanks to Anne-Laure Fourneret, Fréderic Chevrot and Christian Seguy for their availability and service.

I would like to greatly appreciate Maud Baylac, Francesca Villa and Solene Rey from the LPSC laboratory for their long-lasting cooperation in the execution of the radiation tests experiments.

I want to express my gratitude to Stéphane Gailhard and Renaud Stevens from Kalray Company for their support in the applications' development and the experiments on the MPPA-256 many-core processor.

I would like to thank all my friends that I met during this stage of my life, specially Robert, Jessica, Alejandro, Audrey, Martin, Anaïs, Adrien, Cristina et Edison.

Special deep thanks to my mother, my mother-in-law and my aunt Talita for all the efforts and sacrifices that they have made on behalf of my family and for taking care with love of my children during different periods along these research years.

I want to thank all my family for their prays, invaluable support and love, specially to my parents, who have always trusted in me and encouraged me to fight for my dreams, and my siblings for their unconditional availability, confidence and help during these years.

I want to thank my little children, for their great comprehension and tenderness, for their smiles, kisses and hugs that fortify my soul.

Lastly, I want to thank my wonderful husband for his motivation, encouragement, and patience during the development of this thesis, for support me every time especially in the tough moments, love me and make me happy.

Vanessa Vargas

Dedication

To the love of my life, Pablo, and my tresors, Matías and Emmanuel...

To my beloved parents Guido and Lorgia...

"The Lord is my strength and my shield;
my heart trusts in him, and he helps me"
Ps 28:7

Contents

List of Tables

List of Figures

List of abbreviations

ACROSS Advanced Cockpit for Reduction Of StreSs and workload

ALU Arithmetic Logic Unit

AMP Asymmetric Multi-Processing

API Application Programming Interface

ARINC Avionics Application Standard Software Interface

AVF Architectural Vulnerability Factor

BMP Bound Multi-Processing

BSP Board Support Package

CAPACITES Calcul Parallèle pour Applications Critiques en Temps et Sureté

CAST Certified Associate in Software Testing

CC Compute Cluster

CEU Code Emulated Upset

CL Confidence Level or Limit

CMOS Complementary Metal-Oxide Semiconductor

CMP Chip Multi-Processor

C-NoC Control NoC

CORSE Compiler Optimizations and Runtime Systems

COTS Commercial Off-The-Shelf

CPU Central Process Unit

CR Condition Register

CRC Cyclic Redundancy Check

CS Compute Status

DCR Device Control Register

DECOS Dependable Embedded Components and Systems

DECC Double ECC

DMR Double Modular Redundancy

D-NoC Data NoC

DPAR Data Parity

DSU Debug Support Unit

DUT Device Under Test

EASA European Aviation Safety Agency

ECC Error Correcting Code

EDAC Error Detection And Correction

EMC2 Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real time environments

EnergySFE Energy-aware Scheduling and Fault Tolerance Techniques for the Exascale Era

FAA Federal Aviation Administration

FIT Failure In Time

FLOPS Floating-Point Operation per Second

FPGAs Field Programmable Gate Arrays

FPR Floating Point Register

GCC GNU Compiler Collection

GENEPI2 GEnerator of NEutron Pulsed and Intense

GPR General Purpose Register

GPU Graphic Processing Unit

GRP Guaranteed Resource Partition

HPC High Performance Computing

HWIFI Hardware-Implemented Fault Injection

IMA Integrated Modular Avionics

INRIA Institut National de Recherche en Informatique et en Automatique

IPAR Instruction Parity

JTAG Joint Test Action Group

LC Loop Counter

LE Loop Exit Address

LIG Laboratoire d'Informatique de Grenoble

LPSC Laboratoire de Physique Subatomique et Cosmologie

LS Loop Start Address

LR Link Register

MBU Multiple Bit Upset

MCE Machine Check Exception

MCU Multiple Cell Upset

MIMD Multiple Instructions Multiple Data

MM Matrix Multiplication

MPPA Multi Purpose Processing Array

MPPA IPC MPPA inter process communication

MTTF Mean Time To Failure

MWBF Mean Workload Between Failures

MultiPARTES Multi-cores Partitioning for Trusted Embedded Systems

NMR N-Modular Redundancy

NMR-MPar N-Modular Redundancy M Partitions

NoC Network-on-Chip

NP Non-deterministic Polynomial

OpenMP Open Multi-Processing

OS Operating System

PC Program Counter

PCIe Peripheral Component Interconnect

PE Processing Engine

PID Process IDentification number

PLR Process-Level Redundancy

PMC Performance Monitor Control

PMR Performance Monitor Register

PS Processing Status

pSWP parallel Software Partition

RA Return Address

RECOMP Reduced Certification Costs for Trusted Multi-core Platforms

RHBD Radiation Hardened By Design

RIS Robust Integrated Systems

RISC Reduced Instruction Set Computer

RM Resource Manager

RTEMS Real-Time Executive for Multiprocessor Systems

SDK Sofware Development Kit

SEB Single Event Burn-out

SECC Single ECC

SEE Single Event Effect

SEFI Single Event Failure Interrupt

SEGR Single Event Gate Rupture

SEL Single Event Latch-up

SENESCYT Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador

SER Soft Error Rate

SET Single Event Transient

SEU Single Event Upset

SFR System Function Register

SIFT Software Implemented Fault Tolerance

SMEM Static Memory

SMT Simultaneous multithreading

SMP Symmetric Multi-Processing

SOI Silicon-On-Insulator

SP Stack Pointer

SPC Shadow Program Counter

SPR Special Purpose Register

SPS Shadow Program Status

SRR0 Save/Restore Register 0

SRR1 Save/Restore Register 1

SSCA3 third Scalable Synthetic Compact Application

SWIFI Software-Implemented Fault Injection

TIMA Techniques de l'Informatique et de la Microeléctronique pour l'Architecture des systèmes integrés

TMR Triple Modular Redundancy

TSMC Taiwan Semiconductor Manufacturing Company

TSP Traveling Salesman Problem

TTY Tele TYpewriter

TVF Thread Vulnerability Factor

VHDL VHSIC Hardware Description Language

VLIW Very Long Instruction Word

VR Vector Register

V-BSP Virtual Board Supporting Package

XER Exception register

Chapter 1

Introduction

Nowadays, advances in integrated circuit technologies allow having processors with multiple cores which offer a very large computing capacity through the implementation of massive parallelism within an affordable power budget. These devices also provide a great flexibility because they allow implementing different multi-processing modes, programming paradigms and types of applications. In addition, their inherent redundancy capability makes them ideal for implementing fault-tolerant mechanisms. Consequently, most modern computing systems use multi-core and many-core processors as a standard solution to fulfill the increasing demand of performance and reliability without a critical increase of power consumption.

For instance, in High Performance Computing (HPC) systems, the first two SUPERCOMPUTERS Top500 (November 2016) are based on these devices: (1) *1er Top500 : Sunway TaihuLight - Sunway MPP*, based on the Sunway many-core processor SW26010 (260 cores) 1.45GHz , and (2) *2nd Top500 : Thiane-2*, based on the multi-core processor Intel Xeon E5-2692 (12 cores) 2.2GHz + Intel XeonPhi 31S1PL.

In addition, the use of multi-cores is widespread in embedded systems. However, when critical tasks are required only one core is used [MEA+10]. Therefore, international research projects [1] involving industrial and academic partners such as, ACROSS, ARTEMIS - EMC^2, CAPACITES, DECOS, MultiPARTES, are exploring the development of new solutions that allow the use of multi-processors in critical systems. Hence, avionics and spacecraft industries are interested in validating the use of these components for their applications (e.g. international teams are working on this behalf: CAST, EASA,

[1] For further information see Projects References section

RECOMP project, etc).

The trend to promote the use of general purpose or Commercial Off-The-Shelf (COTS) devices in critical systems is done mainly because: (1) the cost reduction by using standard components instead of custom-designed ones, and (2) the significant reduction in the ratio performance to power consumption by using commercial components instead of radiation hardened components [MSM+02, KSD+97].

Having more performing chips requires the improvement of circuit manufacturing process by shrinking the die size. Unfortunately, the degree of miniaturization makes these circuits potentially more sensitive to the effects of natural radiation. Its impact is aggravated by the chip complexity and the huge amount of memory cells that they contain. For this reason, physical designers are continuously searching for new methods to improve manufacturing technologies to reduce Single Event Effect (SEE) consequences. Additionally, manufacturers have implemented protection mechanisms such as parity and Error Correcting Code (ECC) in devices' memories. Besides, some mitigation techniques based on hardware and software approaches are proposed in the literature to minimize the consequences of this phenomenon.

In spite of the manufacturing efforts, there are some areas that remain vulnerable to the effects of natural radiation. Several interesting works dealing with the sensitivity of electronic components can be found in the literature. However, there are very few available works regarding multi-core and many-core processors sensitivity to the radiation effects. Therefore, it is mandatory to evaluate at what extent this sensitivity affects the reliability of applications running on such devices. Furthermore, due to the massive parallelism used for improving the performance of computing systems, the evaluation of the impact on parallel applications becomes essential.

Software fault tolerance for parallel and distributed systems has been largely studied. However, studies regarding systems-on-chip fault tolerance are limited. Taking in advantage of the multiplicity of cores of multi/many-core processors, it is suitable to implement redundancy techniques to improve the reliability of parallel applications.

In this context, the purpose of this thesis is, first, to evaluate the impact of SEEs on parallel application running on multi-core and many-core processors and, second, to propose an approach based on N- Modular Redundancy and partitioning to improve the reliability of parallel applications running on many-core processors.

1.1 Research issues

Considering the wide range of possible software-environment configurations, the evaluation was done by multiple case-studies:

A) A multi-core running each core independently of the others.

B) A multi-core sharing resources and using inter-core communications.

C) A multi-core implementing Triple Modular Redundancy (TMR) as fault-tolerant technique.

D) A many-core execution with minimal use of NoC services.

E) A many-core execution with intensive use of NoC services.

F) A many-core implementing a new proposed fault-tolerant approach on massive parallel applications.

The evaluation considers the use of Asymmetric Multi-Processing (AMP) and Symmetric Multi-Processing (SMP) modes, and two benchmark applications: the Traveling Salesman Problem and the Matrix Multiplication. It was carried-out by Software-Implemented Fault Injection (SWIFI) or by neutron radiation ground testing.

For the experimentation, two COTS devices were considered. The first one was the Freescale PowerPC P2041 -core processor, which was selected because of its manufacturing technology (SOI) and its capability to allow different multi-processing modes. The second one was the KALRAY MPPA many-core processor, which was selected due to its advanced CMOS 28nm manufacturing technology, its architecture, and the huge number of processing cores (256) that it integrates.

The main contributions of this research are: (1) the evaluation of SEE sensitivity of different parallel applications implemented on multi-core and many-core processors, (2) the evaluation of the impact of the programming model and multi-processing mode on the system's SEE sensitivity (3) the development and evaluation of an approach based on redundancy and partitioning called NMR-MPar to improve the reliability of parallel applications running on multi/many-core processors.

1.2 Context of the thesis

This thesis was carried out in the context of a collaboration of two research teams of the University of Grenoble-Alpes: (1) the Robust Integrated Systems (RIS) team of TIMA laboratory, and (2) the INRIA Compiler Optimizations and Runtime Systems (CORSE) team of LIG laboratory. The RIS team has vast experience concerning the reliability of circuits and systems, while the CORSE team has an extensive background on compiler optimization and run-time systems for multi-core and many-core processors. The synergy produced by the contribution of the know-how of both teams was fundamental in the development and success of this research.

In addition, during the development of this thesis I have collaborated with members of other international teams in the context of the project Stic-AMSUD Energy-aware Scheduling and Fault Tolerance Techniques for the Exascale Era (EnergySFE) regarding the development of parallel applications on multi/many-core processors.

Finally, this thesis was funded by two scholarships: (1) the Secretaría de Educación Superior, Ciencia, Tecnología e Innovación del Ecuador (SENESCYT) Program "Convocatoria Abierta 2012 Segunda fase" grant *752-2012*, and (2) by the Universidad de las Fuerzas Armadas ESPE grant *14-005-LCS-DOC-ESPE-a2*. It was also supported by the French authorities through the "Investissement d'Avenir" program - Calcul Parallèle pour Applications Critiques en Temps et Sureté (CAPACITES) project.

1.3 Thesis outline

The remaining of this document is organized as follows:

- *Chapter 2:* summarizes the concepts and principles needed for the understanding of this research regarding: (1) radiation environment and its effects on electronic circuits, (2) main characteristics of multi-core and many-core processors, and (3) reliability issues concerning mentioned devices.

- *Chapter 3:* describes the methodology and tools used to evaluate the impact of natural radiation on applications running on multi-core and many-core processors.

- *Chapters 4 and 5:* present the evaluation of the impact of neutron radiation on dif-

ferent scenarios implemented on the Freescale P2041 multi-core and the KALRAY MPPA-256 many-core processor respectively.

- *Chapter 6:* proposes a fault-tolerant software-approach to improve the reliability of applications running on multi-core and many-core processors. It also presents its evaluation through a case study implemented on the MPPA-256 many-core processor.

- *Chapter 7:* discuss the most relevant related work.

- *Chapter 8:* provides general conclusions and gives some future research perspectives.

Chapter 2

Background

This chapter describes the principal concepts needed for a correct understanding of this thesis. Three main topics are considered. The first one is related to radiation environment, its effects and consequences on electronic systems. The second one is dedicated to present the main characteristics of multi-core and many-core processors. The last one summarizes key concepts on reliability and some useful techniques for improving it. Finally, this chapter concludes with a discussion about the thesis subject.

2.1 Radiation Effects on Electronic circuits

The normal operation of electronic circuits and systems are affected by the environment where they operate. One of the most important factors is the radiation that decreases their reliability. Hence, it is essential to understand this phenomenon and its possible consequences.

2.1.1 Radiation Environment

Natural and artificial radiation affecting electronic devices are present in atmospheric and spatial environments. On the one hand, radiation in the space comes principally from cosmic rays, solar wind and flares, and the Van Allen radiation belts. Cosmic rays are high energy particles coming from the outer space including electrons, protons, heavy-ions and other subatomic particles. Solar wind is a flux of particles originated by the high temperatures of the solar corona which mainly consists in electrons, protons and alpha

particles. A Solar flare is the product of the sudden release of magnetic energy from the solar atmosphere. The radiation belts is a torus of charged and energetic particles around the Earth held by its magnetic field. Figure 2.1 illustrates the Earth's magnetosphere which is formed by the interaction of solar wind and Earth's magnetic field.

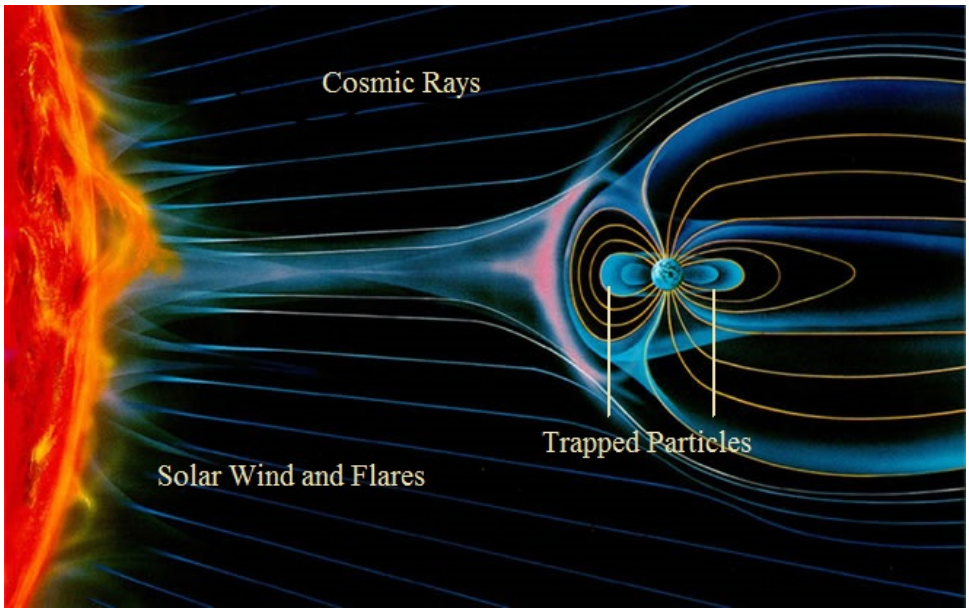

Figure 2.1: Earth's magnetosphere [NASA]

On the other hand, the atmospheric radiation environment is the result of the interaction of cosmic rays with the atoms of the Earth's atmosphere. Particles hit with a nucleus of the air generating a shower of new particles that increase while descending in the atmosphere. Figure 2.2 depicts the development of a cosmic-ray air shower.

Spatial and atmospheric radiation may produce transient, permanent and destructive effects in integrated circuits. This radiation effects can be classified in two categories: cumulative effects and Single Event Effects. Cumulative effects are the result of charges trapped in the oxide volume of a semiconductor which appear after a long period of radiation exposure. SEEs in turn, cause abrupt changes or transient behavior in circuits if the amount of collected charge at a junction exceeds a threshold [MFMR08]. From the sixties, there were observed several problems in space electronics. However, the first proofs of malfunctions in electronic devices on spacecrafts due to space radiation were reported in 1978 [MW79].

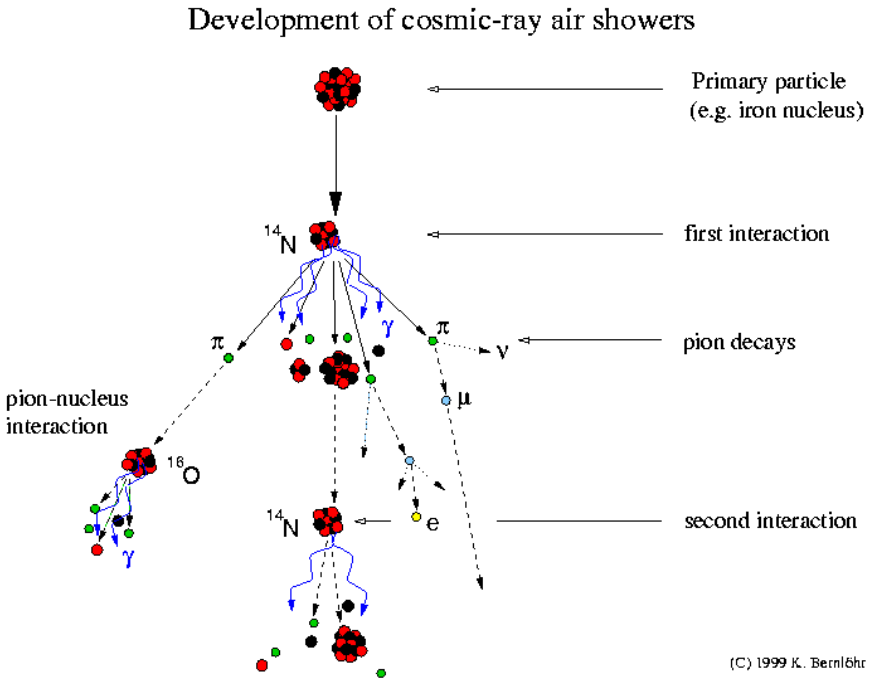

Figure 2.2: Cosmic-ray air showers [NASA]

2.1.2 Single Event Effects

SEEs are all the effects induced by the interaction of a single energetic particle with a semiconductor material. Typically, heavy-ions, protons and neutrons particles produce SEEs when colliding with electronic devices compromising their reliability [Nic11].

- *Single Event Transient (SET)*: An instantaneous voltage spike in a node (analog devices or combinatorial logic) of an integrated circuit produced by a single energetic particle strike.
- *Single Event Upset (SEU)*: the impact of a ionizing particle on a memory cell provoked the change of the bit content also called *bit-flip*.
  - *Multiple Bit Upset (MBU)*: When two or more bits are altered in the same word.
  - *Multiple Cell Upset (MCU)*: When a single event upsets different storage cells.
- *Single Event Failure Interrupt (SEFI)*: A malfunction state of a complex circuit which can be caused by an upset in a control bit or register, forcing the component

to be reset or reinitialised.

- *Single Event Latch-up (SEL)*: A destructive event related to a high increase of power supply current due to the activation of a parasitic PNPN structure induced by the impact of an ionizing particle.

- *Single Event Burn-out (SEB)*: A hard error affecting power MOSFET, IGBT or power BJT transistors by inducing a current flow that turns-on the parasitic NPN bipolar structure which leads to the destruction of the device.

- *Single Event Gate Rupture (SEGR)*: A hard error produced when the incident particle generates a conduction path in a gate oxide which leads to the component damage.

2.1.3 Consequences caused by transient effects

Transient effects may cause dysfunctions in electronic devices, performance degradation and even partial or total destruction of the device. These perturbations can affect directly a component of the device (clock line, memory cells, etc) or can be propagated to other parts of the circuit. Sometimes a SET can be propagated in the system generating a SEU or a SEFI [MFMR08].

This work deals with the consequences of bit-flips in memory cells (*"soft error"*). They can be classified as follows:

- *Silent fault*: there is no apparent effect on system, all the system and application processes are terminated in a normal way and the results are correct.

- *Application erroneous result*: the result of the application is not the expected one.

- *Timeout*: when the program does not respond after a duration equal to the nominal execution time.

- *Exception*: the application triggers an exception routine.

- *System crash*: the system stops working and it is necessary a restart.

- *Segmentation fault*: is a type of memory dysfunction error where the software (system and application) tries to access a non-valid memory address.

- *Abnormal process termination*: when the application is terminated in an abnormal manner either by an abnormal code return or by the Operating System (OS).
- *Unreliable condition*: when an application or system process is affected by the fault.

From the above list, the most critical one is the *"Application erroneous result"* because it is not detected by the system and can cause unpredictable consequences. Throughout the analysis of the results, in some cases the last five types were grouped as *Exceptions*.

2.1.4 Evaluation of the radiation effects on electronic circuits

Accelerated radiation ground tests and real-life tests are widely used to evaluate memory devices and processor-based architectures in terms of SEE cross-section. The cross-section provides the average number of particles needed to cause a bit-flip in a memory cell, and is defined as follows:

$$\sigma = \frac{Number\ of\ Upsets}{Fluence} \tag{2.1}$$

Where *fluence* is the integration of the flux of particles ϕ in the time.

Real-life tests

Real-life tests are the only and trustworthy way to study the effects of radiation on electronic circuits and to measure the soft-error rate of a device, since it is tested in the radiation environment where it is supposed to work (terrestrial atmosphere or space) [PVH09]. It consists in gathering as many devices as possible with the aim of increasing the number of SEEs to obtain valid statistics. The drawback of this evaluation strategy is the necessity of a huge number of devices and the long exposure time to obtain satisfactory results.

Radiation ground tests

There are many ways to characterize integrated circuits to radiation at ground level. Particle accelerators, laser beams and equipments based on fission decay sources are useful to obtain significant results in a short time. The more particles interact with the device,

the more SEE can be obtained. The drawbacks of this evaluation strategy are: particle beam spectrum is not really that of the natural radiation, there are few facilities around the world, high cost in experiment setup and tests[1].

- *Particle accelerator*: is a machine that uses electric fields to accelerate elementary particles such as heavy-ions, protons, electrons to very high energies producing a beam of charged particles. There are two main types of accelerators: linear accelerators, where particle travel along a straight beam line, and circular accelerators where a beam travels in a loop [Dot].

- *Laser beam*: is a narrow and coherent light beam produced by optical amplification that focus its energy in a single tiny spot with intense power [Eri17].

- *Fission sources*: The radiation emission is produced by a spontaneous fission of certain radioactive isotopes such as californium-252 or Cf-252, or by alpha particles impinging upon a low atomic weight isotope such as beryllium, carbon and oxygen [OLA96].

Fault Injection

Due to the high cost of accelerated testing and the long-lasting exposures in real-life tests, fault-injection technique has been rapidly adopted for evaluating the dependability of an electronic device or system. It accelerates the occurrence of faults allowing designers assessing and improving detection and protection mechanisms [HTI97]. Fault-injection campaigns in processor-based architectures are typically performed for simulating the consequences of SEUs at application level to estimate its SEE sensitivity. For devices intended to be used in safety-critical applications, the evaluation of the reliability of the application is a mandatory step.

In the literature, it can be found five categories of fault injection:

- *Hardware-Implemented Fault Injection (HWIFI)*: this technique allows injecting faults in the target system by means of a dedicated hardware platform. HWIFI includes several approaches being the most used the pin-level injection. Pin-level injection is based on stuck-at operations where pins are forced to a particular value

[1]Only as reference the cost of radiating heavy-ions per hour is around 650 USD.

to generate a soft error. Other approaches such as electromagnetic interference and power supply alteration are also used to produce soft errors [Ini11].

- *Software-Implemented Fault Injection*: is a fault injection technique used to emulate the effects of faults in an electronic device by means of software [Sch10]. Faults can be emulated in Central Process Unit (CPU) registers, Arithmetic Logic Unit (ALU) or main memory by producing bit-flips, and can be classified in compile-time and run-time injection. Other approaches corrupt the function call parameters.

- *Simulation-Based Fault Injection*: this technique use a fault-simulator for modeling and simulating the target device and the faults [KN14]. The simulation models are built using hardware description language such as VHDL.

- *Emulation-Based Fault Injection*: This technique is based on the use of Field Programmable Gate Arrays (FPGAs) with the aim of reducing the simulation time compared to simulation-based fault injection [KN14].

- *Hybrid Fault Injection*: this technique is based on the combination of two or more of the above techniques aiming at improving the fault injection time. It can benefit of the versatility of software fault injection and the accuracy of hardware measuring [ZAV04].

2.1.5 Mitigation of Radiation effects

The continuous technology scaling in integrated circuits makes them more sensitive to the effects of natural radiation [DM03]. For this reason, physical designers are continuously searching for new methods to improve manufacturing technologies to reduce SEE consequences.

In the literature there are different approaches that deal with hardening radiation components. These can be grouped as follows:

- *Hardening during conception and fabrication*: during the design stage, the manufacturers improve the fault tolerance capabilities of the devices by modifying the technological parameters, applying additional layouts, and changing the electrical construction.

- *Radiation Hardened By Design (RHBD)*: also called *hardening-by-system*, these techniques do not modify the fabrication process nor the electrical design. They mitigate errors by applying to COTS components some approaches including duplication in logic modules, implementation of error detection and correction circuits, memory cell interleaving, among others.

- *Shielding*: increases the reliability and useful life-time of devices that operate in radiation environments. For protecting against neutron and gamma radiation, heavy and large shielding blocks of lead and concrete can be used.

2.2 Multi-core and many-core processors

A consensus has been reached in the computing community: the only viable way to keep performance improvement rates within a given power budget is by building multi-core processors and exploiting massive paralleling [GPA+11]. Consequently, most of computing systems are nowadays using multi-core processors as a standard solution to fulfill the increasing demand of performance and reliability without a critical increase of power consumption. Moreover, multi-core and many-core processors can reduce the execution time of an application by performing parallel processing instead of increasing system frequency.

In this context, it would seem that increasing the number of cores per chip would be the most suitable solution to improve performance. Nevertheless, there are other factors affecting multi-core performance that should be analyzed, such as the latency produced by complex inter-core communications due to shared workloads, the scalability based on the increasing of workloads, the cache coherency among cores, and the memory and I/O bandwidth.

2.2.1 Architectural concept

A multi-core processor is an electronic circuit which integrates on a single chip multiple processor cores running in parallel. It can integrate multiple dies in the same package or several cores in a single die, Chip Multi-Processor (CMP). Figure 2.3 illustrates the processor evolution from the first conceptual model proposed by Von Neuman in 1945.

Typically, the Reduced Instruction Set Computer (RISC) is implemented on multi-core processors. There is an instruction-level paralleling implemented in the core architecture to increase the speed up of processing based on pipelined.

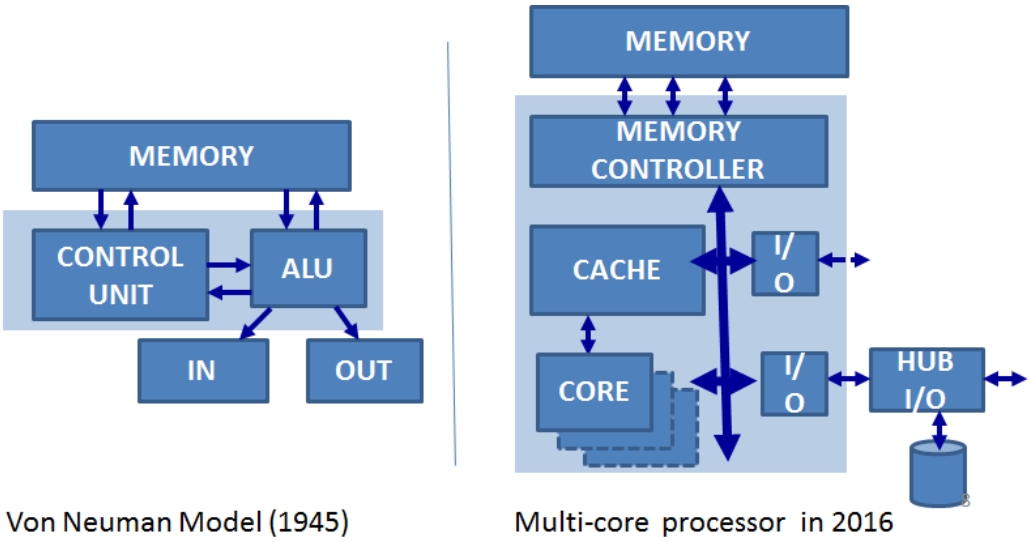

Figure 2.3: Evolution of processor architecture

Inside the chip, each core acts as an independent processor. The OS manages the internal resources and its scheduler assigns the processes to cores. Reference [Vaj11] details the architectural issues of both multi-core and many-core processors. Regarding the many-core processor, the large increase in the number of cores implies considerable evolution in the architecture of the device. The main constraints are related with the inter-core communications and the management of memory resources and I/O devices. Regarding the intercore communication mechanisms, traditionally in a multi-core processor each core communicates by a common shared bus with the other cores; however, in many-core processor the use of NoC is indispensable. On the other hand, for managing efficiently the memory resources some approaches are proposed such as ring, mesh and crossbar interconnections. Figure 2.4 illustrates a many-core processor architecture.

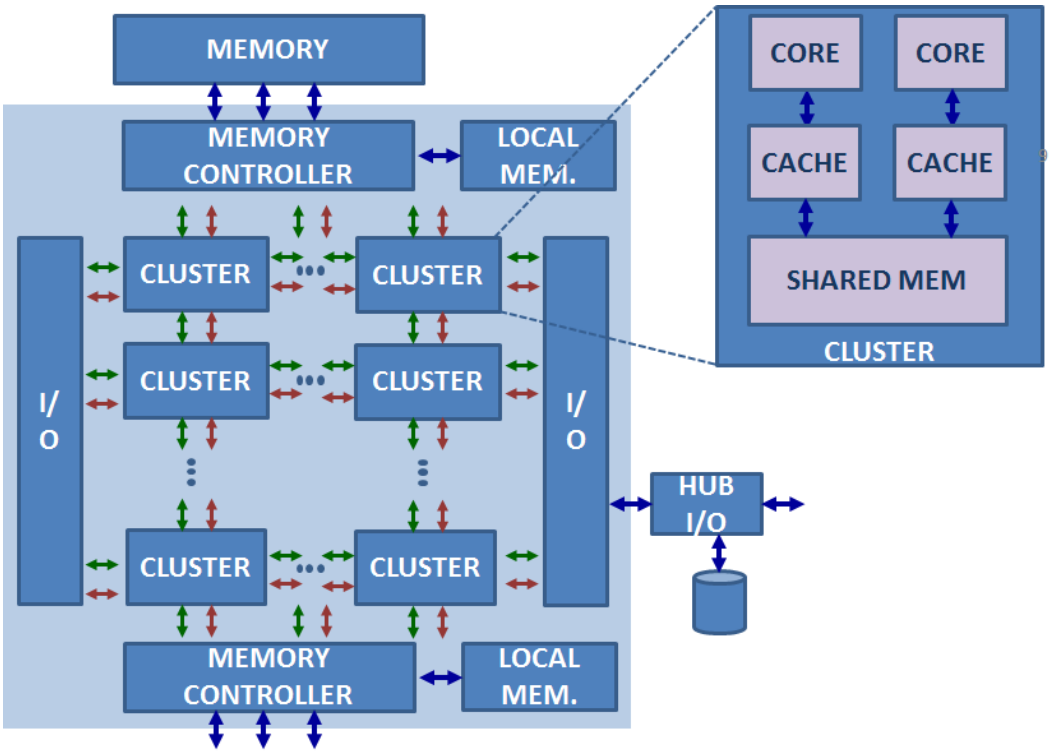

Figure 2.4: Many-core processor architecture

2.2.2 Memory hierarchy models

Multi-core and many-core processors include the use of cache memories as fast memories to reduce the memory access time by minimizing the access to main memory. Indeed, its use increases significantly the performance of the system, so several levels of caches are proposed. Figure 2.5 illustrates a typical hierarchy memory model used in these devices.

L1 caches memories are always private while L2 can be private or shared depending on the specific architecture device. The main advantages of private caches is its closer location to the processing unit which reduces access time. Also, its implementation minimizes contention. In contrast, by using shared cache, if few threads run on the device, the system performance could be improved due to the more cache space availability. On the other hand, by implementing private caches, the problem of keeping data consistent across cache memories arises. For solving this cache coherence problem, many solutions based on algorithms and coherence protocols are proposed.

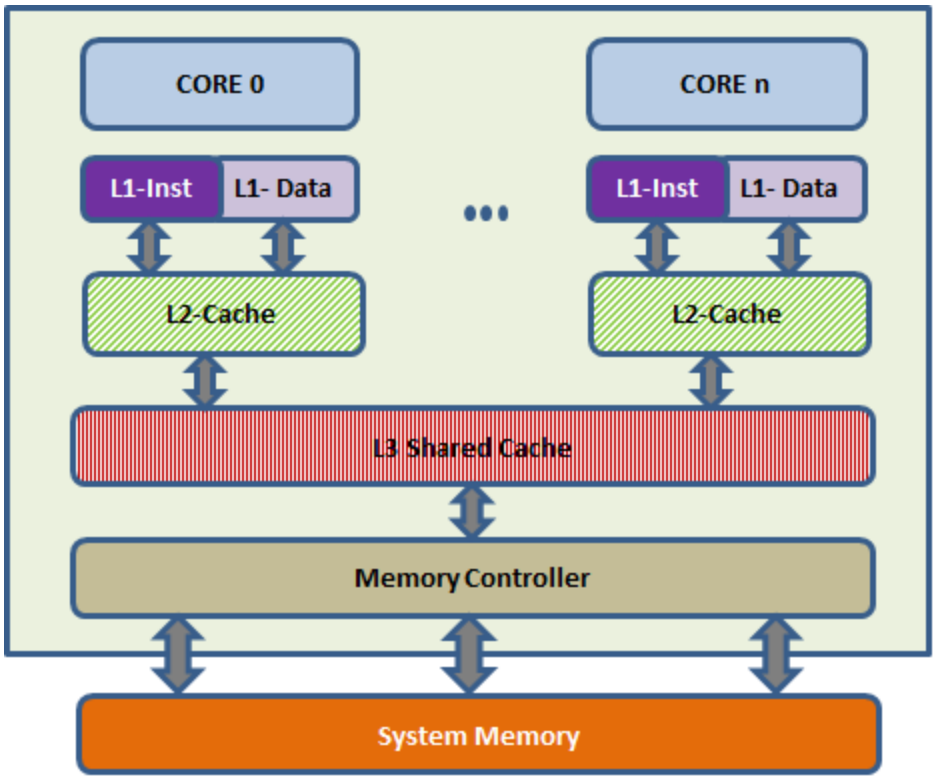

Figure 2.5: Memory architectural model in multi-core processors

Multi-core and many-core processors use shared and distributed memory models. In *shared* memory models, there is one common shared memory accessed by all processors while in *distributed* memory each processor or group of processors has its own local memory. Typically, multi-core processors use shared memory model and many-core processors use a mixed model. Some manufacturers group several cores in clusters. Inside each cluster, they implement a shared memory.

2.2.3 Software issues

To exploit massive paralleling, the application developers have to move from serial to parallel execution model and choose the appropriate system configuration to achieve maximum concurrency and consequently performance improvement. In addition, some capabilities need to be isolated to guarantee the dependability of the system. This implies that the software designer has to take into account issues such as multi-processing mode, programming model and the access level to hardware resources to better deploy the application.

Multi-processing modes

Regarding the multi-processing modes, there are two principal models: a) Symmetric Multi-Processing (SMP) and b) Asymmetric Multi-Processing (AMP). Figure 2.6 depicts these two Multi-Processing modes.

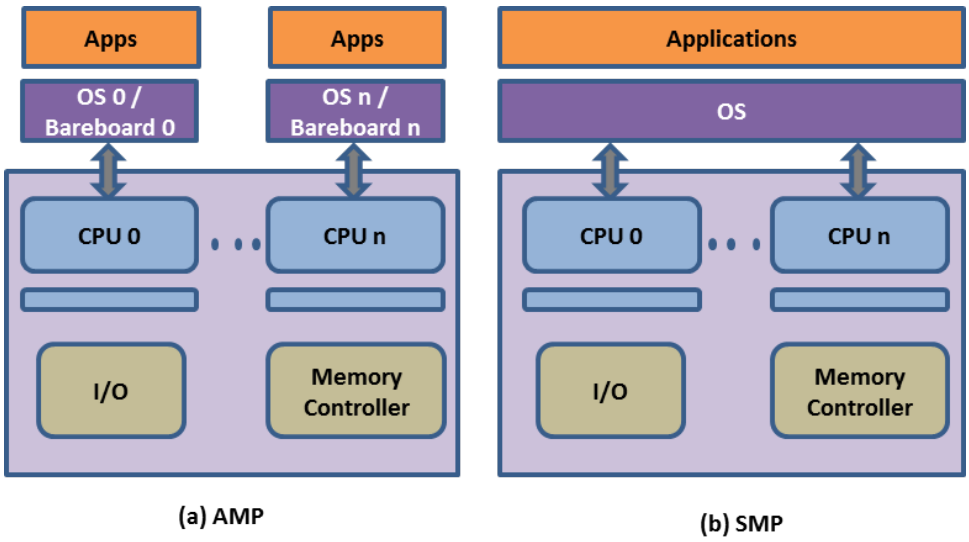

Figure 2.6: AMP and SMP modes

In AMP mode, the processor cores in the device are largely independent of each other. It can be configured with or without OS. Each core has its own private memory space, although there is a common infrastructure for inter-core communications. Hence, each core is managed by a separated OS image that exists in the main memory. Therefore, the OS can be different for each core which may be very useful when working with embedded systems. Typically, each software process is locked to a single core. Also, resources can be dedicated to critical tasks, resulting in more deterministic performance [Fre12].

On the contrary, in SMP mode a single OS manages all processors and schedules the processes. It is the simplest model. For the user, it seems that the program runs on a single core. In SMP mode, a single OS that runs on all the cores is responsible for achieving paralleling in the application. It dynamically distributes the tasks among the cores, manages the organization of task completion, and controls the shared resources.

The main advantages of these modes can be summarized as: (1) SMP provides greater scalability and paralleling than AMP, along with simpler shared resource management, (2) AMP is the only approach that works with two separate OSs, thus allows that resources can be dedicated to critical tasks resulting in more deterministic performance, and (3)

AMP often has higher performance than SMP since the cores spend less time handshaking one with another [Fre12].

Programming models

A programming model refers to the manner that the software assigns the application tasks to the hardware. Throughout the development of this research, the following programming models were used:

- *Bare-metal:* no OS is used, the programmer uses the Board Support Package (BSP) functions provided by the manufacturer to access hardware resources. There is no abstraction layer from hardware architecture. All the configurations and the distribution of the tasks must be programmed. The programmer has the control of each function.

- *Low Level:* no OS is used; however, there are a set of libraries that can be used. It provides a little abstraction from hardware architecture. The functions and commands used are closely related to the specific device capabilities on which the application is implemented.

- *POSIX:* is a low-level Application Programming Interface (API) defined on top of OS. It allows the control of parallel tasks (*threads*) where the programmer must control the management of threads. It is independent of the hardware architecture [IEE01].

- *OpenMP:* is a higher level of multi-threading API defined on top of POSIX which is easier and more portable. It consists on a group of compiler directives that can be executed on shared memory architectures, and its implementation is independent of hardware and the OS [Ope15].

Access levels to hardware resources

Multi-core and many-core processors provide different levels of protection that allow software access to hardware resources and configurations. These levels are also called *privilege modes*. The most common privilege modes are:

- *Hypervisor:* is the highest level and has access to all instructions and resources. This state provides partitioning and virtualization for OS software. It serves as a host of the possible OSs managing any shared resource and prevents from interfering one with the others.

- *Supervisor* or *Guest supervisor*: has a medium privilege level that concerns the OS which is a guest to the hypervisor. This state allows to perform several privileged operations that are not hypervisor privileged. When it attempts to access hypervisor privileged resources and instructions, an hypervisor interrupt is produced, so the hypervisor has to provide an appropriate emulation.

- *User:* has no privileges and commonly has access to the majority of instructions. In this state a program can run under the control of an OS guest to a hypervisor, of an OS running in bare-metal or under control of a hypervisor. These cases differ in the manner that some interrupts are managed when the user program tries to access a privileged resource.

2.2.4 Sensitivity to neutron radiation

The SEE sensitivity of multi/many-core devices can be evaluated through static and dynamic radiation tests. The first one is considered as the worst-case sensitivity of the device since all the accessible memory cells are monitored. The test consists in writing fixed patterns, and while the Device Under Test (DUT) is exposed to radiation, a program continuously reads the memory-cells to verify their content. If ECC is implemented in the device, the machine-check error-architecture reports Single ECCs (SECCs) and Double ECCs (DECCs) events. At the end of the test, the static cross-section (σ_{Static}) that characterizes the intrinsic sensitivity of the device's manufacturing technology is obtained.

On the side, the dynamic test consists in exposing to radiation the DUT while it executes the selected application to obtain the dynamic cross-section (σ_{Dyn}). This straightforward method described in [FVM+07] is performed to evaluate the behavior of an application implemented in processor-based devices.

Several works dealing with the sensitivity of electronic components to neutron radiation can be found in the literature. Reference [ND10] summarizes the sensitivity to SEEs

induced by neutrons of different integrated circuits (i.e., SRAMs, microprocessors and FPGAs) applicable to avionics. However, there are very few works available regarding multi-core and many-core processors sensitivity.

Some factors coming from emerging technologies increase this sensitivity to natural radiation: (1) the voltage scaling permits low energy particles produce SEE [Nor96], (2) the increase in size of cache memories produces a spline increase in soft error-rates [AVTK05, CSE+06], (3) the vulnerability on non-volatile memories [NAR+13].

Vulnerable zones

In a general way, every memory cell inside a device is a potential target of a SEE. However, the randomise in time (*when*) and space (*where*) of the bit-flip, and *how* the memory cell is used, affect the vulnerability of the system. The vulnerability to SEE is defined as the inability to resist the effects of radiation. In the context of this thesis the term SEEs refers to SEUs, MBUs, MCUs and SEFIs.

In order to modelling the vulnerability, in the literature there are several works that propose different metrics [MV16]. The Architectural Vulnerability Factor (AVF) proposed by [MWE+03] is one of the most used. The AVF is defined as the probability that a bit-flip in a component does not produce an error. Concerning the improvement of the reliability by reducing the system vulnerability, some approaches are also proposed in [MMM12, FZLF08].

The vulnerable zone of an application depends on the used resources and the exposure time of the data stored in memory cells. For a better comprehension of the problem of the vulnerability of an application, one could consider a simple example, such as a matrix addition, $C = A + B$ of size $n \times m$. The algorithm for this application is illustrated in figure 2.7.

```
for i ← 0 to n-1
do { for j ← 0 to m-1
     do C[i][j] = A[i][j] +B[i][j]
```

Figure 2.7: Algorithm for n x m matrix addition

In a bare-metal version running sequentially , the code section of this application is small and there are few registers (*i,j, accumulator*) used by the compiler for computing it. Therefore, the register-sensitive zone is minimal being the memory occupied by the

variables (*A,B,C*) the main sensitive zone. However, when the same application is executed on the same device but managed by an OS, the determination of the vulnerable zone becomes complicated due to the analysis difficulty of the functions and resources used by the kernel services and scheduling during the execution of the application. This determination becomes even more difficult when a parallel algorithm is applied and multi-threading is used due to factors such as synchronization between threads, data coherence, among others. Hence, authors of [OTKT12] propose a methodology for measuring the Thread Vulnerability Factor (TVF).

Moreover, if a multi-core or a many-core is used, additional issues should be considered:(1) the complexity of system configuration, such as distributed memory systems, (2) the use of many special registers at hardware level that are hidden to the programmer (3) the dynamic allocation of resources and processing cores used by the application.

Protection Mechanisms

The complexity of multicore architectures, due to the number of cores, concurrency issues, shared resources and interconnections among cores is a potential source of a wide range of errors. In order to facilitate error handling, the manufacturers have introduced the *error reporting architecture* by the *machine check error registers* to provide information of the possible sources of errors. A fault injection framework was proposed in [LPC+12] for supporting dependability analysis of multi-core systems, where machine-check-errors are emulated to analyze how the system responds to errors.

Reference [GPA+11] summarizes the most representative error detection and repair techniques that have been proposed in the literature for dependable multicore architectures. Mentioned work focused on dependable multicore-processor's architectures that integrate solutions for online error-detection, diagnosis, recovery, and repair during field operation.

Designers continuously improve the reliability of computing systems. Some techniques are proposed for microarchitectural components, such as processor registers, functional units, cache and main memories. Indeed, additional hardware implementations have been included at architecture level for improving their reliability. Examples of these protection mechanisms are the implementation of ECC and parity in memories. Hamming codes are very useful to mitigate SEUs since they can detect double errors and correct single ones.

Nevertheless, new hardware introduces an extra area with the corresponding increase in power consumption and performance degradation [AVTK05].

To reduce the overhead of the implementation of these mechanisms in cache memories, there are some approaches that proposes mixed mechanisms. For instance, in [Kim09] is proposed the use of ECC to only dirty cache lines [2] while clean lines are protected by parity. Also, authors of [LSI+06] propose partially protected caches by ECC and the use of selective data protection criteria. That means, critical functions use the protected area whereas the non-critical used the not protected one.

2.3 Reliability of multi-core and many-core

The impact of faults produced by SEE in the reliability of computing systems is worrying all domains since a single particle can produce system malfunction with important financial consequences. In reference [MV16] several examples of SEE consequences are referred.

The high demand of reliability for several applications implies that device manufacturers implement complex error-detection and correction circuits. Nevertheless, these implementations lead to an overhead that causes unpredictable slowdowns to the system [MV16]. For this reason, it is not possible to protect all the sensitive areas even if a physical protection is feasible. Furthermore, device vendors search to decrease the cost of designing and testing circuits for critical/embedded applications. Therefore, it is essential to improve the reliability of the system by using a fault-tolerance technique that minimizes the impact of its implementation cost.

2.3.1 Conceptual basis

Shooman in [Sho02] defines the reliability as the probability of no failure within a given operation time. It can be expressed in terms of the hazard function $z(t)$ which is also called *failure rate*.

[2] Dirty cache lines contain data that have been modified only at cache memory level and must be written back in main memory.

$$R(t) = e^{-\int z(t)dt} \tag{2.2}$$

The description of the failure rate function becomes complicated for complex systems with many components. Therefore, the Mean Time To Failure (MTTF) was introduced to simplify the interpretation of the system reliability. It is defined as the expected time to operate until a failure occurs. Considering a constant failure rate $z(t) = \lambda$, the reliability function $R(t)$ and the MTTF become the following expressions:

$$R(t) = e^{-\lambda t} \tag{2.3}$$

$$MTTF = \frac{1}{\lambda} \tag{2.4}$$

If failure rate increases with time $z(t) = kt$, the reliability and MTTF can be expressed as follows [Sho90]:

$$R(t) = e^{-kt^2/2} \tag{2.5}$$

$$MTTF = \sqrt{\frac{\pi}{2k}} \tag{2.6}$$

From [Vig], the behavior of the failure rate on semiconductors, typically considers that its value slightly decreases over early life, then stabilizes during its useful life, and increases outside the limit of useful life. Hence in semiconductors, the failure rate is considered as a constant λ in most of the cases. Additionally, there are other terms used to describe failure rate in semiconductors:

- *Failure In Time (FIT)* : Measure of failure rate in 10^9 device hours.
- *Confidence Level or Limit (CL)* : Upper confidence interval level of the estimation of the probability of a population failure rate.

Generally, when a particular radiation environment is known, the Soft Error Rate (SER) is expressed in terms of FIT value. It is computed as follows [Gai11]:

$$FITvalue = \sigma \times \phi \times 10^9 \tag{2.7}$$

Where σ is the cross-section of the device expressed in $cm^2/device$, and ϕ is the flux of neutrons n in the real environment expressed in $n/cm^2/h$. In FIT calculations, the flux at New York $\phi_{ref} = 14n/cm^2/h$ is considered as a reference.

In order to have an idea of the reliability of a system composed by various elements, a comparison of the system reliability under three different scenarios is illustrated in Figure 2.8. In all the cases the systems comprise the same element with a constant failure rate λ: (1) a system with a single element, (2) a system composed by two elements in series, and (3) a system with two elements in parallel.

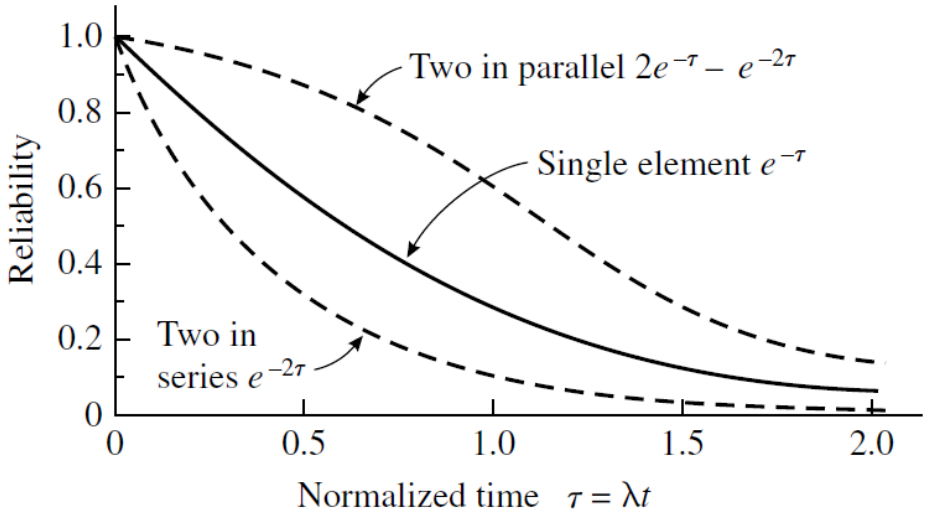

Figure 2.8: Comparison of reliability for different systems [Sho02]

From the above figure, one can see that the reliability is improved by the use of parallel systems. In contrast, the reliability decreases when the system uses two elements configured in series mode.

In the literature can be found several techniques used for modeling and improving the reliability of computing systems. Reference [MV16] presents an interesting survey of a considerable number of techniques. Techniques based on redundancy have been largely studied to improve the reliability of computing systems for both permanent and transient

faults. Furthermore, the partitioning concept is being applied to multi-core systems to guarantee the reliability in mixed-criticality systems.

2.3.2 Redundancy techniques

Redundancy allows a system to continue working properly in case of one or more components fail. This is possible due to the fact that parallel components added to the system provide an alternative path to operate. In general, the improvement in reliability can be achieved by *parallel* or *standby redundancy*. In the first case, the parallel components are powered-up and operating, while in the *standby* case some components are powered-down and they are turned-on when the main system fails. Regarding the nature of the replication, redundancy can be classified as:

- *Spatial Redundancy*: uses different physical components. It can separate identical data signals in space. The main advantage is that there is no inherent maximum operating frequency. The latter depends on the components characteristics. The main disadvantage is the overhead.

- *Temporal Redundancy*: uses the same physical components. It separates identical data signals in time. The maximum operating frequency depends on the technique implemented. The main advantage is the use of fewer components while the main disadvantage is the latency penalty.

- *Redundancy in data*: replicates the information. It stores the data in different memory spaces. The main advantage is the error detection and correction during the execution. The main disadvantage is the latency in memory access.

- *Redundancy in execution*: it replicates states machines. It performs copies of the same process. The main advantage is the transparency for the user. The main disadvantages are: (1) it deals with divergent causes on processes such as asynchronous signals and non-deterministic functions, and (2) the access to shared memory is a constraint.

Traditionally, the cost of implementing redundant systems has been significant. However in multi-core and many-core processors, various approaches based on redundancy

techniques can be considered due to the multiplicity of cores. Reference [HBR11] exploits several redundancy techniques to improve the reliability in multi-core processors. Its authors propose the use of redundancy in all the stages involving the device, from its design to the execution of applications.

There are several approaches that improve the reliability by the replication of contents of processor components during the execution of an application without modifying the hardware: caches [KS99, ZGKS03, Zha05, SIM07], instructions [SAL+08], and registers contents [MKO05, TS12]. In contrast, other redundancy techniques modify the hardware. Among them, the TMR [LV62] is a well-known fault-tolerant technique that implements triplication of modules and majority voting. This technique has been largely used to avoid errors in integrated circuits but imposes very high hardware overhead. Nevertheless, this method becomes more attractive when implemented on multi-core and many-core processor architectures.

N-Modular redundancy

A N-Modular Redundancy system consists of N identical parallel systems with the same input. It also includes a voter to obtain the system output. The voter compares the outputs of each parallel systems and applies majority criteria to select the response. In general N is an odd integer to simplify voting operation. However, it is possible to use an even integer depending on the characteristics of the system [Sho02].

Regarding the reliability of the system, it is important to note that N-modular redundancy is better to a single module only for $\lambda t < 0.69$. Figure 2.9 illustrates the reliability for diverse N-modular redundancy parallel-systems implementing $2n + 1$ modules. The curves consider a perfect voter with reliability $R_v = 1$.

It is clear that the reliability of a N-modular redundancy systems depends on the voter [KS79]. Therefore, considering the limitations of an imperfect voter, it is also possible to implement redundant voters. The most common implementations of this kind of systems are the Double Modular Redundancy (DMR) that allows error's detection, and the TMR that masks faults by detecting and correcting errors. Reference [Sho02] presents an extensive analysis of the reliability and availability of parallel TMR, standby TMR and repair TMR, and the corresponding voting systems.

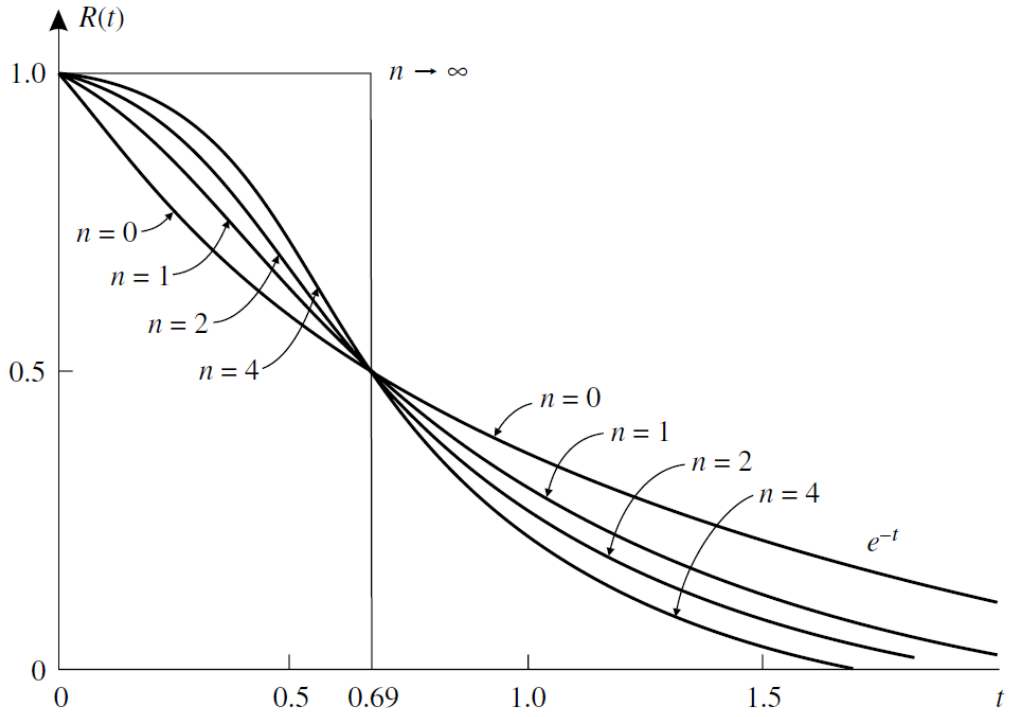

Figure 2.9: Reliability of NMR containing 2n+1 circuits [Sho02]

2.3.3 Partitioning

Mixed-criticality systems use partitioning to increase reliability of embedded systems [TCAP14]. For achieving fault tolerance by partitioning, the behavior of one partition must not be affected by other partitions. Hence, time and spatial partitioning must be satisfied.

- *Time partitioning*: the use of resources are exclusively assigned to a partition during a scheduled period of time. In a general manner, this is associated with the CPU time assigned to each partition.
- *Space partitioning*: a function can not change the code or data of another partition. This is associated with the memory space assigned to each partition.

Robust partitioning is a traditional concept used by Federal Aviation Administration (FAA) and European Aviation Safety Agency (EASA) to certify safety-critical avionics applications. It is defined in the standard RTCA/DO-297. However, according to [JFG+12], with the introduction of Integrated Modular Avionics (IMA) and multi-cores this concept must be clarified.

Space partitioning can be easily carried out by multi-core and many-core processors since the system can prevent invalid memory accesses while achieving spatial partitioning in such devices, which is not a trivial issue. Reference [LNT14] gives some directions concerning temporal partitioning on multi-core processors used for avionics.

2.4 Discussion

The use of multi-core and many-core processors allows improving the overall performance in terms of: (1) speed, performing parallel processing, (2) efficiency, choosing the appropriate mode of multi-processing and programming paradigm to achieve maximum concurrency, and (3) reliability, implementing fault tolerant applications through redundancy and partitioning.

Expensive dedicated devices combined with the high cost of the implementation of spatial and aircraft applications, makes suitable the use of COTS for their applications. However, in spite of the efforts done by the researchers to improve the security of applications running on COTS devices, it usage is limited due to the high reliability requirements. Regarding multi-cores, to the author knowledge, until now only dual cores are certified for being used in cockpit application in a mono-core configuration [Cer16].

Considering the multiplicity of cores and the use of redundancy techniques to improve reliability, it is important to determine the possibility of using multi/many-core COTS processors for running applications in harsh radiation environments. For that, it is necessary to study the degree of sensitivity to radiation of multi-core and many-core processors, evaluate how these effects impact the reliability of applications running on these devices, and improve the reliability of the system.

There are very few works in the literature that evaluates the SEE sensitivity of multi/many cores which are summarized in section 2.2.4. It is well known that the reliability of the application is intimately related to the reliability of the platform where it runs. However, there are other issues related to hardware and software configurations that may influence this sensitivity (e.g. the configuration of cache memories, management of interrupts, traps and exceptions). Moreover, if paralleling is performed, factors coming from the implementation of the multi-threading or distributed paradigms can also alter this sensitivity. Therefore, this thesis focuses on the evaluation of the SEE sensitivity of

applications running on multi-core and many-core processors considering different scenarios. Chapter 3 describes the methodology and tools used by this research to address this issue.

Finally, considering the current efforts for improving the reliability of COTS systems in order to use them in embedded and critical systems, the main goal of this thesis is to propose an approach that improves the system reliability profiting of its intrinsic redundancy capability and the partitioning concept. This approach was applied only at user level without involving the OS, aiming at overcoming the SEE consequences in the system.

Chapter 3

Experimental Methodology to evaluate the impact of SEEs Sensitivity

This chapter describes the methodology and tools used for assessing the impact of Single Event Effects on applications implemented in multi-core and many-core processors. This research proposes multiple case-studies applied to two different experimental platforms. The selection of the case-studies is done using a bottom-up strategy. The fault injection strategy as well as the radiation facility used to evaluate them are also detailed.

3.1 Generalities

It is well known that the reliability of a computing system depends on software, hardware and operator reliability and their interdependency [Sho02]. For the purpose of this thesis, the human operator reliability R_o was considered independently of the others. Moreover, due to human reliability is beyond the scope of this research, it was considered an ideal case where $R_o = 1$. Hence, only the dependency on software and hardware reliability is considered for assessing the impact of SEEs on the reliability of applications running in multi-core and many-core processors.

For accomplishing an adequate evaluation, it is important to take into account aspects such as hardware complexity and device configurations in terms of software environment. In this work, the problem is addressed using quantitative theory applied to multiple case-studies. The objective is to observe to what extent the dynamic chip response depends on the application, programming paradigm, multi-processing mode, and the usage of

resources. In order to design the experiments, the parameters to be considered are:

a) Hardware Parameters : (1) number of cores and resources, (2) manufacturing technology, and (3) inter-core communications

b) Software parameters: (1) type of application, (2) multi-processing mode, and (3) application levels and libraries.

For the experiments, two test platforms were selected: the first one is based on a multi-core processor and the second one is based on a many-core processor. AMP or SMP were adopted as multi-processing modes depending on the case-study.

The bottom-up strategy was used to select the case-studies. The first case, which is the simpler one, considers a multi-core processor running a bare-metal application where each core executes independently of the others. In contrast, the last case proposes a many-core processor running a redundant massive parallel application under POSIX. Six case-studies were evaluated:

A) The multi-core running each core independently of the others

B) The multi-core implementing TMR as a fault-tolerant technique

C) The multi-core sharing resources and maximizing inter-core communication

D) The many-core execution with minimal use of NoC services.

E) The many-core execution with intensive use of NoC services.

F) The many-core implementing a fault-tolerant approach on massive parallel applications

Throughout this thesis, fault-injection based on SWIFI and neutron radiation ground testing were used to evaluate the different cases. On the other side, for analyzing the experimental results, the consequences of a bit-flip in a memory cell were classified in: *silent faults*, *erroneous results*, *timeouts* and *exceptions*.

A mandatory step prior to the design of the test experiments, was to get familiar with the methods and tools required to perform experiments in the radiation facility. Hence, radiation experiments on SRAM memories were performed to asses their neutron radiation sensitivity. Some of the obtained results were presented in [CFV+15, CFV+16].

A significant consideration for the analysis of the results issued from radiation campaigns is the addition of uncertainty margins to the results, due to the scarcity of experimental data. For numerous events (typically >100), the Poisson distribution can be used to calculate such margins. However, for fewer events the most accurate and universal way to calculate the uncertainty margins consists in using the relationship between the cumulative distribution functions of the Poisson and chi-squared distributions as described in [AMRG14]. Therefore, the following equation has been applied:

$$\frac{1}{2}\chi^2(\frac{\alpha}{2}, 2N_{err}) < \mu < \frac{1}{2}\chi^2(1-\frac{\alpha}{2}, 2(N_{err}+1)) \tag{3.1}$$

where $\chi^2(p, n)$ is the quantile function of the chi-square distribution with n degrees of freedom, α is a parameter that defines the 100(1-α) percent confidence interval, and N_{err} is the number of errors detected. In general, this thesis uses a confidence level of 0.95 ($\alpha = 0.05$).

This study considers the evaluation of the reliability of a system only during its useful life. Hence, the failure rate function (λ) is assumed to be constant and the reliability function is characterized by the expression 2.3.

$$R(t) = e^{-\lambda t} \tag{2.3 revisited}$$

Where $\lambda = \sigma \times \phi$, being σ the cross-section of the system, and ϕ the flux of particles at the specific operation environment. For computing the reliability, the upper limit of the cross-section confidence-interval was used. In addition, it was considered that applications are intended to be used in avionics domain, being the neutron flux at 35000 *ft* of altitude ($\phi_{ev} = 2993.2n/cm^2/h$) with a system life time of 50000 h [1].

3.2 Evaluation strategies

Fault injection in processor-based architectures was a topic largely addressed by the scientific community to validate the reliability of critical applications. In this thesis, the Code Emulated Upset (CEU) approach principles have been adapted to multi-core and many-core processors. This approach based on interrupt signals, provides error-rate

[1] Commercial airlines fly over 10 km and the mean of economic life-time is 50000 fly hours

results close to those obtained in radiation tests, as demonstrated in [RVE+01, VFP10]. Its applicability to a complex processors as the PowerPC4748, allowed validating this device for aeronautical applications [PEP+08].

This research proposes to evaluate the different cases under neutron radiation. The radiation ground tests were conducted at the GEnerator of NEutron Pulsed and Intense (GENEPI2) facility located at the Laboratoire de Physique Subatomique et Cosmologie (LPSC) in Grenoble, France [VBR+14]. This accelerator was originally developed for nuclear physics experiments, and recently it has been used to irradiate integrated circuits from different technologies.

3.2.1 Fault-injection strategy

The CEU approach is summarized in equation (3.2). It combines the static cross-section of the device (σ_{Static}) with the error-rate issued from fault injection campaigns (τ_{Inj}) to estimate the error-rate (τ_{SEU}) of an application implemented in a processor.

$$\tau_{SEU} = \tau_{Inj} \times \sigma_{Static} \tag{3.2}$$

On one hand, the error-rate of an application is derived from SEU fault-injection campaigns. This quantity is defined as the average number of injected faults needed to produce an error in the result of the application. On the other hand, the static cross-section of the device (σ_{Static}) is obtained from radiation experiments.

$$\tau_{Inj} = \frac{Number\ of\ Errors}{Number\ of\ Injected\ Faults} \tag{3.3}$$

It is important to note that the objective of this approach is to reproduce, without intrusion, the effects of SEU faults. It was done by setting asynchronous interrupt signals. The execution of the interrupt handler produces the error in a randomly chosen target. In previous work, the interruption was produced by an external device. In the case of multi/many-core processors, it is possible to benefit of the multiplicity of cores for using one of them as a fault-injector while the others run the chosen application.

A compulsory prior step to the fault-injection campaign is to set the number of cycles needed to execute the application. It is done in order to know the range of time in which

the fault should be injected. While the application is running on the processing cores, the fault-injector selects in a random way the core, the memory-cell, the injection instant, and the bit to be changed. At the injection instant, if the memory-cell is accessible to the fault-injector, it changes directly its value, opposite case it sends an inter-processor interrupt to the selected core. The latter performs the corresponding bit-flip.

This kind of fault-injector is architectural dependent. The approach was adapted in each case-study depending on the software environment and the used development platform. The details of implementation are described in the concerned case-study.

3.2.2 Neutron-radiation facility

GENEPI2 is an electrostatic accelerator producing neutrons by impinging a deuteron beam onto a Tritium (T) target. After acceleration at 220 keV, deuterons (d) produce neutrons (n) by the fusion reaction $d + T \rightarrow n + {}^4He$.

From the target, neutrons are emitted in all directions. The DUT is set facing directly the target at a distance determined to adjust the neutron flux. While the DUT is fully exposed to neutrons, a dedicated neutron shielding can be used to protect the readout electronic platform as depicted in Figure 3.1.

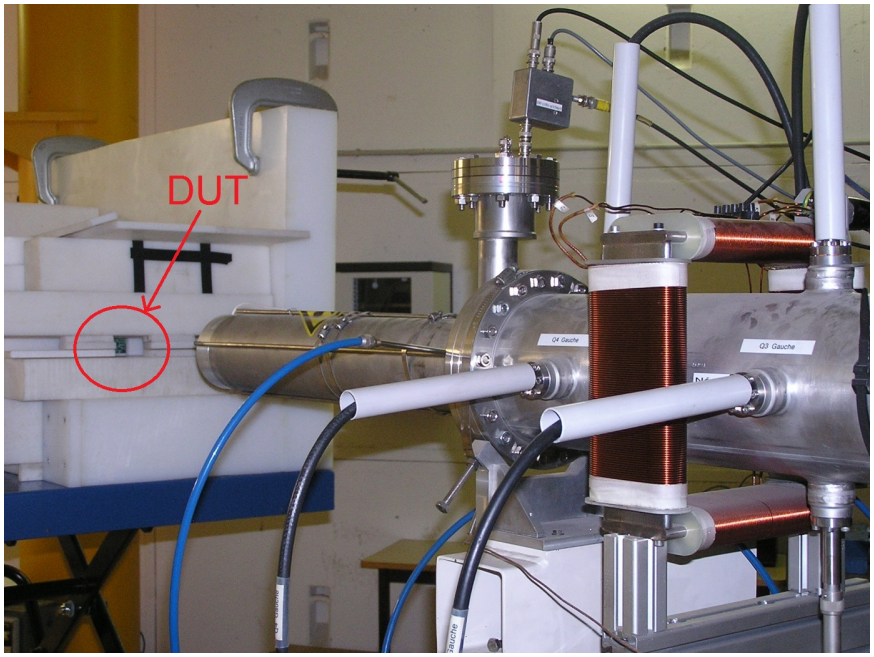

Figure 3.1: P2041 Experiment at GENEPI2 facility

Neutrons are produced with an average energy of 14 MeV. For the radiation campaigns

it was considered, to first approximation, that only neutrons emitted fully forward will impact the DUT. In this case, the neutron energy is maximal at 15 MeV. Reference [MWC+13] discusses the relevance of using 14 MeV neutron test to characterize the SEU sensitivity of digital devices.

Neutron production is monitored throughout the experiments to determine the neutron dose for each irradiation. An online Si detector, located within the beam pipe 1 meter upstream of the target, collects the recoil particles back-scattered from the target during the fusion reaction. Early 2015, a fresh T target was installed, generating a maximum neutron flux of $4.5 \times 10^7\ n \cdot cm^{-2} \cdot s^{-1}$.

3.3 Experimental Platforms

The multi-core experimental platform is the P2041RDB development board which based on the Freescale PowerPC P2041 quad-core. It was selected as the target device mainly due to:

1. The device is built in Silicon-On-Insulator (SOI) technology which has been proved to be less sensitive than CMOS bulk technology [GFCB+02].

2. In past years, the COTS Freescale *PowerPC4748* based on the same Power Architecture Technology was validated for aeronautical applications [PEP+08].

3. This multi-core processor allows different multi-processing modes: AMP, SMP, Bound Multi-Processing (BMP) [Fre12].

The MPPA Developer which is based on the KALRAY MPPA-256 many-core processor that was selected due to:

1. The manufacturing technology is advanced CMOS 28nm.

2. The architecture and the number of cores is similar to the many-core processor ShenWei SW26010 (260 cores). The latter is the base of the *Sunway TaihuLight Supercomputer*, which was ranked in the first position of the TOP500 list on June 2016. More details of the SW26010 can be found in reference [Don16].

3. The MPPA many-core was considered by semiconductor manufacturers and the real-time community for discussing the challenges of using many-core processors in embedded systems [SEU+15]. In addition, the project CAPACITES that gathers French academics and industrial partners uses this device to analyze the possibility of using many-cores for critical real-time embedded systems.

3.3.1 P2041RDB Development Board

The P2041RDB is a compact (micro-ATX) design board that features the quad-core P2041. It is designed for evaluating the P2041 in several kind of applications including aerospace and defense fields. This board comprises several IO lines comprising ethernet and Peripheral Component Interconnect (PCIe). The P2041RDB memory system supports 4 GB of DDR3. It is provided of NOR, I2C EEPROM and SPI memories.

The P2041RDB is delivered with an Embedded Linux Essentials kit. This kit includes a 2.6 SMP Linux kernel, *hugetlbfs*[2] for applications with a large memory footprint, user space Data Path Acceleration for high-performance packet handling, *u-boot*, the GCC tool chain and Mentor System Builder, among other features [Fre11].

3.3.2 MPPA Developer

The MPPA Developer is a development platform based on an Intel core I7 CPU operating at 3.6 GHz and running a Linux OS. The MPPA many-core is available within the Developer as an accelerator of the X86 Host CPU connected through 16 PCIe Gen3 lanes. In addition to the PCIe board, MPPA-256 Processor, the platform includes a PCIe board for debug and probe.

The MPPA Developer is delivered with a user environment and configuration containing Linux CentOS 7 x86 64 , Eclipse 4.3 and MPPA ACCESSCORE SDK v2.5 for developing, optimizing and evaluating applications. The latter includes three programming models for developing an application: POSIX, Kalray OpenCL and Lowlevel.

[2]Linux function to support the memory management for huge pages.

3.4 Benchmark applications

This thesis proposes the use of two types of parallel applications: a *CPU-bound* and a *memory bound*. In a *CPU-bound* application, the computation time is the bottleneck in the performance evaluation. On the contrary, in a *Memory-bound* application the execution time depends primarily on the time needed for accessing memory.

The selected CPU-bound application was the Traveling Salesman Problem (TSP), a Non-deterministic Polynomial (NP) hard problem very used for evaluating computing system optimization [ABCC07]. In [JP95], it is affirmed that "The TSP is probably the most important among hard problems. It has served as a testbed for almost every new algorithmic idea, and it was one of the first optimization problems conjectured to be "hard" in a specific technical sense". The TSP is used in problems of traffic routing, planning, logistic, manufacture among others.

On the other hand, the well-known Matrix Multiplication (MM) was chosen as memory-bound application. The MM is widely used for solving scientific problems related to linear algebra, such as systems of equations, calculus of structures, determinants among others. Also, the parallel version of Matrix Multiplication (MM) is one of the most fundamental problems in distributed and High Performance Computing (HPC).

3.4.1 Traveling Salesman Problem

This application aims at finding the shortest possible route to visit n cities, visiting each city exactly once and returning to the departure city. In a formal way, the problem is represented by a graph of the cities including the distance among them, where the cost $c(i, j) \geq 0$ represents the distance from city i to j. The goal is to find a Hamiltonian cycle with minimum cost for the travel.

To solve the problem there are several proposals. In this thesis, the implemented version by authors of [FCP+15] was used as a basis. They used a brute force exact algorithm based on a simple heuristic. The algorithm performs an in-depth search to find the shortest path. It does not explore the paths that are already know as longer than the current best path. This is done to improve performance. Each possibility of path is seen as a branch of the search tree. The pseudo-code for the sequential version of this algorithm is illustrated in figure 3.2.

```
global min_path
procedure TSP_SOLVE(last_city, current_cost, cities)
 if cities = ∅
   then return (current_cost)
 for each i ∈ cities
   do
 { new_cost ← current_cost + costs[last_city, i]
 { if new_cost < min_path
     then { new_min ← TSP_SOLVE(i, new_cost, cities\{i})
          { ATOMIC_UPDATE_IF_LESS(min_path, new_min)

main
 min_path ← ∞
 TSP_SOLVE(1, 0, {2, 3, ..., n_cities})
 output (min_path)
```

Figure 3.2: Pseudo-code for TSP sequential algorithm [FCP+15]

```
global queue, min_path
procedure GENERATE_TASKS(n_hops, last_city, current_cost, cities)
 if n_hops = max_hops
   then { task ← (last_city, current_cost, cities)
        { ENQUEUE_TASK(queue, task)
   else { for each i ∈ cities
        {   do { if last_city = none
        {      {   then last_cost ← 0
        {      {   else last_cost ← costs[last_city, i]
        {      { new_cost ← curr_cost + last_cost
        {      { GENERATE_TASKS(n_hops + 1, i, new_cost, cities\{i})

procedure DO_WORK()
 while queue ≠ ∅
   do { (last_city, current_cost, cities) ← ATOMIC_DEQUEUE(queue)
      { TSP_SOLVE(last_city, current_cost, cities)

main
 min_path ← ∞
 GENERATE_TASKS(0, none, 0, {1, 2, ..., n_cities})
 for i ← 1 to n_threads
   do SPAWN_THREAD(DO_WORK())
 WAIT_EVERY_CHILD_THREAD()
 output (min_path)
```

Figure 3.3: Pseudo-code for multi-threading TSP version [FCP+15]

For this work, two versions of the algorithm were used: the multi-threading and the distributed versions to be implemented in the multi-core and the many-core processors respectively.

Multi-threading algorithm

A multi-threading algorithm fills a queue of tasks. Each task is one of the branches of the search tree. Each thread takes tasks from the queue and executes the search. The minimum distance is a global variable shared by all the threads. The number of tasks is a function of the levels of the search tree and the number of cities. The pseudo-code for the multi-threading version of this code is illustrated in figure 3.3.

Distributed algorithm

Since the MPPA has a distributed memory, it is necessary to apply a distributed algorithm. In general, this one is very similar to the multi-threading one. Inside each Compute Cluster (CC), the algorithm runs as the multi-threading version. The main difference between the two versions occurs when a new minimum distance is found as the new value and the corresponding path is broadcast to the others CCs. The total number of tasks is the number of threads per CC times the number of CC used.

3.4.2 Matrix Multiplication

In this thesis a $n \times n$ matrix was used, being A and B the inputs and C the resulting matrix. Due to the data type is *single precision* (4 bytes per matrix element), each matrix occupies $4 \times n^2$ bytes. Hence, the size n of the matrix for each scenario was chosen to maximise the memory occupation. The classical sequential algorithm for this application is illustrated in figure 3.4.

```
for i ← 0 to n-1
do {  for j ← 0 to n-1
      do {  C[i][j] ← 0
            for k ← 0 to n-1
              do  C[i][j] += A[i][k] x B[k][j]
```

Figure 3.4: Algorithm for n x n matrix multiplication

The MM has some characteristics that makes it appropriate for being implemented in parallel architectures:

- Each element of the resulting matrix is computed independently of the others.

- The algorithm follows the Single Program - Multiple Data (SPMD) model.
- The number and type of operations to be carried out is independent of data.
- Regularity of the organization of data and operations performed on the data.

The parallelization of the algorithm has been extensively studied in [BDH+12]. There are many approaches proposed to optimize performance. In this work, the approach *divide and conquer* was used.

3.4.3 Intrinsic characteristic issues

The nature of the program and the parallel algorithm applied on each benchmark affects their vulnerability to radiation effects. In the TSP all the cores work together using some common data to find a unique result, while in MM each core works independently on a small part of the total result. This behavior implies a continuous information interchange between cores in the case of TSP. However, if one core stops functioning, the others can achieve a possible correct response. On the contrary, in the MM if one core stops, the correct response is not achieved. This fact can be considered as an intrinsic fault-tolerance characteristic of the TSP application.

The vulnerability of the $n \times n$ Matrix Multiplication application implemented on bare-metal can be easily derived from its algorithm. For instance, a bit flip affecting an input variable could produce an extensive propagation of errors in the results ranging from 1 to n being n the size of the matrix. Other critical variables are the counters used in the processing loops that could also produce exceptions.

Another important point in parallel and distributed algorithms for both applications is the distribution of the tasks. In the implemented TSP version the master dynamically changes the granularity of the jobs to obtain a good load balancing, while in the implemented MM the granularity is fixed. This fact affects also the use of resources for transferring data during execution.

3.5 Concluding Remarks

This chapter summarizes the methodology proposed by this research. One should consider that the fault-injection strategy based on SWIFI techniques can not cover the

sensitive zones non-writable by software means of the device. In addition, the effectiveness of this strategy strongly depends on the intrinsic characteristics of the target device. Thus, it was suitable that all case-studies were evaluated by neutron radiation ground testing. However, it was not possible because of the high cost and the availability of the radiation facility.

The case-studies implemented on the P2041 multi-core processor are detailed in chapter 4. Chapter 5 presents the first two cases concerning the MPPA many-core processor while chapter 6 described the last case-study.

Chapter 4

Case-studies based on the multi-core processor

This chapter describes the case-studies implemented on the quad-core P2041 built in 45nm SOI technology. At the beginning of the chapter some specific details of this device are given. Then, the three case-studies for evaluating the SEE sensitivity on applications running on this multi-core are presented. Each case summarizes the system configuration and the obtained results.

4.1 Description of the P2041 multi-core processor

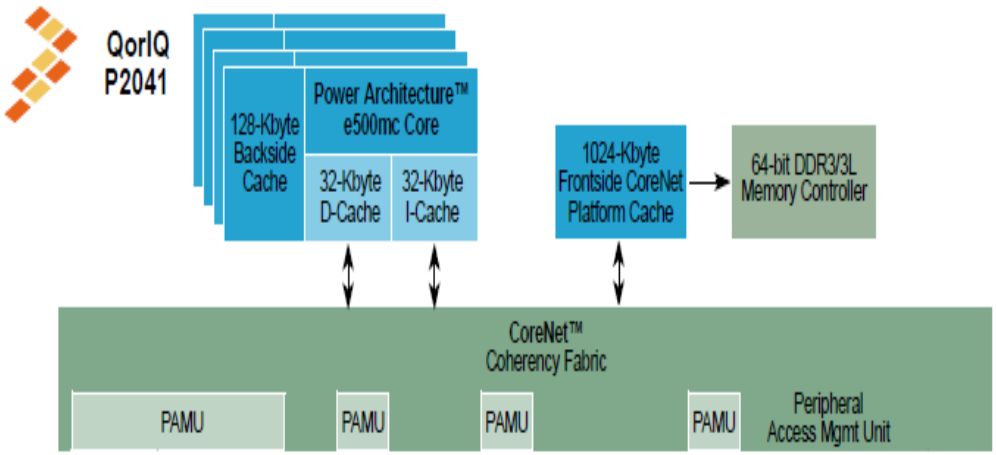

Figure 4.1: Memory hierarchy model for P2041 multi-core [Fre13b]

The targeted multi-core processor was the Freescale P2041 [Fre13b], which is based on four e500mc cores built on Power Architectures and manufactured in 45nm SOI technology. This quad-core can operate up to 1.5 GHz and includes a three-level cache hierarchy. Figure 4.1 depicts the memory architecture of the studied multi-core processor.

The e-500mc core is a 32-bit super-scalar processor with classic RISC architecture that includes independent on-chip 32 KB L1 caches for instruction and data, and a unified 128 KB backside L2 cache. Additionally, the P2041 includes a 1024 KB L3 cache shared between the four cores. The L1 instruction and data caches are implemented with automatic cache invalidation when a parity error is detected. Both, the L2 backside cache and the L3 shared front-side cache are protected with configurable ECC or parity for the data array, and parity for the tag array. This architecture corrects single-bit errors and detects double-bit ones [Fre13a]. Table 4.1 gives details about the sensitive areas of the multi-core processor that were targeted during the evaluation. They include L1, L2 and L3 cache memories and General Purpose Registers (GPRs), Floating Point Registers (FPRs) and Special Purpose Registers (SPRs).

Table 4.1: Sensitive areas of the P2041 targeted in this work

Sensitive zone	*Location*	*Capacity*	*Description*
L1	Cores 0, 1, 2, 3	32 KB / D and 32 KB / I per core	Data / Instruction Cache
L2	Cores 0, 1, 2, 3	128 KB per core	Backside Unified Cache
L3	Multi-core	1024 KB per chip	Frontside cache
GPR	Cores 0, 1, 2, 3	32 registers of 32 bits	General purpose register
FPR	Cores 0, 1, 2, 3	32 registers of 64 bits	Floating point register
SPR	Cores 0, 1, 2, 3	5 registers of 32/64 bits	Special purpose register - user mode access

This device provides three different levels of protection for software operations: *Hypervisor*, *Guest supervisor* and *User*. It implements a *machine-check error-report* architec-

ture that alerts the system through exceptions. A machine-check exception is produced when a hardware or software failure performs actions for which the hardware has not been designed to handle or cannot provide a suitable result.

4.2 Case-study A: Multi-core processor running each core independently

This case-study evaluates the dynamic response in terms of SEE sensitivity of the multi-core P2041 when the use of memory resources is the main issue. For that, the processor was configured to operate in AMP mode without OS and the MM memory - bound application was implemented. The evaluation of the system was done by radiation experiments. For the purpose of this thesis, this case-study was called *MM-AMP-P2041* scenario.

4.2.1 System configuration

For the experimental tests, the multi-core was set-up in bare-metal where each core performs independently of the others. Only at the start up configuration the core 0 is responsible to launch the other cores. The four cores were configured in *write shadow mode* where all modified data in the L1 cache is written through into the L2 cache. This ensures that, if data or parity tags are corrupted in the L1 cache, it can be invalidated and repopulated with the valid data from the rest of the memory hierarchy [Fre13a].

To log all the SEEs occurred during the radiation experiments, the machine-check error interrupt and the *Cache Error Checking* bits were enabled.

Benchmark details

Each core executes the same 80x80 Matrix Multiplication (MM) and compares its results with a predefined value in order to identify errors. The size of the matrix was selected in order to maintain a trade-off between the amount of memory used and the execution time. The matrices A, B and C were located in consecutive memory vectors. Matrix A was filled up with 1's and B was filled up with 2's, thus the expected result was 160 for all the elements of matrix C. The matrices were filled up with fixed values in

order to simplify the data analysis since a known value helps to identify which bit or bits have been changed during the test. In this way, MBUs and MCUs can be easily detected.

4.2.2 Experimental evaluation by neutron radiation

A radiation test campaign was carried out to obtain the dynamic cross section in MM-AMP scenario ($\sigma_{DYN\_MM\_AMP\_P2041}$). The device was set at a distance of $\sim 19.1\ cm$ from the target. The neutron energy was 14 Mev with a flux of $\sim 1.96 \times 10^5\ n \cdot cm^{-2} \cdot s^{-1}$ at a 500 Hz frequency. Two tests, each one lasting 2 hours were performed.

Table 4.2: Results of radiation experiments for the P2041 working in AMP scenario

SEE Type	*Type of error*	*Test 1*	*Test 2*	*Consequences*
SEFI	Load Instruction	1	0	Hang
SEU	L1 Data parity	19	17	None
SEU	L2 Single-bit ECC	9	20	None
SEFI	L2 Tag parity	0	4	Hang
SEU		3	1	None
SEU	Multiple L2 errors	3	1	None
SEU	L3 Single-bit ECC	3	2	None
SEFI	Instruction fetch	0	1	Hang
MBU	Other errors	6	0	Erroneous result
Total		44	46	

Table 4.2 shows the events produced in L1, L2 and L3 caches. The *Load Instruction* and *Instruction fetch* errors are the most critical ones since they produced processor hang. Half of the observed L2 *Tag parity* errors lead to processor hang. L1 *Data cache parity* errors are not critical since L1 cache is invalidated when parity fails. Finally, L2 and L3

Single-bit errors are not critical as the ECC corrects them.

From the results observed in Table 4.2, only 12 of the 90 detected events produced errors: (1) one SEFI in test 1 and five SEFIs in test 2, (2) six application erroneous results in test 1 that were not detected by the multi-core *machine-check error report*. These errors that produce *application erroneous results and hangs* were considered to calculate the dynamic cross section with a total fluence of $2.82 \times 10^9 n \cdot cm^{-2}$.

As it was presented in [RVB+15, RVB+16], the application erroneous results were produced by a MBU modifying different address tags of the device. These perturbations in the cache memories were not detected since the parity remains the same when two bits change in the same word.

Applying Equations (2.1) and (3.1) for a confidence level of 0.95 ($\alpha = 0.05$), the lower and upper limits for the dynamic cross-section in MM-AMP-P2041 scenario without OS are:

$$2.17 \times 10^{-9} \frac{cm^2}{device} < \sigma_{DYN\_MM\_AMP\_P2041} < 7.33 \times 10^{-9} \frac{cm^2}{device} \tag{4.1}$$

4.3 Case-study B: Multi-core implementing TMR as a fault-tolerant technique

In the previous case-study, it was shown that hardware protection mechanisms are not enough to protect independent parallel application from erroneous results. For this reason, the present case-study implements a state-of-the-art fault-tolerant architecture: the Triple Modular Redundancy (TMR) benefiting of the multiplicity of cores of the processor. In this case, the system was evaluated through fault injection where three cores execute in parallel the same task while the other plays the role of fault injector.

This case-study considers two scenarios: fault injection in memory variables and fault injection in registers. Fault injection in memory variables was considered due to the fact that during radiation experiments there were observed errors in the cache memories in spite of the implemented ECC and parity. In contrast, fault injection in registers was considered since these memory cells do not have any protection mechanism. These results were also presented in [VRM+14].

4.3.1 System configuration

Similarly with the previous case-study, the multi-core processor was configured in AMP mode in bare-meta. The synchronization between the cores is guaranteed for the main core using a Master-Slave scheme. It is thus, necessary to have a shared memory location for inter-processor communications [Fre12]. The slave cores perform the redundant execution while master core votes, saves correct data and reports errors.

Benchmark Details

The selected benchmark algorithm to be tested via the proposed fault injection method was a 40×40MM. The implementation of the algorithm minimizes the use of internal variables to manage the loops and store some intermediate results. The total number of variables used for the implementation of the 40x40 matrix multiplication is 4803, distributed as follows: 3200 input variables, 1600 output variables and 3 indexes for loop operations. The nominal duration of the executed program was 95605 clock cycles.

4.3.2 Fault-injector details

The experiment begins when the main core initializes data that is going to be used by other cores. After that, it sends a message through an inter-processor interrupt, to start the execution of the application in the slave cores. Once the message is received by the slaves, they confirm the reception and continue with the execution of the application. While the application is running on the slave cores, the main core performs the fault injection.

When a slave core finishes its task, it sends a message to master core indicating that execution was completed. The master core waits until the three messages arrive and then compares the obtained results from each core and selects the final result based on a majority vote.

In order to allow fault injection in variables while executing the studied application, it was required to place, by programming means, all the variables used by the application in a shared global array [MRAV14]. For this case, that array was physically allocated in an external DDR3 memory of the target board. This strategy permits that all variables are shared by all the processor cores. Thus, the fault injector core is able to modify the

content of the targeted address.

For performing fault injection in processor registers, the fault injector interrupts the other cores via the set of instructions, since it does not have access to their registers. An interruption handler is launched in order to perform the fault injection in the selected core. Note that this strategy can be applied to any tested application.

4.3.3 Experimental evaluation by fault injection

Two fault injection campaigns were performed to validate the robustness of the software TMR technique against SEU and to identify potential weaknesses. The first one targeted the 37 processor registers which are accessible by software. The registers to be tested are 32-bits sized, and thus the target sensitive area is about 1Kbits.

The second fault injection campaign is related to fault injection in variables. Each variable is implemented in 32 bits, thus the targeted sensitive area for this campaign was about 150 Kbits.

Fault injection in processor registers

The target registers to be perturbed by this campaign were the following: 32 general purpose registers numbered from 0 to 31, and 5 of the special purpose registers that may cause critical failures in program execution: Save/Restore Register 0 (SRR0) (that contains the next instruction to execute), Save/Restore Register 1 (SRR1) (that contains the machine state register), Link Register (LR), Condition Register (CR), and Exception register (XER).

Table 4.3 shows the results of the first test campaign that targets the processor registers where 500 faults were injected due to the criticality of certain registers in which faults may result in synchronization loss or exceptions.

From these results, it was calculated the application error-rate of the registers applying (3.3), and considering as errors the erroneous results, timeouts and exceptions.

$$\tau_{Inj\_Reg\_P2041} = \frac{Nb\ of\ Errors}{Nb\ of\ Fault\ Inj} = \frac{63}{500} = 12.6\%$$

In addition, the results in table 4.3 show that 25 errors were corrected by the TMR, and thus the total error rate was reduced to 7.6%. It is important to note that ma-

Table 4.3: Results of fault injection at register level for TMR MM AMP

Targeted Registers	*Runs*	*Silent faults*	*Erroneous results*	*Time outs*	*Exceptions*	*Errors corrected by TMR*
GPR 0-7,9,10,12	145	128	17	0	0	16
GPR 8	19	12	4	0	3	4
GPR 11	16	9	4	0	3	4
GPR 13-31	235	235	0	0	0	0
SRR1	26	18	0	8	0	0
SRR0	27	3	1	16	7	1
LC/CRF /XER	32	32	0	0	0	0
Total	500	437	26	24	13	25

nipulating certain processor registers become critical because it produces a high rate of timeouts and exceptions. Hence, the performance of the whole system is dramatically affected. Nevertheless, the occurrence of a fault in such critical processor registers has a low probability since the corresponding physical area is very small: about 0.6% of the area occupied by program variables and data.

The most critical register is SRR0. This register saves the context of the program before the interruption subroutine. 59% of injected faults in the SRR0 register lead to a timeout and 26% produce exceptions. This can be explained as the program counter contains the address of the next instruction to be executed after the interruption, and perturbing its content will cause a loss of sequence. Another critical register is SRR1. It stores the content of the machine state register when an interruption occurs. A fault injected in this register may produce a timeout when processor state is performing an address translation for instruction and data memory access. On the other hand, the experiment proves that for this application, bit flips injected in special registers LR, CR and XER have no incidence in the program execution. Regarding GPRs, they present different types of results depending on how often are used by the compiler for this application.

Fault injection in global variables

The experiment considers the emulation of one or two bit flips per execution. When two bit-flips are injected, the bit flip are produced in space memories corresponding to different variables from different cores. It emulates the behavior occurred in the radiation experiments described in section 4.2.2 when an application erroneous result was produced by the same particle affecting the tag of the caches of two different cores.

Table 4.4 shows a general overview of the second test campaign where 50000 faults were injected. As shown in Table 4.4, the standard MM algorithm is sensitive to SEU errors targeting the variables of the program. However, results show that approximately 20% of injected faults were silent.

Table 4.4: Results of fault injection in variables for TMR MM AMP

Bit-flip per run	*Runs*	*Silent fault*	*Erroneous results*	*Time outs*	*Exceptions*	*Errors corrected by TMR*
1	43104	9288	33814	1	1	33812
2	6896	455	6439	1	1	5503

When one SEU was injected per execution, the obtained error rate was about 78%. However, thanks to the TMR implementation, the 99.99% of errors were corrected. There were just two errors that could not be corrected by TMR; the errors caused by faults injected in an index variable during the loop execution. On the other hand, when two bit-flips were injected, the error rate reached 93% while the error correction factor decreased to 85%. It is possible to determine the silent faults thanks of the *Machine-check-error report architecture* that allows logging the detected ECC and parity errors.

Table 4.5 summarizes the calculated error-rates of program variables with one and two bit-flips injections by applying equation (3.3), and considering as errors the erroneous results, timeouts and exceptions.

These results demonstrate that by applying the TMR, the probability of having errors in the application is significantly reduced. Even in the case where 2 bit-flips were injected, there is a significant reduction of the error rate. Note that the probability of occurrence of this phenomenon is very low.

Table 4.5: Injection Error-Rates in Variables for TMR MM AMP

Bit-flip per run	τ_{Inj_var}	$\tau_{Inj\_var\_TMR}$
1	78.45%	< 0.01 %
2	93.40 %	13.60%

However, for this specific scenario, there are two cases in which the implemented TMR cannot provide correct results:

- When fault injections are performed in the first input matrix of one core, and in the second input matrix of another core. This will produce an error propagation in one row and one column of the output matrix of each core respectively.
- When the fault injections are performed in the same corresponding input-matrix row or column for the two cores.

Both fault injection campaigns in registers and variables provide a good feedback about the error detection and correction capabilities of the TMR algorithm. Results show that the effectiveness of implementing TMR to improve the reliability of an application depends on the affected zone and the multiplicity of the bit-flips. Also, it is possible to verify that its efficiency is reduced when the SPR registers are touched, since they are more liable to produce *timeouts* and *exceptions* that can not be recovered. Reference [Sho02] deeply analyzes the reliability of this redundancy technique.

4.4 Case-study C: Multi-core sharing resources and maximizing inter-core communication

This case-study evaluates the dynamic response of the multi-core P2041 when processor cores share resources. For that, both applications the TSP CPU-bound and the MM memory-bound were implemented when the processor operates in SMP. The evaluation of the system was done by radiation experiments and fault injection. Then, four scenarios were analyzed:

- The TSP evaluated by fault injection: *TSP-SMP-P2041-FI*

- The MM evaluated by fault injection: *MM-SMP-P2041-FI*
- The TSP evaluated by neutron radiation: *TSP-SMP-P2041-NR*
- The MM evaluated by neutron radiation: *MM-SMP-P2041-NR*

4.4.1 System configuration

The multi-core processor was configured in SMP mode in which the OS manages the resources in order to maximize the processing capacity of the cores. The OS was the embedded Linux SDK V1.6. For the purpose of this work, it was necessary to modify the original *traps* code in the kernel and in the u-boot of the Linux OS. The *traps* code is the code that is executed by the system when an exception or a fault occurs. In this approach, the *traps* code was modified to log all the events detected by the *machine-check error report*. Additionally, when detecting L2 cache errors, the L2 *error registers* values were logged to obtain more details about the event. It is important to note that, the original *traps* code logs the recoverable and unrecoverable conditions that cause a machine-check exception. If the condition is recoverable by the machine check, then it returns to the previous state and its operation is resumed.

For the experimental tests, all cores were configured in *write shadow mode*. To log all the SEEs occurred during the experiments, the machine-check error interrupt and the *Cache Error Checking* bits were enabled.

Benchmark details

The TSP CPU-bound application was implemented to maximize the use of CPU resources and scheduling. In contrast, the MM was implemented to maximise the memory occupation. Both executions were distributed among the cores.

The parallel versions of the studied applications were implemented by using the *pthread* library in the TSP case, and the *OMP*library for the MM. The implemented version of the *TSP* application makes this benchmark intrinsically fault-tolerant since if one core is stopped by any reason, another core could find the correct result.

4.4.2 Fault-injector details

In this case-study, an application based on Linux *PTRACE-Process trace* functions was implemented to monitor and/or inject faults on multithreading applications. The fault injector can inject SEU faults in GPRs, selected SPRs and in memory regions used by the application under test. It uses the *fork* principle, where the parent process is the fault injector and the child process executes the target application.

In order to inject faults, *PTRACE* functions were used to access the memory and registers of the child process. Using this method, it is not necessary to have the source code of the tested application since it works directly on the executable file.

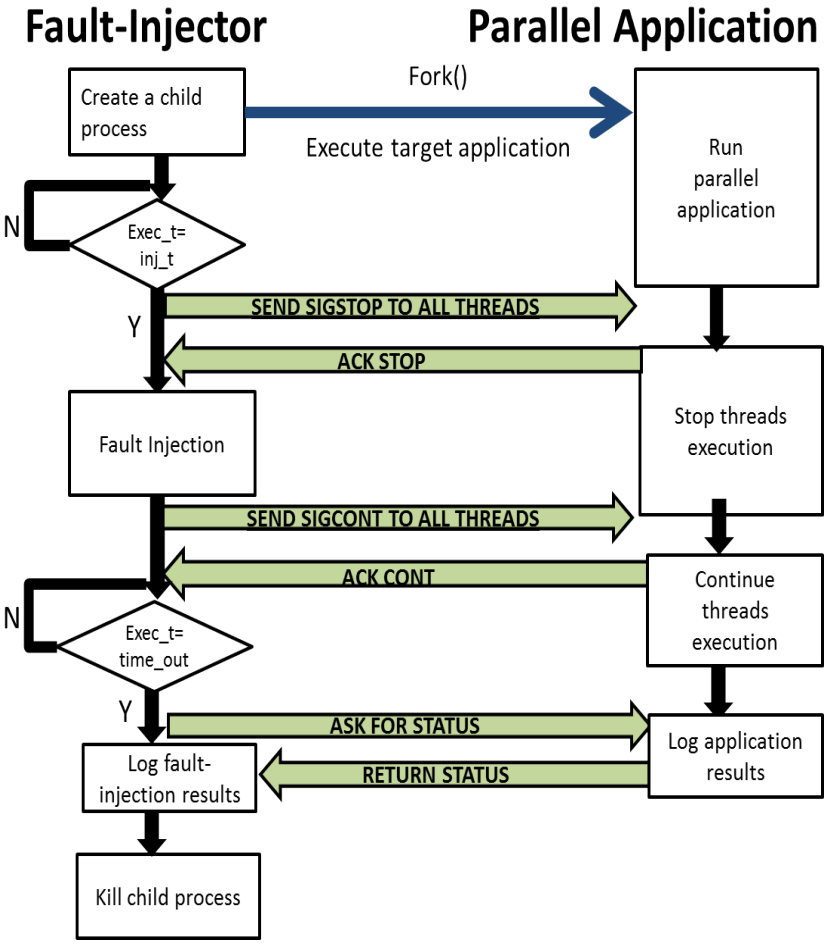

Figure 4.2: Proposed software fault-injector by fork principle

Since the multi-core processor is configured in SMP mode, instead of selecting a target core, the fault injector randomly selects a target thread. This experiment only considers the injection of one SEU per execution, even if the proposed approach allows injecting multiple SEUs. Figure 4.2 illustrates the proposed fault injection mechanism.

The fault injector creates a child process that executes the application. The child process starts its execution, and at the selected instant the fault injector reads the information about the number of tasks and the corresponding Process IDentification number (PID) of each thread associated with the child process. Then, it stops all the child processes threads via the SIGSTOP signal. After that, it injects the fault using specific PTRACE functions. Concerning SEUs injected in memory locations, the fault injector reads the memory regions used by the child process from the PID maps file. It randomly selects the memory region and the memory address where the fault should be injected.

Once the fault injection is accomplished, the fault injector sends SIGCONT signals to all threads to resume the operation. Then, it waits until the timeout parameter is reached. Afterwards, it checks the status of each PID thread application and analyzes the consequences of the injected fault based on the exit status. Finally, to avoid any "zombie" process, the fault injector kills all the child process.

4.4.3 Experimental evaluation by fault injection

This experiment comprises a TSP problem with 15 cities and a 80×80 MM. In Table 4.6, it can be seen that in TSP the code section predominates over data section, while in MM the data section predominates. This work was presented in [VRV+15]. Two fault injection campaigns per target application were performed on a quad-core processor: (1) Fault injection in processor GPRs and selected SPRs, (2) fault injection in memory regions of the tested application. Table 4.7 provides the parameters of the two fault injection campaigns.

Table 4.6: Applications summary for TMR scenatio

Target	*DATA*			*CODE*	
	Input Variables	*Output Variables*	*Internal Variables*	*Work Threads*	*Tasks*
TSP	260	17	88	4	2200 approx.
MM	12800	6400	12	4	4

Table 4.7: Fault-injection campaigns' details for TMR scenario

Target	*Standard exec. time [ms]*	*Exec time with fault injection*		*Runs per campaign*
		Register Campaign[ms]	*Memory Campaign [ms]*	
TSP	2977.8	3287.6	3472.4	7500
MM	21.0	23.2	24.8	50000

Fault injection in processor registers

These fault injection campaigns target 44 Processor Registers which are accessible by software: 32 GPRs and 12 SPRs. It is important to note that in this work the Program Counter (PC) and the Stack Pointer (SP) registers were not considered as fault injection targets since they are not accessible by software means. The considered registers are 32-bits sized, and thus the target sensitive area is about 1.4 Kbits. The results of this campaign are summarized in Table 4.8.

Table 4.8: Results of fault injection in registers for SMP scenarios

Application	*Bit flips injected*	*Silent faults*	*Erroneous results*	*Timeouts*	*Exceptions*
TSP	7500	4904	12	20	2564
MM	50000	42191	735	6758	316

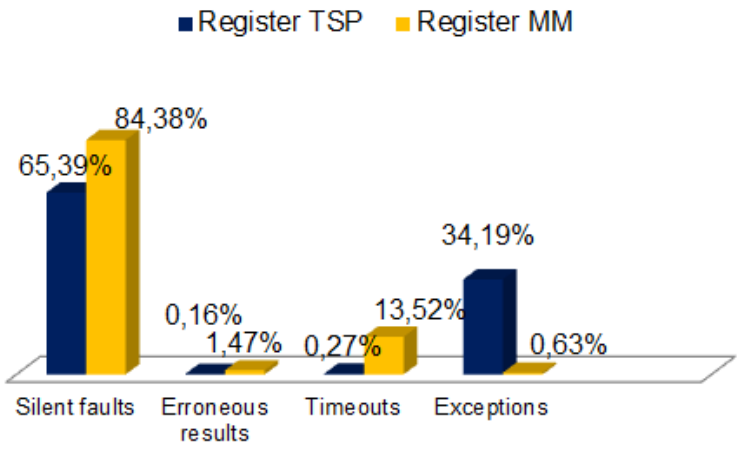

Figure 4.3: Fault-injection consequences when targeting registers on P2041 under SMP

Figure 4.3 illustrates the distribution of the fault injection consequences in the tested applications when targeting registers. The results show that regarding exceptions, TSP application is about 50 times more sensitive to SEUs than MM application. This can be explained by the fact that TSP uses a greater number of registers compared to MM. Also, it is important to consider the type of application. In TSP, processor registers access more frequently to the code section than data section while MM does the opposite. Concerning timeouts, MM algorithm is about 50 times more sensitive than TSP because in MM all the threads work together to determine the result. In the case of TSP, a thread can find the right result by itself.

Fault injection in memory region

These fault injection campaigns target only the private code memory sections of the corresponding process: the initial process stack memory, the thread's stacks memory and the process' heap memory. Other regions included the shared library memory were not considered.

The memory regions are dynamically assigned by the OS during the process execution. It is important to note that for each run, a new process is generated by the operating system. Therefore, the sensitive area of the target memory region dynamically changes in time. The results of this campaign are summarized in Table 4.9.

Table 4.9: Results of fault injection in memory for SMP scenarios

Application	*Bit-flip injected*	*Silent fault*	*Erroneous results*	*Timeouts*	*Exceptions*
TSP	7500	7244	0	144	112
MM	50000	29911	1302	7127	11660

The probability of injecting a fault in a memory cell that is being occupied by the process is lower than the probability of injecting a fault in a register that is being used. Figure 4.4 shows the distribution of the fault injection consequences when targeting memory.

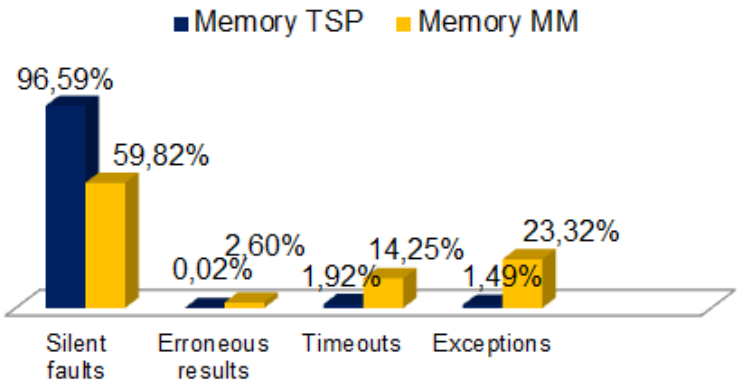

Figure 4.4: Fault-injection consequences when targeting memory locations on P2041 under SMP

In this fault injection campaign, exceptions and timeouts are respectively 15 and 7 times more frequently in MM than in TSP. This can be explained by the following reasons:

- MM data is about 50 times greater than TSP data.
- TSP accesses shared library functions more frequently than MM.

Concerning result errors, MM presents 130 times more errors than TSP. This is explained due to the area occupied by the variables in MM is considerably higher than the one of TSP, and also, due to the fact that TSP has an intrinsic fault tolerance characteristic.

The injection error-rates are calculated from the fault-injection results applying (3.3), and considering as errors the erroneous results, timeouts and exceptions. The results of the four campaigns are summarized in Table 4.10.

Table 4.10: Error-rates for SMP scenarios in P2041

Application	*Nb. of faults injected*	*Nb. of errors in Registers*	*Nb. of errors in Memory*	τ_{Inj_Reg}	τ_{Inj_Mem}
TSP	7500	2596	256	34.61%	3.41%
MM	50000	7809	24846	15.62 %	40.18%

4.4.4 Experimental evaluation by neutron radiation

Both applications were evaluated by neutron radiation ground testing. The details and the results of each scenario are discussed in the respective sections. At the end of the MM section, a comparison of both scenarios is presented.

Evaluation of TSP-SMP-P2041-NR scenario

Three radiation campaigns were carried out to calculate the dynamic cross section in SMP scenario ($\sigma_{DYN\_TSP\_SMP\_P2041}$) for a 16-cities TSP. In the first campaign the code was loaded from the NOR flash memory provided in the design board. During the test, the application definitely stopped due to a fatal crash in the OS after only 22 minutes. When a reboot of the system was performed, the image of the OS could not be loaded since the NOR flash memory was corrupted as well. Once the OS image was restored, the test continued but a fatal crash in the OS occurred again after 28 minutes, giving a total test time of 50 minutes. That is the reason why for the other two campaigns the OS image was loaded from a hard disk. The second and the third campaigns lasted one and four hours respectively. Table 4.11 shows the characteristics of the radiation test campaigns.

Table 4.11: Test campaigns characteristics for TSP-SMP-P2041-NR scenario

Test Campaign	*Flux* $[n \cdot cm^{-2} \cdot s^{-1}]$	*Time* [min]	*Fluence* $[n \cdot cm^{-2}]$
Test 1	$\sim 1.96 \times 10^5$	50	6.00×10^8
Test 2	$\sim 1.62 \times 10^5$	60	5.83×10^8
Test 3	$\sim 1.45 \times 10^5$	240	20.88×10^8

Table 4.12 summarizes the obtained results for the three tests. The fault classification was done based on the OS fault-handling messages and the monitor application. Faults with multiple indications were scored at the most critical level. The order of the rows in this table depends on the criticality of the fault, being the last one the most critical. It is important to note that the results include the messages obtained during the execution of the application and during the idle time.

Most of the *Machine Check Exceptions (MCEs)* were produced by errors affecting the

Table 4.12: Results of radiation experiments for TSP-SMP-P2041-NR scenario

SEE Type	*Type of OS fault*	*Test1*	*Test2*	*Test3*	*Consequences*
SEU	Machine Check exception - Cache	6 0	10 0	53 1	None Unreliable sys.
SEU	Machine check exception - Code lost	0 1	0 2	1 0	None Timeout
SEU	Other error messages	1 0	0 0	0 1	Timeout Unreliable sys.
SEU	Abnormal process termination	6 2	3 3	11 22	Timeout Unreliable sys.
SEFI	System hang	3	3	17	System crash
SEFI	Automatic system restart	1	1	8	System crash
	Total	20	22	114	

cache memories. When the condition exception was recoverable by the system, there were no consequences neither in the application nor in the system. However, there was one case in which a system process of the scheduler was affected. A *Machine check exception - Code lost* occurs when the *MCE routine* has lost the raised error code that has produced the exception; consequently there is no possibility to determine the source of the error.

An *Abnormal process termination* occurs when the monitor detects a timeout in the application, or when the *MCE* logs an exception in kernel code which causes an unreliable

system condition. A *System hang* is produced when the system shell does not respond to any command, or when the *MCE* logs a message showing that a rebooting is needed. In the most critical level, the *MCE* logs an *Automatic system restart* message. Finally, there were errors that did not come neither from the *MCE* nor the monitor application. They were classified as *Other error messages* and have caused, in one case an application timeout, and in the other case a killing of a system process.

To estimate the dynamic cross-section, only faults that led to *unreliable system* condition, application *timeouts and system crashes* were considered. The total number of events for the three tests was 156, and among them 86 produced errors. The total fluence was $3.27 \times 10^9 n \cdot cm^{-2}$. Applying Equations (2.1) and (3.1) for a confidence level of 0.95, the dynamic cross section in SMP scenario is:

$$2.10 \times 10^{-8} \frac{cm^2}{device} < \sigma_{DYN\_TSP\_SMP\_P2041} < 3.25 \times 10^{-8} \frac{cm^2}{device} \tag{4.2}$$

Evaluation of the MM-SMP-P2041-NR scenario

Two radiation campaigns were carried out to evaluate the dynamic cross-section in SMP scenario ($\sigma_{DYN\_MM\_SMP\_P2041}$) for a 80×80 MM. Table 4.13 shows the characteristics of the radiation test campaigns.

Table 4.13: Test campaigns characteristics for MM-SMP-P2041-NR scenario

Test Campaign	*Flux* $[n \cdot cm^{-2} \cdot s^{-1}]$	*Time* *[min]*	*Fluence* $[n \cdot cm^{-2}]$
Test 1	$\sim 1.62 \times 10^5$	33	3.21×10^8
Test 2	$\sim 1.45 \times 10^5$	90	7.83×10^8

Table 4.14 summarizes the obtained results for the two tests. The fault classification was done based on the OS fault-handling messages and the monitor application. Faults with multiple indications were scored at the most critical level. The order of the rows in this table depends on the criticity of the fault, being the last one the most critical. It is important to note that the results include the messages obtained during the execution of the application and during the idle time.

To estimate the dynamic cross-section, only faults that led to *unreliable system* condition, application *timeouts and system crashes* were considered. The total number of

Table 4.14: Results of neutron radiation tests for MM-SMP-P2041-NR

SEE Type	*Type of OS fault*	*Test1*	*Test2*	*Consequences*
SEU	Machine Check exception - Cache	1	11	None
		0	4	Unreliable sys.
SEU	Machine check exception - Code lost	0	0	None
		1	1	Timeout
SEU	Abnormal process termination	3	13	Timeout
		1	4	Unreliable sys.
SEFI	System hang	1	9	System crash
SEFI	Automatic system restart	1	4	System crash
	Total	8	46	

events for the three tests was 54, and among them 42 produced errors. The total fluence was $1.104 \times 10^9 n \cdot cm^{-2}$. Applying Equations (2.1) and (3.1) for a confidence level of 0.95, the dynamic cross section for MM running on SMP is:

$$2.74 \times 10^{-8} \frac{cm^2}{device} < \sigma_{DYN\_MM\_SMP\_P2041} < 5.14 \times 10^{-8} \frac{cm^2}{device} \tag{4.3}$$

The *machine-check-error report* and the *traps* implemented in the OS allowed determining the source of the errors of these faults. Figure 4.5 illustrates the distribution of OS depending on the zone affected by the SEE. One can easily verify that the zone including protection mechanisms (in the left side of the figure) has minimum fatal consequences on the system, while particles affecting not protected zones (right side of the figure) produced the majority of fatal consequences. Also the figure gives an idea of the distribution of particles affecting different zones according to the resources used by the system or application. For this specific scenario, around 20% of faults affects processor and chip resources that are not protected and cause errors in application.

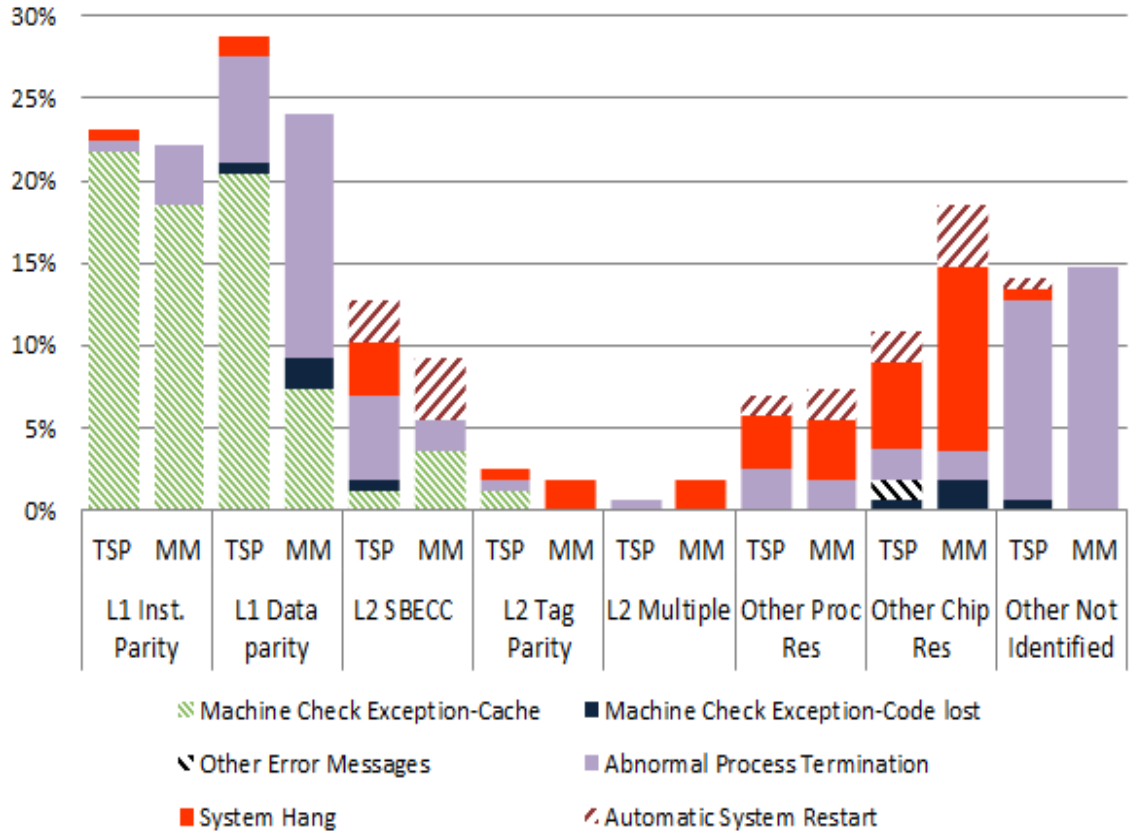

Figure 4.5: Distribution of OS faults by zones affected by SEE in SMP P2041 NR scenarios

4.5 Discussion of the obtained results

The P2041 multi-core processor has been evaluated through different scenarios. A comparison of the obtained results provide interesting clues about the variables that affect the reliability of parallel applications running in a system exposed to neutron radiation.

Figure 4.6 summarizes the results obtained in section 4.4.3 concerning fault-injection campaigns on TSP and MM applications where data and code memory sections were targeted. Note that shared libraries as well as OS were not modified during these campaigns. From the results issued from fault injection, it is possible to see that when targeting the MM, 15.62% and 40.18% of the injected faults produced errors in registers and memory respectively. On the other hand, for the TSP application, 34.61% and 3.41% of the injected faults produced errors in registers and memory respectively .

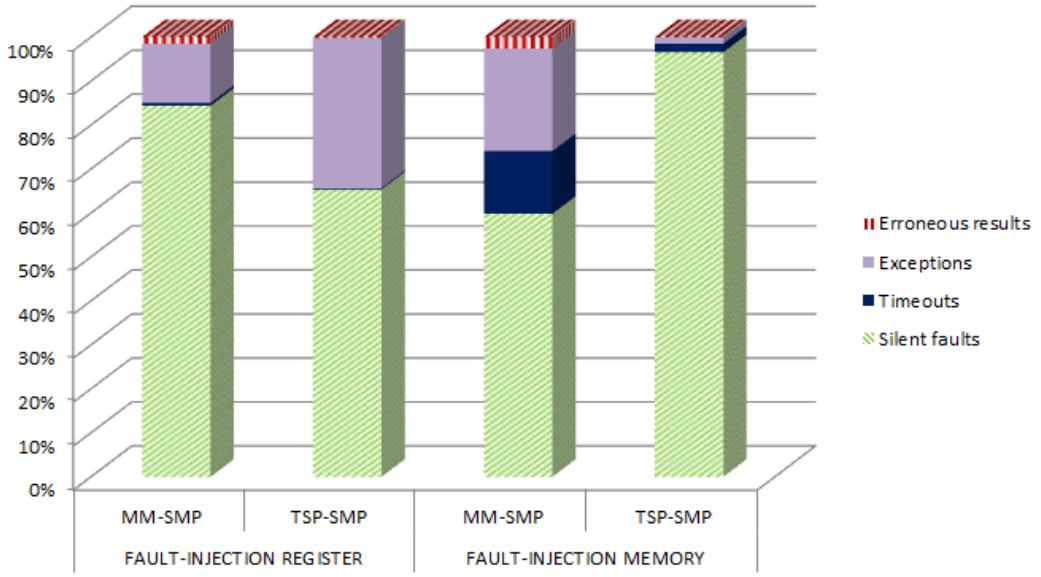

Figure 4.6: SEE consequences per scenario from fault-injection campaigns on P2041

In contrast, the distribution of the SEEs consequences from the three dynamic scenarios evaluated under neutron radiation is shown in Figure 4.7. In the MM-AMP scenario, 13.33% of events caused errors. Among them, 6.67% produced erroneous results and 6.67% caused exceptions (system hangs). On the other hand, 77.78% of the observed events in the MM-SMP produced errors. 29.63% of them caused timeouts, and 48.15% caused exceptions (unreliable condition and system crash). Finally, in the TSP-SMP, 55.13% of detected events lead to errors on the system or application. 13.46 % of them produced timeouts and 41.67% exceptions (unreliable condition in system and system crash).

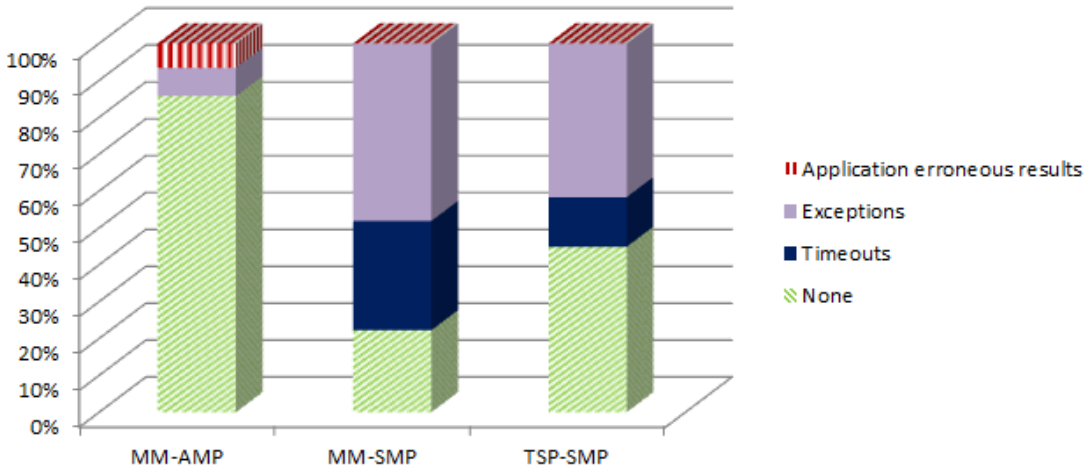

Figure 4.7: SEE consequences per scenario from radiation testing on P2041

Comparing the results issued from radiation experiments with those obtained from fault-injection campaigns, one can see that the use of OS has a significant impact in the SEE consequences on a parallel application, since the OS performs kernel services and manages the ensemble of resources involving sensitive zones that can not be targeted by fault injection because they could be affected by software means. Indeed, the obtained results reveal that errors may occur in SMP mode, even if the OS is in IDLE mode.

For evaluating the reliability of the three scenarios tested under neutron radiation, as it was stated in section 3.1 the expression used is equation (2.3).

$$R(t) = e^{-\lambda t} \qquad \text{(2.3 revisited)}$$

Where $\lambda = \sigma \times \phi$, being σ the cross-section of the system, and ϕ the flux of particles at the specific operation environment. It was considered as an example, the radiation environment at avionic altitude (35000 feet). The upper limit of the confidence interval of their corresponding dynamic cross-sections was taken into account to provide the worst-case reliability of the applications. Figure 4.8 illustrates the reliability of the application evaluated for avionics. These results suggest that the great difference in the reliability of the application for both AMP and SMP scenarios comes not only from the fact that SMP uses OS functions, but also due to the parallel algorithm implemented. It is important to note that in the AMP scenario, the implemented MM is a sequential version.

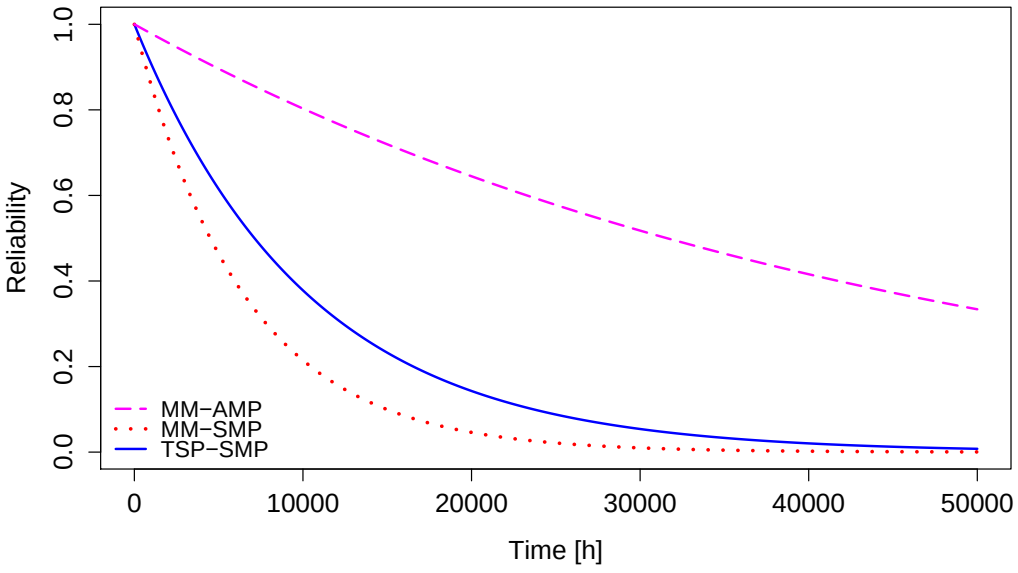

Figure 4.8: Evaluation of reliability under neutron radiation for the P2041

On the other hand, to establish the reliability of the TMR technique evaluated by fault injection, one should consider the failure rate obtained by the application of CEU approach expressed as follows:

$$\begin{aligned} \lambda_{CEU} &= \tau_{SEU} \times \ \phi \\ \lambda_{CEU} &= \tau_{Inj} \times \sigma_{Static} \times \phi \end{aligned} \tag{4.4}$$

Results from reference [RVB+16] allows determining the static cross-section of the multi-core P2041 as $8.51 \times 10^{-9} cm^2/device$ being its confidence interval $[4.40 - 14.90] \times 10^{-9} cm^2/device$. To compute the failure rate, the upper limit of the confidence interval was used as well as the injection error rate only considering single bit-injection per execution. Figure 4.9 illustrates the reliability of this application evaluated for avionics. In the figure, they are plotted the reliability computed using the obtained static-cross section, and using the upper confidence limit(CI).

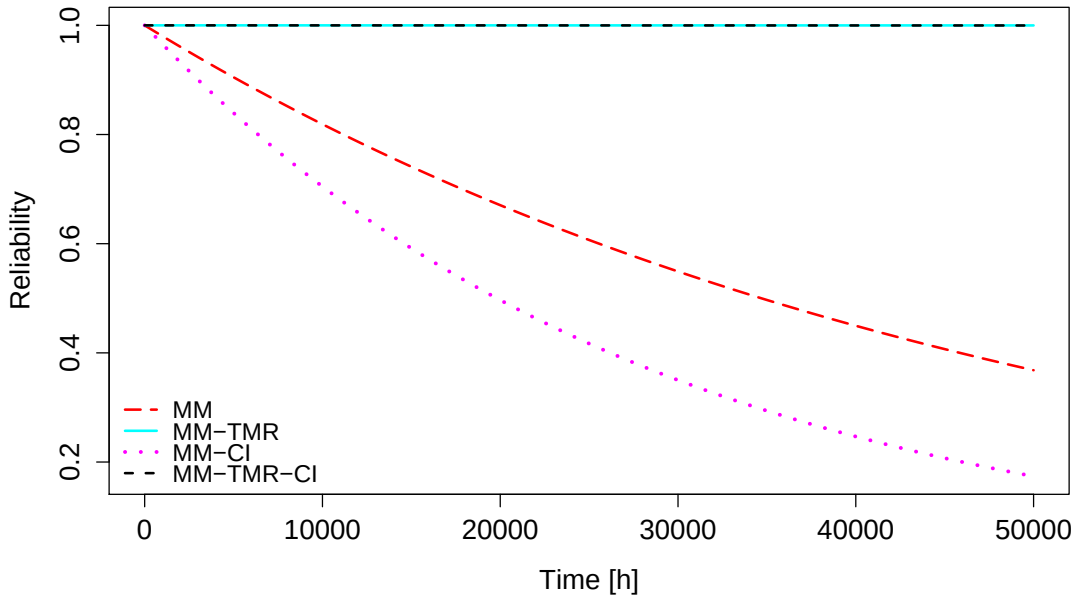

Figure 4.9: Evaluation by fault-injection of the MM reliability implementing TMR on P2041

From the results one can verify the effectiveness of the TMR fault-tolerant approach to improve the reliability of the applications. Due to the fact that fault injection and radiation ground testing campaigns implement the same application, both results can be briefly compared. This comparison shows that the behavior of the application under

radiation and fault injection is basically the same. The slightly difference in the reliability value is due to the size of the targeted sensitive area for each case [1].

4.6 Concluding Remarks

The dynamic AMP tests have demonstrated that in spite of the parity and ECC protection mechanisms, errors have been occurred in the application results. A deeper analysis allowed determining that errors were caused by MBUs in the *address tags* and *data* of the targeted device.

Radiation and fault injection campaigns confirm that the impact of SEUs on parallel applications strongly depends on the software environment: (1) Operating System capabilities, (2) multi-processing mode and (3) intrinsic characteristics of the application (including the used multi-processing paradigms (pthread or OMP)).

In the literature, there is an interesting work that compares the performance of the SMP and AMP modes both using operating systems for a dual-core, concluding that SMP outperforms AMP mode [DWVVL+13]. Inferring this affirmation to results presented in the previous discussion, which demonstrate that AMP is more reliable, it is possible to suggest the existence of a trade-off between reliability and performance according to the selected multi-processing mode.

In spite of the significant reliability improvement achieved when applying TMR, the implementation of this redundancy technique for a quad-core processor limits its parallel processing capability. Per contra, due to the high number of cores existent in many-core processors, its implementation can be an interesting solution to mitigate the impact of SEE on applications running on such devices.

From the above mentioned, it is desirable to evaluate this redundancy technique for improving the reliability of parallel applications running on a many-core processors. However, a mandatory step prior to propose an approach based on redundancy is to evaluate the SEE sensitivity of the selected application. Chapter 5 will discuss this evaluation through the case-studies applied to the MPPA many-core.

[1] Matrix size of 40×40 for fault-injection versus 80×80 for neutron radiation

Chapter 5

Case-studies based on the many-core processor

For evaluating the impact of SEEs on applications running on the MPPA many-core processor, two case-studies were proposed. The first one maximizes the use of internal compute-clusters resources while the second one exploits massive paralleling. The evaluation platform used to appraise the many-core processor is the MPPA Developer. The evaluation was achieved through fault injection campaigns and/or neutron radiation testing. Details of each case are described in the related sections.

5.1 Description of the MPPA-256 many-core processor

The KALRAY MPPA-256 is a 64 bit many-core processor manufactured in TSMC CMOS 28nm technology. The processor operates between 100 MHz and 600 MHz, for a typical power ranging between 15 W and 25 W. Its peak floating-point performances at 600 MHz are 634 GFLOPS and 316 GFLOPS for single and double-precision respectively.

The second version of this processor, called Bostan, is considered in this work. This device is based on an array of 16 CC and 2 I/O clusters that are connected to the 32 nodes of NoC with a toroidal 2D topology. The 16 inner nodes of the NoC are connected to the CC while the 16 peripheral nodes are connected to the I/O subsystems. The NoC is comprised of 2 parallel networks: the Data NoC (D-NoC) that is optimized for bulk data transfers while the Control NoC (C-NoC) is optimized for small messages at low latency [Kal16, dDAB+13]. Figure 5.1 illustrates an overview of the device.

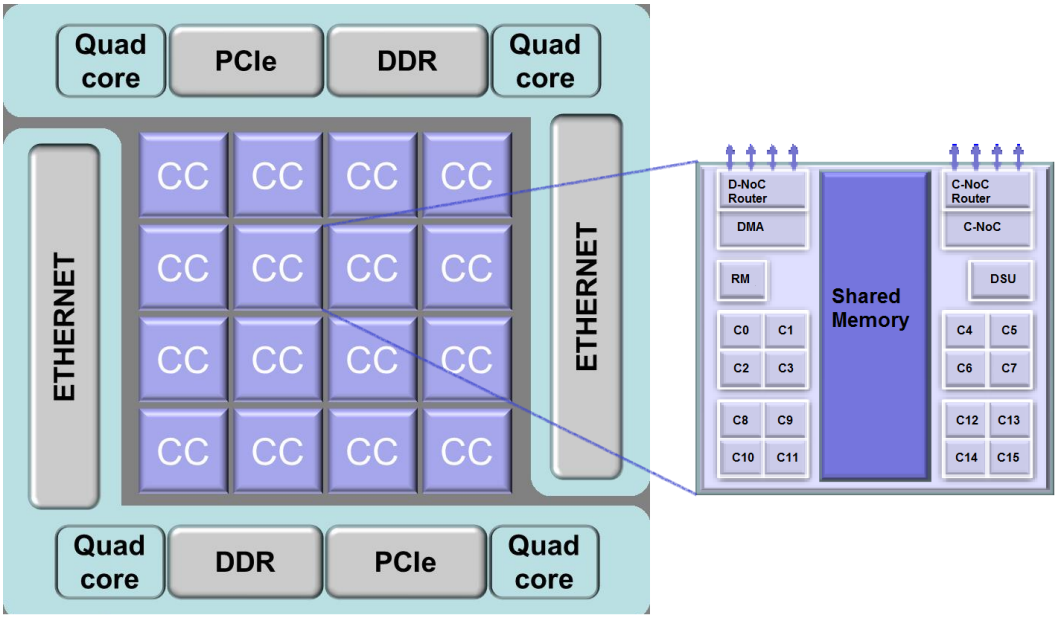

Figure 5.1: MPPA-256 many-core processor components

The MPPA is a distributed memory system. Each Compute Cluster (CC) is built around a multi-banked local Static Memory (SMEM) of 2MB shared by 17 identical Very Long Instruction Word (VLIW) cores: 16 Processing Engine (PE) + 1 Resource Manager (RM) without cache coherency. This configuration creates an interconnection with high bandwidth and throughput between PEs. The PEs are dedicated to execute the application code while the RM is in charge of managing the NoC interfaces by means of dedicated event lines and interrupts.

Each IO cluster features 2 I/O subsystems. Each I/O subsystem comprises 4 network interfaces and a quad-core of RMs in a Symmetric Multi-Processing (SMP) configuration connected to two main banks of 2 MB and to a 4-lane Interlaken controller. One of the I/O subsystem in the IO cluster is connected to a DDR interface (access to 2 GB) and 8-lane PCIe controller. The other is connected to a quad 10Gb/s Ethernet controller.

The MPPA integrates 256 PE cores and 32 RM cores. Both types of cores are based on the same VLIW 32-bit/64-bit architecture. The VLIW core implements separate 2-way associative instruction and data cache memories. There is no hardware cache coherency mechanism between cores, nor between data cache and instruction cache. However, to enforce memory coherency, several software mechanisms are available to programmers.

5.1.1 Sensitive zones

The main memory areas of the many-core processor are covered by error protection mechanisms except the instruction and data cache memories of the VLIW core that are protected by parity. The SMEM of the clusters interleaves bits of 8 adjacent 64-bit words which allow localized errors spread as multiple SECC errors. They are detected and corrected on the fly. The NoC router queues (512 of 32-bit flits each) are also protected by ECC. Note that SECC errors are silently corrected while DECC errors are signaled. The registers do not implement any protection mechanism.

Table 5.1: Memory cells of the MPPA-256 many-core processor

Sensitive zone	*Location*	*Capacity*	*Description*
SMEM	Computing Cluster	2 MB per cluster	Static Shared Memory
SMEM	I/O Cluster	4 MB per cluster	Static Shared Memory
IC-CC	CC VLIW core	8 KB per core	Instruction Cache
DC-CC	CC VLIW core	8 KB per core	Separated Data cache
IC-IO	IO VLIW core	32 KB per core	Instruction Cache
DC-IO	I/O cluster	128 KB per core	Shared/Separated Data cache
GPR	VLIW Core	64 registers of 32 bits per core	General Purpose Registers
SFR	VLIW Core	51 registers of 32 bits per core	System Function Registers

The VLIW core includes General Purpose Registers (GPRs) and System Function Registers (SFRs). Outside the processor, the MPPA comprises different types of specific registers for controlling DMA, D-NoC ,C-NoC, cluster power controller, trace, debug, and the different I/O. Table 5.1 summarizes the sensitive areas of the many-core processor that were targeted during this work.

5.1.2 Programming MPPA issues

In order to program the MPPA Developer, the application is commonly divided into a HOST part and a MPPA part. However, it is also possible to run applications only on the MPPA. The HOST part can use full Linux capabilities available on the CPU host. For the MPPA part, each Compute Cluster or IO cluster can run an executable file. Therefore, the many-core processor can simultaneously run as many executable codes as there are clusters. The execution of a multi-binary file on the hardware or on the Platform simulator can be accomplished either by Joint Test Action Group (JTAG) or PCIe expansion bus [dDdML$^+$13, Kal15].

The MPPA provides a great configuration flexibility. It allows running independent applications per cluster or to program multi-cluster applications in a classic master/slave scheme, where the IO cluster performs as the master.

Communication between the HOST and the MPPA is achieved using specific drivers provided by the manufacturer. For inter-clusters communication, it is also provided a library with a set of functions for data exchanges through the MPPA Network-on-Chip (NoC).

The main challenges the programmers face when adapting parallel applications to this device are the following: (1) the use of NoC primitives for communication, (2) the cache coherence must be guaranteed by the programmer, and (3) the limited memory inside the cluster (2MB for OS, code and data).

5.2 Case-study D: Many-core execution with minimal use of NoC services

This case-study aims at evaluating the sensitivity of the internal computing-cluster resources. To do this, the many-core processor is configured in bare-metal where each cluster executes independently the same application. Its behavior was evaluated by fault injection and neutron radiation tests.

Results presented in [VRR$^+$16] show that during the radiation test campaigns on the MPPA, no errors were produced in SMEMs of the clusters since they implement ECC and interleaving. Consequently, this thesis only considers fault injection in processors'

registers. On the other side, for the evaluation of the device under neutron radiation, two scenarios are considered: the first one with cache memories enabled and the second one with cache memories disabled.

5.2.1 System configuration

The many-core was configured in AMP mode and has implemented a bare-metal application to minimize the use of libraries. The dynamic response of the device was evaluated through the execution of a testing application that must accomplish the following characteristics: (1)intensive use of the cluster resources, (2) code and data size maximum of 2 MB, (3) evenly load distribution among PEs, (4) enough execution time to ensure all PEs running in parallel.

In general, there are no shared resources between compute clusters. However, the NoC resources are used for inter-cluster communications when the IO cluster spawns the executable code to the compute clusters, and when the clusters log the results.

The code is loaded by means of the JTAG in the SMEM of the IO cluster 0. This cluster then spawns the same executable into the 16 compute clusters and orders them to start the execution of the program. Within each cluster, the RM core wakes-up the 16 PE cores, and each one of them starts the execution of the application.

Benchmark Details

The application to be tested within each compute cluster of the device is an assembler optimized version of a cooperative 256×256 matrix multiplication. The matrix multiplication is performed 256 times, and the result C is the summation of these computations as stated in (5.1). The iteration of the matrix operation is done to guarantee that each cluster computes enough time so that all the clusters work in parallel during a considerable time slice. For a 256 matrix size, it takes around 1M IO cycles to spawn 1 cluster. Since clusters are spawned one after another, cluster 15 starts execution around 15M IO cycles after cluster 0.

$$C = \sum_{n=1}^{256} A \times B \tag{5.1}$$

A, B and C are single precision floating-point matrices. The size of the matrix was cho-

sen so that data remain in the local SMEM memory. Each compute cluster is configured in AMP mode and the computational work is distributed evenly among the processing cores, so each PE belonging to the cluster computes $^1/_{16}$ of the cluster result. The synchronization of the computation is done by events between the RM and the PE cores. The RM wakes up the 16 PEs and sends a notification to each one to start the computation. Then, it waits for a notification from each PE indicating the work was done.

Once all PEs computations have finished, the RM core compares the result matrix with a reference result-matrix E, and reports any mismatch including the associated addresses and values. Then, the matrix C is filled up with zeros and the PEs start again the computation. The program executes continuously the same algorithm in each cluster along the radiation test.

Since there is no hardware memory coherency in the compute cluster, each PE ensures memory coherency by software means, by updating shared data before (read coherency) and after the computation (write coherency). In addition, the RM calls the memory coherency functions when using the shared data.

5.2.2 Fault-injector details

The fault injector used in this case-study is similar to the one presented in [VRM+14] and described in chapter 4. It uses inter-processor interrupts to inject faults in other cores. However, in the case of the MPPA processor, it is possible to benefit of the multiplicity of cores for using some of them as fault injectors.

This case-study considers one fault injector per cluster due to each CC performs the application independently of the others. While the application is running on the processing engine cores, the resource manager core of each compute cluster performs the fault injection. It randomly selects the target core (any of the processing engine or itself), register, injection instant (in terms of clock cycles), and the bit to be inverted. At the injection instant, the RM sends an inter-processor interrupt to the selected core. The latter performs the bit-flip in its register.

This work only considers SEU emulation where one SEU per cluster and per run is injected. The fault injection procedure is repeated several times in order to obtain enough amount of samples to calculate the injection error-rate τ_{inj}.

The fault-injection campaign is devoted to inject faults in GPRs and SFRs of the

compute cluster's cores (PEs or RM). Since some SFRs are non-writable by software means, only 34 SFRs of 51 SFRs are targeted. Among the targeted SFRs, the most critical ones are the 8 registers saved during context switching: Shadow Program Counter (SPC), Shadow Program Status (SPS) Return Address (RA), Compute Status (CS), Processing Status (PS) , Loop Counter (LC), Loop Start Address (LS) and Loop Exit Address (LE). On the contrast, other registers such as processing identification, system reserved and performance monitor are not targeted. The SPC and the SPS registers serve to emulate bit-flips in the PC and PS registers respectively. This is done due to the fact that in the context switching of the interruption routine, the value of the PC and PS are saved in the shadow registers (SPC and SPS) and when the program flow exits from this routine, it uses the values of these registers to be restored in the PC and PS to continue with the current program execution.

5.2.3 Experimental evaluation by fault injection

In these experiments, the SEU faults were injected at a random instant within the nominal duration of the executed program which was around 5.3×10^8 clock cycles.

Table 5.2 shows a general overview of the fault injection campaign where 94316 faults were injected in the GPRs and accessible SFRs.

Table 5.2: Results of the fault-injection campaigns for MM-AMP-MPPA

Targeted Registers	*Silent faults*	*Erroneous results*	*Timeouts*	*Exceptions*
GPRs	36472	16387	6678	1996
SFRs	22745	2034	6365	1639
TOTAL	59217	18421	13043	3635

From these results, it was calculated the application error-rate applying (3.3), and considering as errors the erroneous results, timeouts and exceptions.

$$\tau_{Inj} = \frac{Nb\ of\ Errors}{Nb\ of\ Fault\ Inj} = \frac{35099}{94316} = 37.21\%$$

This result shows that 37.21% of the injected SEUs in the accessible registers cause

errors in the application. Since registers have no protection mechanisms, this campaign is very useful to emulate the behavior of the application in presence of SEUs.

5.2.4 Experimental evaluation by neutron radiation

The goal of this test is to evaluate the dynamic behavior of the many-core processor when no operating system is used. To do this, two different scenarios are considered.

In the first scenario, the cache memories of the VLIW cores are all enabled and configured in write-through mode. In the second one, the cache memories are all disabled. In both cases, the application cross-section (σ_{DYN}) and the average execution time are obtained. The SECC and other errors such as Data Parity (DPAR) and Instruction Parity (IPAR) are reported with a message.

Test parameters

The device under test was decapsulated and placed facing the center of the target perpendicularly to the beam axis at a distance of 15.2 ± 0.5 cm. The DUT fan was placed laterally to cool-down the device under test, rather than being placed on the device. Consequently, the computing cluster frequency was set to 100 MHz to reduce power consumption. The bias voltage of the device was set to 0.9V. The neutron beam energy was 14 MeV with an estimated flux of $1.2 \times 10^5\ n \cdot cm^{-2} \cdot s^{-1}$ at 500 Hz frequency with an error of $\pm\ 0.1 \times 10^5\ n \cdot cm^{-2} \cdot s^{-1}$.

Experimental Results

One radiation test campaign per scenario was carried-out. For the first scenario, both instruction and data cache memories were enabled in the compute cluster' cores. In the second scenario cache memories were all disabled. Each test campaign had an exposure time of one hour providing a fluence of about $4.32 \times 10^8\ n \cdot cm^{-2}$. Table 5.3 summarizes the results of both dynamic radiation campaigns. *Test CE* means test with caches enabled while *Test DE* means test caches disabled.

From table 5.3, it is possible to identify five different types of errors. Among them, the SECC and the Instruction and Data Cache Parity errors were all corrected by the ECC and parity protections. On the contrary, Register Trap and Memory-Comparison Failed

Table 5.3: Results of the dynamic radiation tests for MPPA scenarios

Detected Error	*SEE Type*	*Occurrences Test CE*	*Occurrences Test CD*	*Consequences*
SECC	SEU	676	602	None
Data Cache Parity	SEU	36	N/A	None
Inst. Cache Parity	SEU	6	N/A	None
Register Trap	SEFI	1	1	Hang
Memory Comp. Failed	SEU	2	1	Erroneous Result
Total		721	604	

errors are non-correctable errors since processor registers do not implement protection mechanisms. The Memory Comparison Failed error was detected by the RM core when it identifies differences between the results of the application and the expected values. Note that these errors observed during radiation tests were similar to those occurred during fault injection campaigns in GPRs.

To determine the dynamic cross-section only the non correctable errors; *3* for *Test CE* and *2* for *Test CD* were taken into account. For a 95 % confidence interval ($\alpha = 0.05$), the lower and upper limits for the dynamic cross-section when cache enabled were:

$$1.43 \times 10^{-9} \frac{cm^2}{device} < \sigma_{Dyn_{ce}} < 20.29 \times 10^{-9} \frac{cm^2}{device}$$

Since the code is quite small and only occupies around 400 bytes that easily fit in the PEs' caches, there were no cache misses for the instruction caches. Concerning data caches, the miss rate was roughly 2.5%. Even though the 256 PEs are working all the time fully loaded, 30% of the execution time is spent waiting for missing data to arrive from the SMEM.

Regarding the caches disabled scenario, the confidence interval of the dynamic cross-section is:

$$0.56 \times 10^{-9} \frac{cm^2}{device} < \sigma_{Dyn_{cd}} < 16.72 \times 10^{-9} \frac{cm^2}{device}$$

With regard to performance of the application when both instruction and data cache memories are enabled, 679 computations of the matrix multiplication were completed per cluster in one hour. The average computation time was ~ 5.30 seconds at 100 MHz frequency. On the contrary, in the cache disabled scenario 348 computations of the matrix multiplication were completed per cluster in one hour and the average computation time was ~ 10.34 seconds at the same frequency.

Comparing the results of the two dynamic test campaigns, it can be seen that the matrix multiplication algorithm performs twice as fast when cache memories were enabled without reliability penalty, since the detected errors were corrected by the parity protection.

5.3 Case-study E: Many-core execution with intensive use of NoC services.

This case-study aims at evaluating the sensitivity of the MPPA when massive paralleling is used. For that, a CPU-bound (TSP) and a memory-bound (MM) were implemented as parallel inter-cluster applications. The programming paradigm used intra-cluster was POSIX and the inter-cluster communication were done by NoC services. However, due to certain constraints, a *Lowlevel* configuration was used in the IO cluster.

The dynamic response of the TSP and the MM were only evaluated through fault injection in processors' registers, since the SMEM of the MPPA many-core implement effective protection mechanisms.

5.3.1 System configuration

The many-core processor was configured in a typical master/slave scheme for running parallel multi-cluster applications. The master runs on the IO cluster while the slaves run on the Compute Cluster (CC). In this case study, the application was configured without a HOST part. The MPPA part was loaded through the JTAG port to the IO-DDR0. For this scheme, 3 RM cores of the IO-DDR0 were configured:

- The RM1 is the master of the application.

- The RM3 is the fault injector.
- The RM0 coordinates the actions between the application and the fault injector. Also, it is in charged of configuring all the inter-cluster communication.

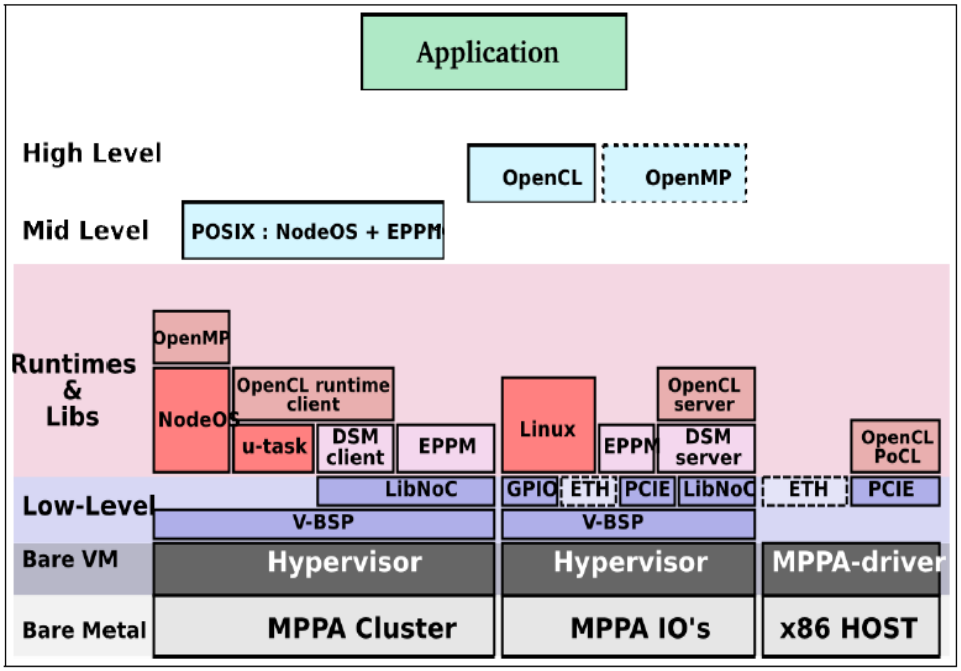

Figure 5.2: Kalray software stack [Kal15]

At present, the Real-Time Executive for Multiprocessor Systems (RTEMS) OS which runs on IOs is only capable of using one RM. For this reason, the *Lowlevel* programming model was selected for the IO cluster. The IO was configured in bare-metal without OS, including the Virtual Board Supporting Package (V-BSP) and LibNOC libraries. Figure 5.2 allows a better understanding of the software configuration details. It illustrates the several abstraction layers included in the software stack of the MPPA ACCESSCORE SDK. These libraries can co-exist independently, and the user can mix and match the libraries he wish to use [Kal15].

On the other hand, the intra-cluster application was configured with the POSIX model. The *main* process runs on the RM1 and is responsible to spawn the sub-processes from the IO to the target CCs. Each sub-process is a multi-thread program based on POSIX. The CCs were configured for using the NodeOS, an OS POSIX standard. The NodeOS implements an asymmetric multi-processing architecture, where the RM performs the kernel routines and NoC interface services while the PEs run one user thread

per PE. All the inter-cluster communications are done through the NoC using the MPPA inter process communication (MPPA IPC). This library contains the routing functions, an API for the power-on and spawning of CCs and the communication primitives: the classic POSIX IPC and some specific primitives for the MPPA many-core [Kal15].

Since there is no shared memory between clusters in the MPPA architecture, the implementation of an inter-cluster application requires distributed algorithms. The following specific MPPA communication primitives were used during inter-cluster communications: *Sync*: N senders /M receivers, *Portal*: N senders /M receivers, and *Queue*: N senders/1 receiver.

To use these primitives, the device must prior initialize some properties of the communication. Any RM can transmit data to the selected CC(s) by using any of the mentioned primitives. However, the IPC library configures the RM0 as the default receiver for portals and queues. Therefore, in this work the RM0 is in charge of initializing the communication primitives and update the related data when receiving information from the CC(s) by the NoC interrupt service.

5.3.2 Benchmark details

The implementation of both applications allows configuring the number of CCs from one to four as well as the problem size. Table 5.4 illustrates the execution time of different possible configurations for the TSP application. The time in seconds is done for a configuration of the device with an operating frequency of 400 MHz.

Table 5.4: Standard execution time for different configuration of TSP on the MPPA

Nb cities	*1 cluster*		*2 clusters*		*3 clusters*		*4 clusters*	
	[Gcycles]	*[s]*	*[Gcycles]*	*[s]*	*[Gcycles]*	*[s]*	*[Gcycles]*	*[s]*
16	30.8	77	16.2	40.5	11.3	28.3	8.6	21.5
17	188.8	472	102.8	257	71.2	178	58.1	145.2
18	521.9	1304.2	260.3	650.8	188.2	470.5	159.8	399.5

```
global portals, queues
procedure RUN_APPLICATION(nb_threads, application_variables, nb_clusters)
sync ← CREATES_SYNC_CONNECTION( )
INITIALIZE_VARIABLES( )
for i ← 1 to nb_clusters
  do SPAWN_CLUSTER( SLAVE_WORK(application_variables, nb_threads) )
WAIT_BARRIER(sync)
finished_clusters ← 0
while finished_clusters < nb_clusters
do { TX_RX(portals)
     TX_RX (queues)
     if CLUSTER_FINISHED( ) = 1
     then { finished_clusters ++

WAIT_BARRIER(sync)
LOG_RESULTS( )
CLOSE_SYNC_CONNECTION(sync)
```

Figure 5.3: Generic master pseudo-code for the applications running on the MPPA

For performing both applications, this case-study considers 4 CCs as *slaves* of the RM1 of the IO cluster. Thus, 5 RMs and 64 PEs are involved on the application itself plus the RM0 that starts, monitors, and manages the messages received by the NoC. Before starting the distributed application, the RM0 initializes the primitives needed for the communication, then wakes up the other RMs, the master of the application and the fault injector on the IO cluster. Once the RM1 is started, it performs the *run_ application* function, which pseudo-code is shown in figure 5.3. It is important to note that, if any cluster does not finish its job in the normal execution time, the RM0 logs an timeout and ends the application.

The TSP was configured to solve a 17 cities problem in order to have enough processing time to observe errors within a reasonable overall simulation time. For this application the RM0 creates the following MPPA connections for the *master*: (1) two *sync* for synchronization with the *slaves*, (2) one *queue* for job request, (3) four *queues* for job response (one per cluster), (4) one *portal* for receiving the results of the minimum distance and its path, and (5) four *portals* for transmitting the results.

The details of the stages of the generic master code are the following:

- During the initialization, the master populates a queue of jobs from which each cluster takes the tasks to be executed by its PEs.

- The first barrier serves to wait that all clusters wake up.

- In the loop the *master* waits for job requests done by the CCs and then assigns jobs to slaves.

- The second barrier waits for the end of the task of all the involved clusters.

- A consolidation of results is done before logging the minimum distance and the corresponding path.

Regarding the code performed by the *slaves*, it corresponds to the multi-threading TSP version detailed in chapter 3 section 3.4.1. Each cluster performs the *slave* code almost independently of the others. It has its own minimum distance and path variables. The only moment when they interact with each other is when a new minimum distance is found by a CC. The latter broadcasts this information to the *master* and other *slaves* so that each one of them updates its related variables. Both synchronization barriers, are intended to synchronize the *slaves* with the *master*. Inside the cluster, each PE uses MUTEX directives for accessing global CC variables.

Each CC executing the *slave* code uses the following MPPA connections: (1) two *sync* for synchronization with the master, (2) one *queue* for job request, (3) one *queue* for job response, (4) one *portal* for receiving the results of the minimum distance and its path, and (5) four *portals* for broadcasting the results.

On the other hand, the MM is practically the same cooperative 256×256 matrix multiplication described in section 5.2.1. The main difference is that the computation is iterated 8192 times. Each *slave* computes $^1/_4$ of the result. The main process of each CC creates 16 POSIX threads, one for each PE. Therefore, each PE calculates $^1/_{64}$ of the result by executing the same assembler optimized version of the cooperative MM used in the previous case-study.

The MPPA connections created for each cluster, the *master* in the IO cluster and the *slaves* in the CC cluster for the MM are: (1) two *sync* for synchronization, (2) one *portal* for receiving the results, and (3) four *portals* for broadcasting the results.

5.3.3 Fault-injector details

This case-study implements one core of the device as fault-injector, being the RM3 core of the IO cluster 0 that performs this function. Fault-injection campaigns target only processor registers as stated in the previous case-study. In order to interrupt the targeted core, it is used the portal primitive. Thus, the configuration of one portal per cluster (*portal_fi*) is needed for fault-injection purposes. In the case of the *master*, the RM0 is in charged of its configuration.

To emulate a bit-flip in the selected register, the fault injector must interrupt the selected core. For achieving this goal, at the random instant, the fault-injector writes the fault-injection variables, *random PE, random address and random bit* in the *portal_fi* of the CC that contains the selected core. This causes an interruption of the RM which immediately assigns the execution of the interruption handler to any one of the PEs. The assigned PE reads the information in the *portal_fi*, that contains the selected core, register and bit to change. If it is the targeted core, it changes the bit in the selected register. Otherwise, it sends an inter-processor interrupt to the corresponding PE which performs the bit-flip emulation.

The fault injection campaigns of both applications are devoted to inject faults in GPRs and 15 SFRs of the PEs belonging to the compute clusters. The targeted SFRs are: (1) the 8 registers saved during context switching: SPC, SPS RA, CS, PS , LC, LS and LE, (2) the 6 event registers, and the Performance Monitor Control (PMC) register.

The other SFR could not be targeted due to the following reasons: (1) 5 SFRs are hardwired and 2 are unused by the current *Bostan* version of the MPPA processor, (2) the 4 performance monitor registers cannot be modified by software, (3) the 5 registers used by the debugger can only be accessed in debug mode, (4) the exception vector is non-writable by the user, and (5) the other SFR are managed by the OS so when the user changes any of them, the OS produces always an exception.

5.3.4 Experimental evaluation by fault injection

The following experiments only consider the emulation of one SEU per execution, so that one SEU is injected in the targeted register belonging to one of the 64 PE cores used by the application. The fault is generated at a random instant within the nominal

duration of the application. In order to avoid the propagation of errors to the next execution, the HOST resets the platform and reloads the code to the MPPA processor after each run. Hence, the random variables required by the fault-injector are provided by the HOST, being the random instant, core, register and bit additional arguments of the main function. Table 5.5 provides details about the two fault-injection campaigns.

Table 5.5: Fault-injection campaigns details for the MPPA

Application	*Standard exec. time*		*Runs per campaign*
	[Gcycles]	*[s]*	
TSP	58.06	145.2	8417
MM	4.55	11.4	72497

Table 5.6 shows a general overview of the fault-injection campaigns on the TSP and MM applications. These results confirm the intrinsic fault-tolerant capability of the TSP application.

Table 5.6: Results of the fault-injection campaigns on applications running on the MPPA

Application	*Silent faults*	*Erroneous results*	*Timeouts*	*Exceptions*
TSP	8197	2	21	197
MM	62071	5215	112	5099

From these results, the error-rates of both applications were calculated by using equation (3.3). The erroneous results, timeouts and exceptions were considered as errors. Table 5.7 summarizes the obtained results. Three types of exceptions were produced: (1) the PE targeted by the fault-injector was completely stuck and does not respond anymore to the JTAG requests "*core stuck*", (2) the bit-flip caused the core tried to access a memory not allocated producing a "*segmentation fault*", and (3) the MPPA device stopped its execution and produced an exit of the process "*device exit*". Figure 5.4 illustrates the details of the error consequences in both application during fault-injection campaigns.

Table 5.7: Error-rate by fault injection for POSIX scenarios in MPPA

Application	*Nb. of faults injected*	*Nb. of errors in Registers*	τ_{Inj}
TSP	8417	220	2.61%
MM	72497	10426	14.38%

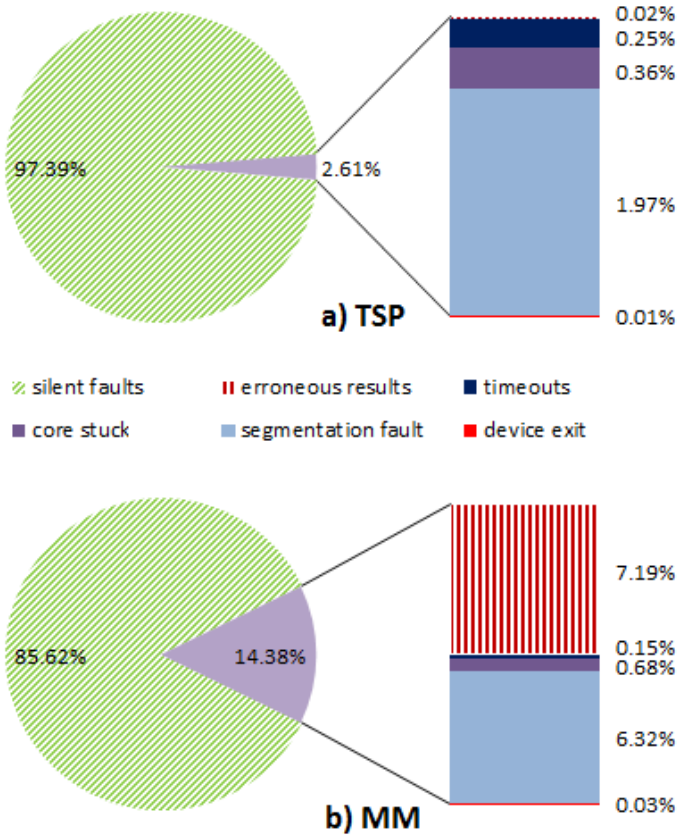

Figure 5.4: Fault-injection consequences on MPPA when targeting registers

As expected, during the fault-injection campaigns, it was observed that the critical registers are application dependent. For instance, in the case of TSP the most critical SFRs registers were: the Shadow Program Counter, the Shadow Program Status and the Return Address. Also, some of the errors were produced by bit-flips in Processing Status and Loop Counter. Concerning the GPRs, the most critical was the GPR13 followed by GPR43, GPR54 and GPR60. On the other hand, in the MM application, the most critical SFRs were: the Shadow Program Counter, the Shadow Program Status, the Return Address, the Loop Start Address and the LE. Regarding the GPRs: the most critical was the GPR13 followed by GPR10, GPR16, GPR34, GPR35, GPR39, GPR42,

and GPR50.

Concerning the type of errors, the most critical one is the *erroneous result* since this error is ignored by the application which considers it as a valid result. The *timeouts* and *exceptions* are not so critical since they are detected and the application/system is able to manage them. For instance, this case-study implements an application with a *timeout* principle, so if the execution time of the application is greater than the standard execution time, the application is finished. In addition, the system exceptions are managed using *traps* in the OS and/or by a monitor in the HOST that kills the process in the MPPA if the application does not respond.

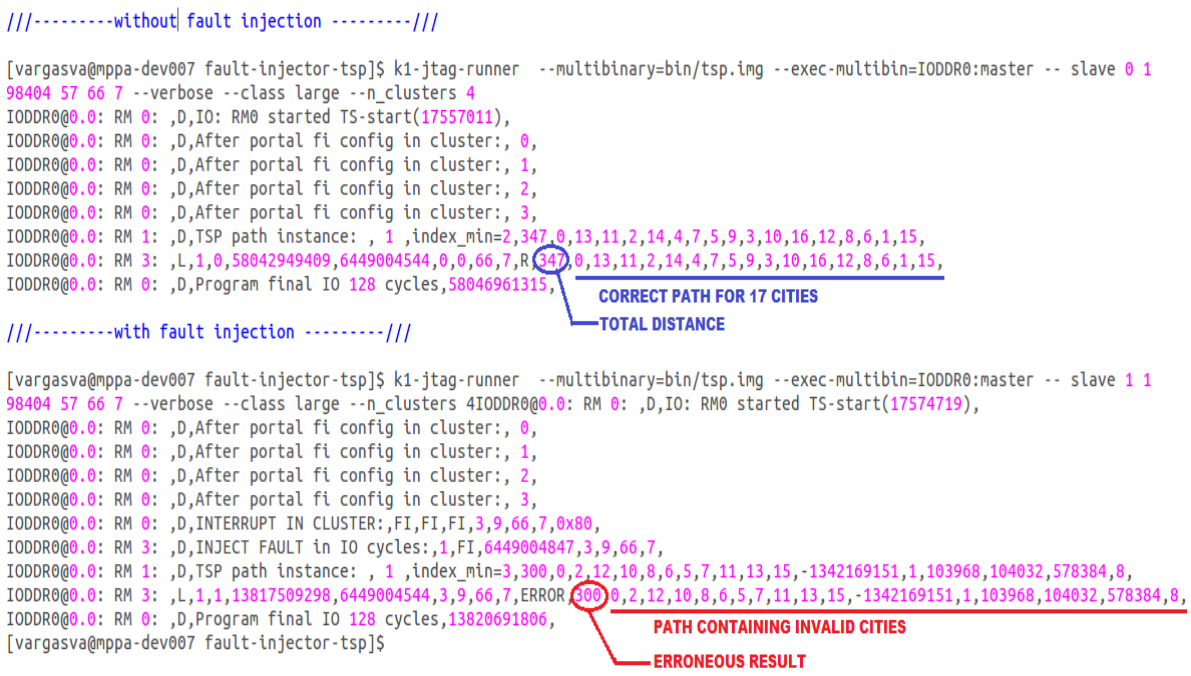

Figure 5.5: Example of erroneous result logged by the TSP

Figure 5.5 illustrates an erroneous result produced in the TSP when a fault injection was performed on core 57, Cluster 3, PE 9, Return Address register, bit 7 at instant $\sim$ 6.45 Gcycle clock. The correct response of the distance is 347, however the application has logged 300. The error is easily observable due to the presence of invalid cities in the path.

5.4 Discussion of the overall results

In this chapter, three scenarios were evaluated:

- A parallel MM running independently on each computing cluster configured on a bare-metal system with shared memory. Two scenarios were proposed: cache memories enabled and cache memories disabled.
- A MM running on a distributed system using a master/slave scheme, where the master runs on the IO Cluster, while each one of the 4 slaves runs in a Compute Cluster. Each slave is an AMP system programmed with POSIX which computes a part of the total result.
- A TSP running on a distributed system using a master/slave scheme, where the master runs on the IO Cluster, while each one of the 4 slaves runs in a Compute Cluster. Each slave is an AMP system programmed with POSIX which computes a part of the total result.

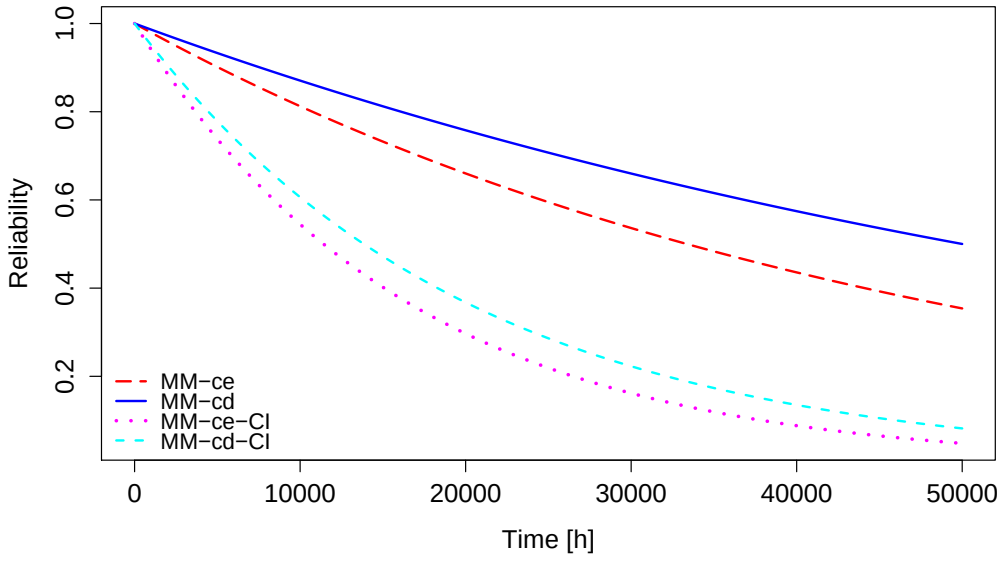

Figure 5.6: Evaluation of reliability by neutron radiation for MM -Bare-board running on MPPA at 35000 ft

The reliability of MM-Bare-board for these scenarios was evaluated under neutron radiation. Results are summarize in Figure 5.6. Two curves per scenario were plotted considering the upper value of the confidence interval and the measured cross-section

value. Note that: (1) *ce* stands for *cache memories enabled*, (2) *cd* stands for *cache memories disabled*, and (3) *CI* stands for *Confidence Interval*. These curves confirm the assertion mentioned in section 5.2.4 which states that by enabling the cache memories there is no reliability penalty. Consequently, for this many-core processor, it is convenient to enable cache memories even in harsh radiation environments.

Regarding fault-injection experiments, the used and targeted resources per scenario are summarized in Table 5.8. Unfortunately, it was not possible to target all the resources used by each scenario for the following reasons: (1) some registers are not-writable by software means, (2) the RM interrupt routine should not be modified by the programmer to guarantee a correct operation of NoC services. On distributed applications, the communication between clusters is achieved by the use of NoC libraries provided by the manufacturer which use interrupts. Thus, it is not possible to overwrite the interrupt routine to program inter-processor interrupts which allow injecting faults in the RM.

Table 5.8: Summary of the different fault-injection scenarios evaluated on the MPPA

Scenario	*Registers Targeted per core*	*Cores Targeted per CC cluster*	*Resources used by the application*	*Resources targeted*
MM - Bare-board	64 GPRs + 34 SFRs	RM + 16 PEs	CCs	CCs
MM - Posix	64 GPRs + 15 SFRs	16 PEs	IO + CCs + NOC	CCs
TSP - Posix	64 GPRs + 15 SFRs	16 PEs	IO + CCs + NOC	CCs

On systems configured in bare-metal, it is possible to target more registers by software means since there are no restrictions caused by the use of an OS. A distribution of the consequences of fault-injection campaigns on the tested applications when targeting registers is shown in Figure 5.7.

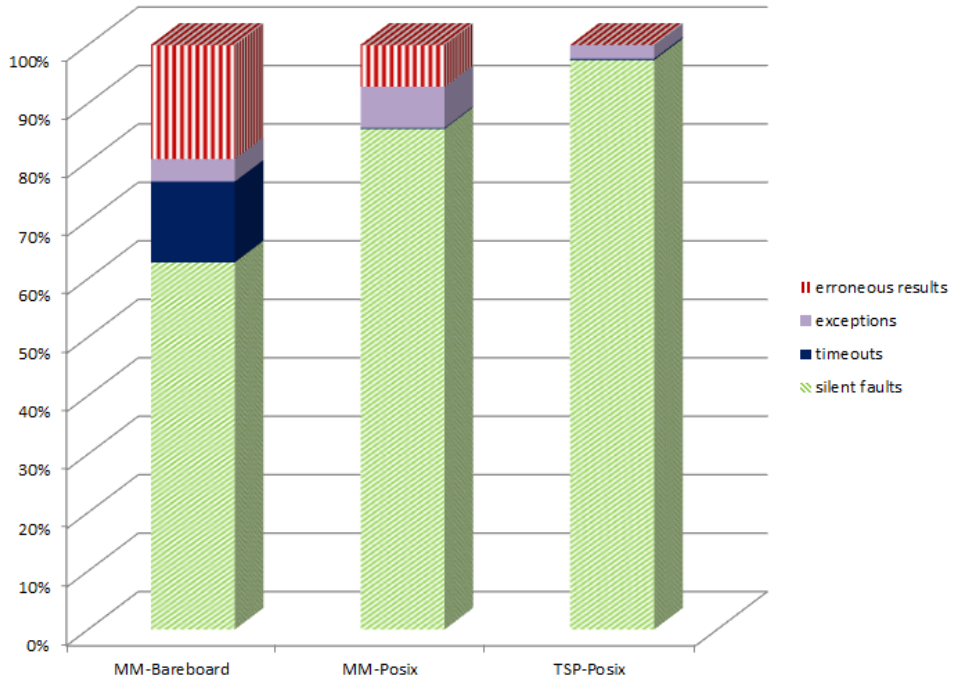

Figure 5.7: Distribution of fault-injection consequences on MPPA for different scenarios

From the above results, MM-Posix scenario seems to be less sensitive than MM-Bareboard. However, it can not be concluded based only on the results obtained from the fault-injection campaigns due to the considerable underestimation of errors in POSIX scenarios. Note that IO clusters and NoC resources used in POSIX cannot be targeted by fault-injection means. Consequently, further radiation experiments are needed: (1) to evaluate at what extent this fault injection limitation affects the results, considering that RMs are in charge of manage the OS, and (2) to provide a fair comparison of both scenarios.

Concerning the evaluation of the reliability of the three scenarios by fault-injection, the failure rate was obtained applying equation 4.4 as follows:

$$\lambda_{CEU} = \tau_{Inj} \times \sigma_{Static} \times \phi \qquad \text{(4.4 revisited)}$$

The σ_{static} of the device was obtained from work [VRR+16], which value is $12.71 \times 10^{-9} cm^2/device$. Figure 5.8 compares the reliability of POSIX applications evaluated for avionics. From these results, it is clearly observable that the TSP has an intrinsic fault-tolerance capability.

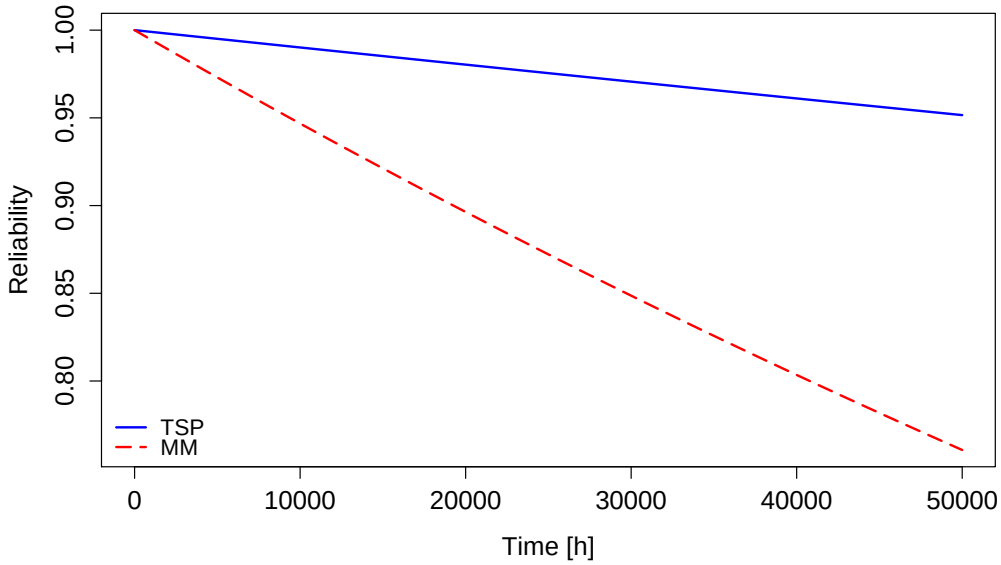

Figure 5.8: Evaluation of reliability for MPPA-POSIX applications at 35000 ft

5.5 Concluding remarks

Once evaluated the dynamic response of the device under neutron radiation, it is shown that by enabling the cache memories is possible to increase the performance of the application without affecting the reliability. It is due to parity protection mechanisms have detected all the errors produced in data and instruction cache memories, and have performed the corresponding cache invalidation. Uncorrected errors encountered in different dynamic tests came from bit-flips in processor registers as they do not implement any protection mechanism.

Results presented in work [VRR+16] show that the predicted application error-rate is reasonably close to the measured one. Consequently, in spite of the hardware complexity of the many-core processor, the mentioned work support the relevance of the use of the CEU approach to estimate the error-rate of applications implemented in such devices.

Although the application error-rate is underestimated because of the inaccessibility of certain registers depending on the configuration of the system, fault-injection campaigns targeting registers are meaningful to emulate the behavior of the application in presence of SEUs.

Considering the redundancy capability of the MPPA many-core processor as well as the lack of protection mechanism at register level, it is imperative to implement a fault-

tolerant approach to reduce the SEE impact on the running application. The goal is to have a reinforced system to be considered for safety-critical and embedded parallel applications. Chapter 6 presents the NMR-MPar approach applied to the *case-study E* and its evaluation.

Chapter 6

NMR-MPar: a fault-tolerant approach

This chapter describes a fault-tolerant approach developed and evaluated in this thesis called NMR-MPar. The name stands for N-Modular Redundancy and M-Partitions. Redundancy and partitioning are the basic concepts used to improve the reliability of applications running on multi-core and many-core processors. A prototype is implemented on the MPPA-256 processor performing two parallel benchmark applications. The effectiveness of this approach is evaluated by fault-injection campaigns.

6.1 Description

Combining the partitioning concept with the modular redundancy technique, NMR-MPar allows multi/many-core processors to perform critical functions in mixed- criticality systems. Benefiting of the capabilities of multi-core and many-core processors, NMR-MPar approach uses the partitioning principle to create different *partitions* that co-exist in the same device. This is typically done by the hypervisor which provides virtual Central Process Units (CPUs) to each partition. In contrast, this approach proposes a physical resource distribution to each partition, to minimize the propagation of faults producing dysfunction in other resources or cores. Each partition can be setup as a mono or multi-core running on bare-metal, AMP or SMP mode. Consequently, there is a considerable versatility on the system configuration that is enhanced by the number of cores comprised in the device.

Furthermore, it is proposed for critical functions, that N partitions with the same configuration participate of a N-Modular Redundancy system. Thus, several partitions

execute the same application and their results are used by a voter to build a fault tolerant system. The voter system can include one or more cores of the device or an external one if desired. Depending on the partition of the device, several N-Modular redundancy systems may run on it concurrently.

Figure 6.1 illustrates an example of this approach implemented on a multi-core having 16 processor cores. The example establishes seven partitions (M=7). Partitions *P0* until *P2* are quad-core while *P3* to *P6* are mono-core. The quad-core partitions are part of a TMR system. Each one of them runs in SMP mode the same parallel application. On the other side, mono-core partitions *(P3 to P5)* are also part of another TMR running another application. Finally, in this configuration the mono-core of the partition *P6* is the voter of both TMRs. The voter system was configured in *bare-metal* to reduce the use of resources minimizing the impact of faults.

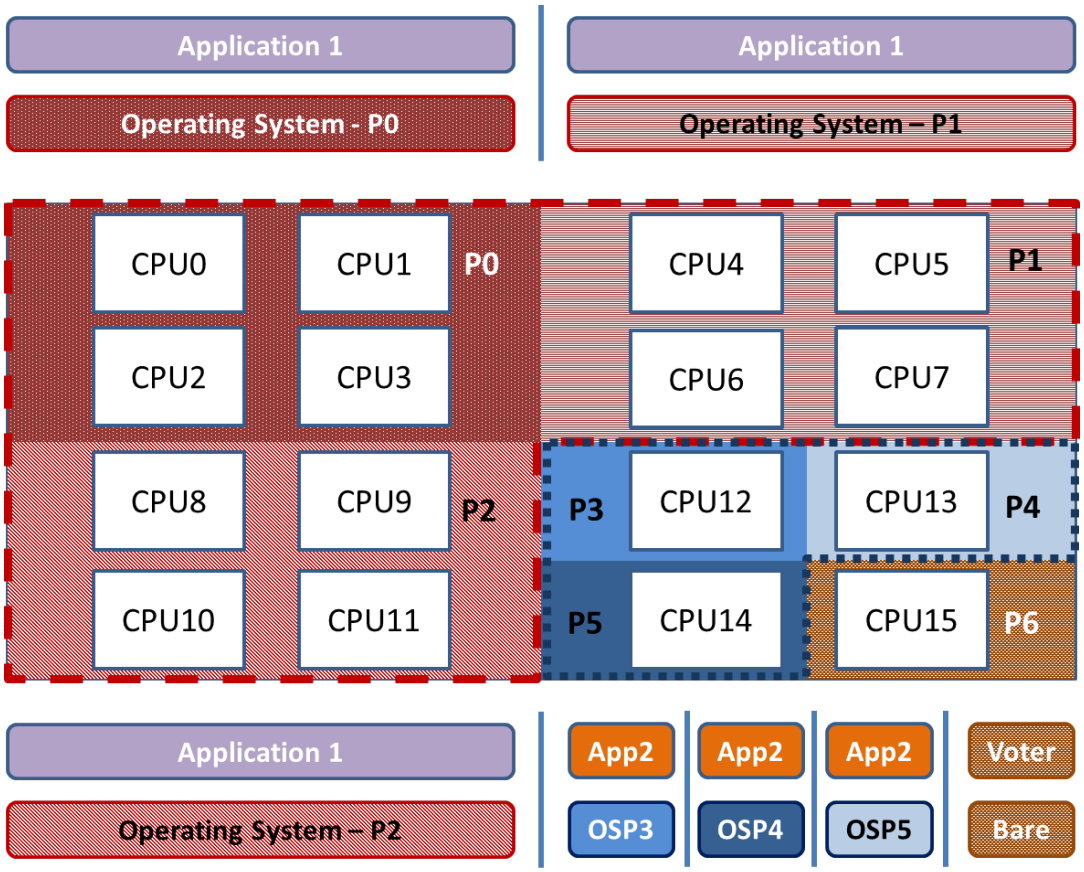

Figure 6.1: NMR-MPar Approach

Through this approach, temporal and spatial isolation of the applications are guaranteed by this type of partitioning. The security of each partition against access and data modification can be obtained by the configuration of the system.

6.2 Case-Study F: NMR-MPar implemented on the MPPA-256 processor

This case-study implements the NMR-MPar approach to improve the reliability on parallel applications running on the MPPA-256 many-core processor being N=4 and M=8. In N-modular redundancy systems, N is typically an odd integer. However, it is possible to select N as an even integer depending on extra information that this could lock-out malfunctioning systems [Sho02]. The present work takes advantage of the intrinsic architecture of the device that allows dividing the chip in two global independent hardware parts. Each part is used in *master/slave* scheme. In this case a symmetric division is proposed, so each IO cluster manages a half of the cores, thus, 8 CCs are the slaves of each IO cluster. To maximize the paralleling used and due to the fact that the minimal possible partitioning is a Compute Cluster, this case-study implements two modules per IO cluster. In this way, it is like having one system configured as a double DMR. Consequently, if one IO cluster has a dysfunction, the other is able to continue working. In addition, this division guarantees the spatial and temporal isolation for critical systems.

Another interesting point of this case-study is that the many-core processor provides a great level of security in their Processing Engines, since each CC has its private 2MB SMEM memory. It is not possible to access directly the SMEM. When an executable is loaded in memory, the only way to modify data from the outside is through the use of communication primitives of the NoC that involves the permission of the RM of the corresponding cluster.

For this case study, the NMR-MPar approach was applied to the TSP and the MM applications. Both implementations were evaluated through fault injection. Similarly to the *case-study E* described in chapter 5 section 5.3, fault injection campaigns only targeted PE registers.

6.2.1 System configuration

The system is partitioned in eight partitions. Four are multi-core partitions and the other four are mono-core partitions. Each multi-core partition is composed of one RM of the IO cluster and 4 CC running on AMP mode. The RM of the IO cluster

is configured in bare-board while the CC run the *NodeOS* with *POSIX*. Regarding the mono-core partitions, each one is configured in bare-board. This configuration uses only one quad-core of each IO cluster. Figure 6.2 illustrates this configuration.

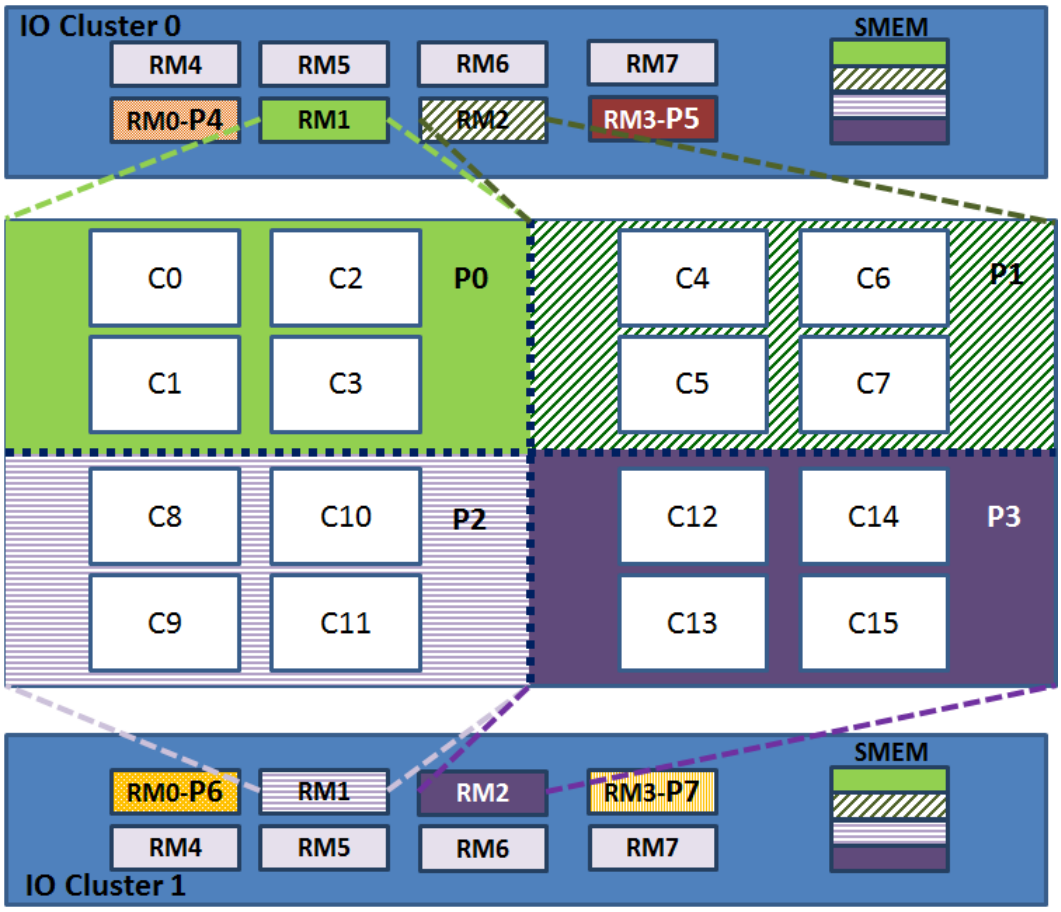

Figure 6.2: Proposed configuration for case-study F implemented on the MPPA-256

Each multi-core partition *P0 to P3* runs independently one *instance* of the proposed 4-modular redundancy. It is thus, in charged of computing the same parallel application. The RM of the IO cluster is the master of the application, while the CCs are the slaves. It means that 5 RMs (4 RM of the CC+ 1 RM of the IO cluster) and 64 PEs cores work together. The remaining RMs runs the kernel services, schedule the tasks inside each cluster, and manages the inter-cluster communication through the NoC services. All the PE cores included in the slaves (CCs) are used to compute the result. Both applications were configured without a HOST part, the multi-binary executable file was loaded to both IO clusters through the JTAG port. Figure 6.3 illustrates the general flow diagram of the 4MR-8Par application.

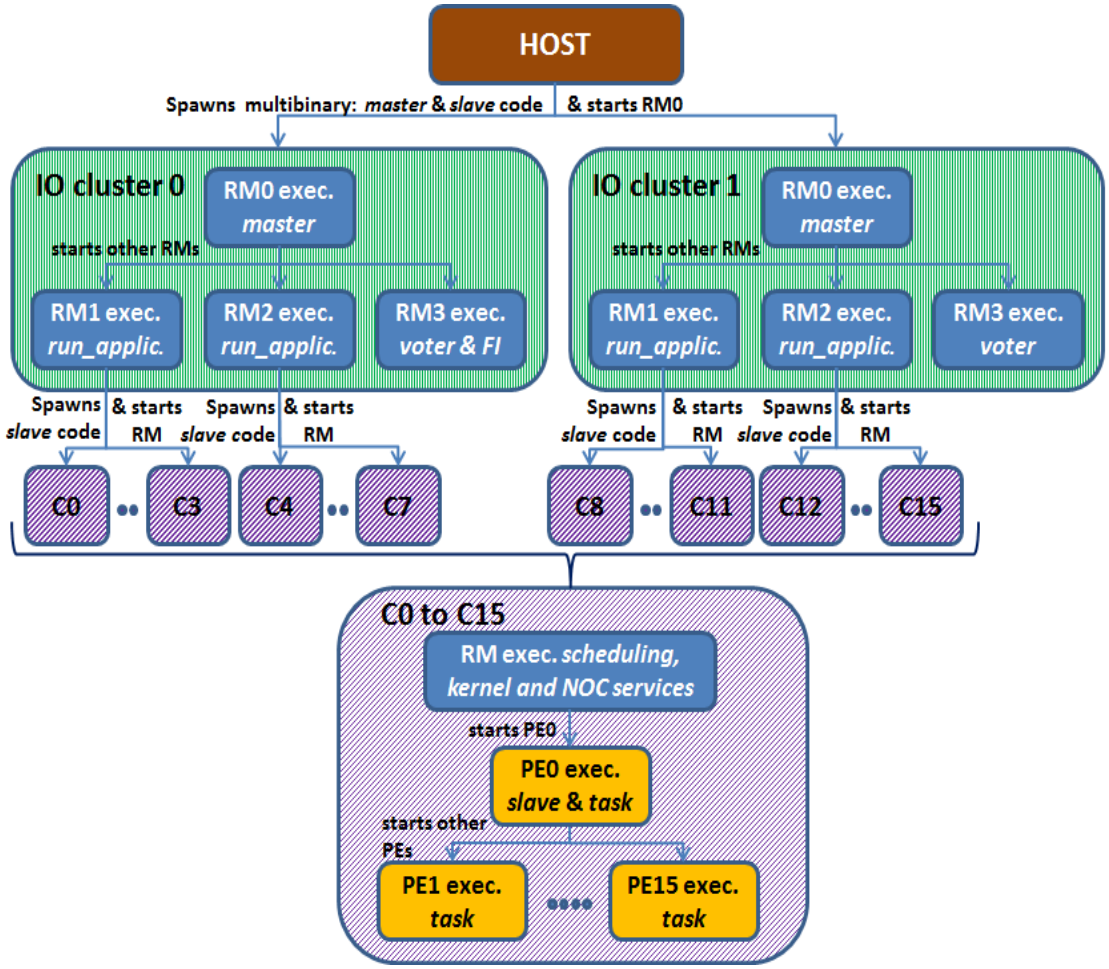

Figure 6.3: General flow diagram of the 4MR-8Par application

The *RM*0 of each IO cluster is used to configure all the MPPA connections and to interface the fault-injector and the application. The *RM*3 of each IO cluster is the *voter*. Finally, the RM3 of the IO cluster 0 is also used as the *fault-injector*.

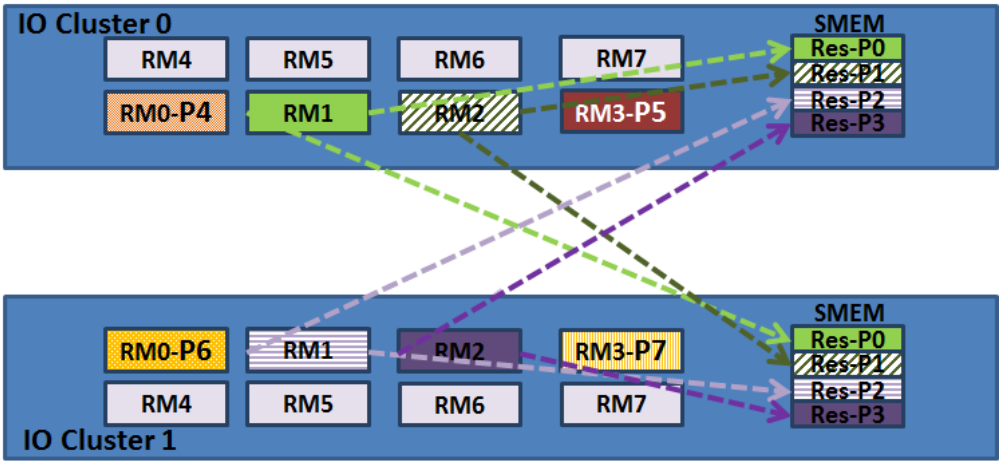

Figure 6.4: Proposed configuration for case-study F - part 2

This configuration allows having redundancy in data results, due to the fact that when each partition has solved the problem it sends its results to both IO clusters. In this way, the results are saved in two different physical locations as shown in Figure 6.4. Each voter uses its local data to apply majority voting to obtain the correct result. The *voter* that completes first the operation, logs the correct results to the HOST. The details of its implementation are shown in annexe A.

Benchmarks Applications

The NMR-MPar approach was applied to the applications detailed in chapter 5 section 5.3.2: (1) The TSP was configured to solve a 17 cities problem, and (2) the cooperative 256×256 MM iterated n times with $n = 8192$.

To validate the result of the TSP application, the verification of the response includes a valid path. This means that each city must be included in the response only once. On the other hand, for the MM application a standard Cyclic Redundancy Check (CRC) code was applied line by line of the resulting matrix. Since the size of each matrix element is 32 bits, the CRC-32 set by IEEE 802.3 was used. It was configured as follows: (1) $POLYNOMIAL$ = 0x04C11DB7, (2) $INITIAL\_REMAINDER$ = 0xFFFFFFFF, (3) $FINAL\_XOR\_VALUE$ = 0xFFFFFFFF, (4) $REFLECT\_DATA$ = YES, and (5) $REFLECT\_REMAINDER$ = YES.

The implementation of both parallel applications allows configuring the number of slaves and the problem size. Table 6.1 compares the execution time in Mega clock cycles for the TSP application running only one instance and applying the NMR-MPar approach. From this information one can see that the overhead produced by the NMR-MPar approach only depends on the number of clusters.

Similarly to the case-study described in chapter 5 section 5.3, both parallel applications were configured with 4 CCs in order to maximize the use of resources. Table 6.2 summarizes the number of MPPA inter-cluster communications for each application. The number of the connections gives an idea of the degree of complexity of the collective communications and their costs. From the casting point of view, some of them are used to broadcast information to all the slaves, other to multi-cast information to some clusters and other to uni-cast information to a specific cluster. The majority of these connections are configured in asynchronous mode so that a notify function is configured

Table 6.1: Execution-time comparison for different TSP configurations on the MPPA

Nb cities	*1 cluster*		*2 clusters*		*3 clusters*		*4 clusters*	
	Standard	*4-MR*	*Standard*	*4-MR*	*Standard*	*4-MR*	*Standard*	*4-MR*
16	30816	31372	16234	16941	11329	12183	8630	9639
17	188755	189310	102829	103537	71175	72038	58055	59060
18	521856	522411	260320	261027	188176	189035	159754	160767
Overhead	*556*		*708*		*854*		*1010*	

Note: All values are expressed in Mega clock cycles.

to interrupt the cluster when data is received. Most of them configure 4 senders and 4 receivers. One should also know that the use of *portal* primitive requires the allocation of some hardware resources. Furthermore, due to the paralleling, there are some connections trying to access the same NoC resources at the same time. Therefore, the lack of resources is frequent in default configuration [Kal13]. This was solved by changing the setup configuration of the functions and by adding delays in the application. The fact of waiting for new available resources degrades the performance of the application.

Table 6.2: NoC connections on applications running NMR-MPar on the MPPA

Application	*sync*	*portal*	*queue*
TSP	10	24	20
MM	10	24	0
Fault injector	0	16	0
Total TSP	10	40	20
Total MM	10	40	0

On the other hand, the use of resources is extremely related with the amount of data sent by the connection. The use of the NoC resources is completely different for both applications. While the MM maximizes the use of resources at the beginning and at the end of the computation, the TSP use continuously the NoC for requesting tasks and for broadcasting the *new minimum distance* and *the path* that was found.

6.2.2 Fault-injector details

The fault-injector is very similar to the fault-injector implemented in the *case-study E* that was described in chapter 5 section 5.3.3. In this case, the RM3 core of the IO cluster 0 performs fault injection in any of the 256 PE cores of the MPPA many-core processor. In order to interrupt the selected core, the *portal* primitive in asynchronous mode is used, so one *portal* per cluster (*portal_fi*) is configured for this purpose. The RM0 of IO cluster 0 is in charged of configuring the *master* part of each portal. Note that, although the device is divided in two independent global parts, the communication between clusters is always possible, so that IO cluster 0 could communicate with the 16 CCs. The portal is used to interrupt the CC that contains the selected PE core for performing the fault-injection.

Likewise, in *case-study E*, the fault injection campaigns of both applications are devoted to inject faults in the GPRs and 15 SFRs of the PEs in compute cluster's. The targeted SFRs are: (1) the 8 registers saved during context switching: SPC, SPS, RA, CS, PS , LC, LS and LE, (2) the 6 event registers, and the PMC register. The other registers could not be targeted.

6.2.3 Experimental evaluation by fault injection

The fault-injection campaigns consider the emulation of one SEU per execution. For this reason, only one bit-flip was injected in the targeted register of one of the 256 PE cores used by the NMR-MPar application. Table 6.3 provides details about the two fault-injection campaigns. For preventing the propagation of errors between successive executions, at the end of each run the HOST resets the platform and reloads the code in the many-core processor. To evaluate the effectiveness of the approach, it is essential to consider the behavior of the system in presence of an exception. When an exception occurs in one of the cores of the MPPA many-core processor, the *k1-jtag-runner* [1] takes the control. There are two possibilities:

- *With Supervision:* If the *STDOUT* of the process is a TTY, it prints the reason of the exception occurred on a given core and waits for the debugger connection while allowing the other cores, that did not halt, to continue working.

[1]Command used in the HOST to execute a process in the MPPA processor through the JTAG

- *Without Supervision:*If the *STDOUT* is not a TTY or no debugger option is passed to the *k1-jtag-runner* command, it considers that the user is not present. Hence, it does not print the exception cause and tries to continue the execution of the halted core(s).

Table 6.3: Fault-injection campaigns details of the 4MR-8Par applications running on the MPPA

Application	*Standard exec. time*		*Runs per campaign*
	[Gcycles]	*[s]*	
TSP	59.06	147.78	8328
MM	5.53	13.83	72697

If the core where the exception was produced is completely stuck, it does not respond anymore to the *k1-jtag-runner* request to continue the execution. Then, the *k1-jtag-runner* kills the process in the MPPA [Kal13].

Taking into account these two options, fault-injection campaigns consider also two scenarios: *4MR-8Par-not-tty (without supervisor)* and *4MR-8Par-tty (with supervisor)*. The main difference between them is the behavior of the system when an exception occurs.In the *4MR-8Par-not-tty* scenario, the system tries to continue immediately with the execution of the halted core, while in *4MR-8Par-tty* the system allows the other cores continue working, waits for the standard execution of the application, and then continues the execution of the halted core. Table 6.4 shows a general overview of the fault-injection campaigns on the TSP and the MM application considering the absence of supervisor. 8328 and 72696 faults were injected respectively. This significant difference in the number of injected faults is due to the larger execution time of the TSP application.

Table 6.4: Results of fault-injection campaigns on 4MR-8Par applications with no TTY

Application	*Silent faults*	*Erroneous results*	*Timeouts*	*Exceptions*
TSP	8124	0	0	204
MM	68303	0	0	4393

On the other hand, for the scenario including a TTY device, only the fault injections resulting in exceptions were re-executed with the same parameters. It was thus necessary to inject 204 additional faults in the TSP and 4393 in the MM. Under this scenario, the number of exceptions was drastically reduced to 1 and 141 respectively. The details regarding the exceptions for both scenarios are summarized in Table 6.5.

Table 6.5: Exceptions produced by fault-injection on the 4MR-8Par applications

Scenario	*Core stuck*	*Segmentation fault*	*Device exit*	*Total Exceptions*
TSP-no-tty	39	165	0	204
TSP-tty	0	0	1	1
MM-no-tty	402	3948	43	4393
MM-tty	0	95	46	141

From these results, it is possible to verify that injecting a fault with the same parameters of a previous fault injection, does not necessarily produce the same consequence. This is explained by the fact that the program is not guaranteed to be in the same conditions at the same operation cycle, due to scheduling policies and other factors related to multi/many-core processors and the execution of parallel tasks.

The considerable decrease in number of errors in the TTY scenario can be explained as follows: (1) in all the executions where a "*core stuck*" was produced, the device could find the correct result thanks to redundant instances, and (2) most of the faults producing "*segmentation fault*" were also overcame by the redundancy. Previous cases are possible since the exception does not affect the whole device, so that other cores could continue working until the HOST kill the process when the standard execution time has run out. Answering why the halted processor is not able to continue is not a trivial issue. Generally, it depends on the exception cause. There are some clues that need further investigation, such as: it could be a double trap, maybe the trap routine did not do its job, not all the causes are recoverable, or the trap routine does not have the chance to be executed.

Applying the fault-injection results to equation (3.3) allows computing the error-rate on registers for both applications. In general, erroneous results, timeouts and exceptions are considered as errors. However, in this case-study only *exceptions* were produced. Table 6.6 summarizes the obtained values.

Table 6.6: Error-Rates for 4-MoRePaR applications implemented on the MPPA

Application	$\tau_{Inj\_no\_tty}$	τ_{Inj_tty}
TSP	2.45 %	0.01 %
MM	6.04 %	0.19 %

6.3 Overall result comparison

The MPPA many-core processor running massive parallel applications on POSIX was evaluated trough different scenarios: (1) the application without redundancy, (2) the application with NMR-MPar approach without TTY, and (3) the application with NMR-MPar approach with TTY. This section presents a comparison between the fault-injection results obtained for the first scenario in the *case-study E* described in chapter 5 section 5.3.4, and the results obtained from the application of the 4MR-8Par approach on the same applications: (1) the TSP -17 cities, and (2) the 256×256 Matrix Multiplication iterated $n = 8192$. Figure 6.5 illustrates the SEE consequences.

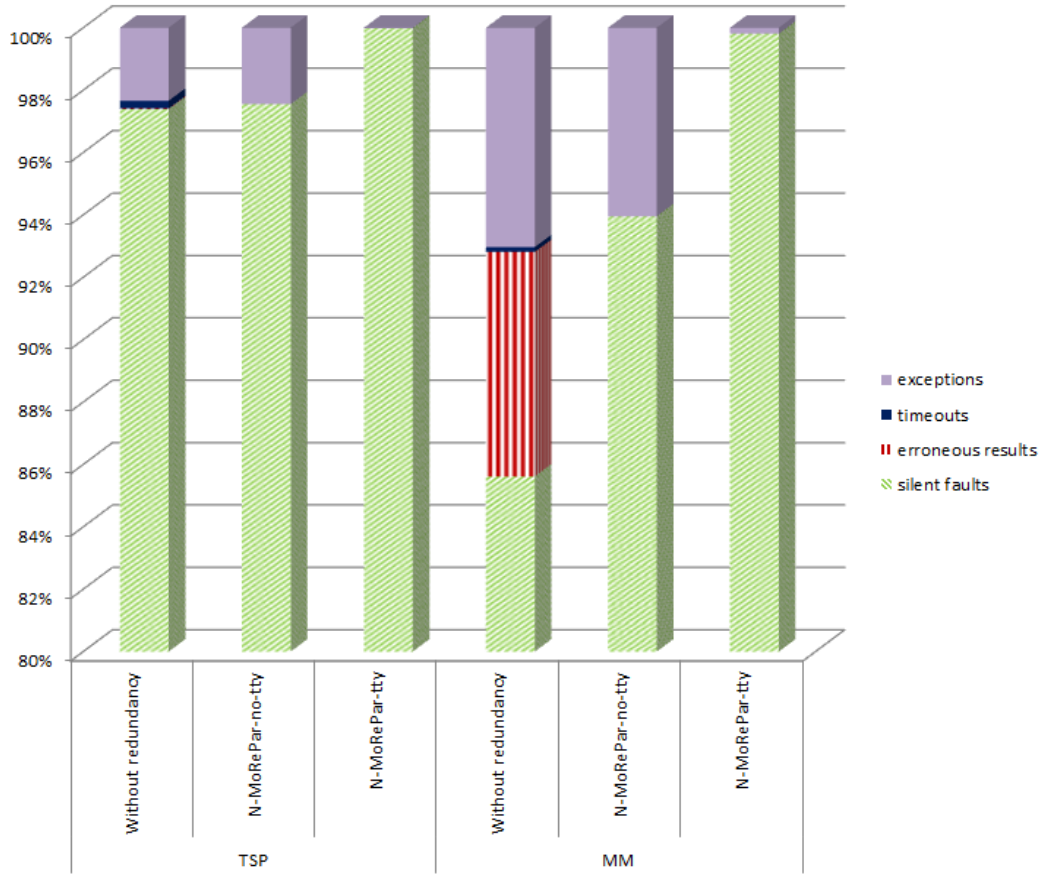

Figure 6.5: Comparison of SEE consequences per scenario in NMR-MPar applications

Regarding the error-rate values obtained by fault-injection, there is a potential underestimation since not all sensitive zones were targeted for the reason mentioned in chapter 5. However, taking into account that authors from [TS12] state that typically GPRs are frequently used and thus more susceptible to errors than SFRs which remain mostly unused, this underestimation could be minimal. Similarly to case-study E, for obtaining the failure rate the equation 4.4 was applied.

$$\lambda_{CEU} = \tau_{Inj} \times \sigma_{Static} \times \phi \qquad \text{(4.4 revisited)}$$

The $\sigma_{Static} = 12.71 \times 10^{-9} cm^2/device$ was obtained from work [VRR+16]. Table 6.7 summarizes the estimated failure rates per hour at avionique altitude for both applications under three scenarios: (1) implemented without fault-tolerant approach [2], (2) implemented with NMR-MPar running with TTY, and (3) implemented with NMR-MPar running without TTY device.

Table 6.7: Failure rates per hour at 35000 feet for 4MR-8PaR applications implemented on the MPPA

Application	*without_FT*	*no_tty*	*tty*
TSP	9.93×10^{-7}	9.32×10^{-7}	3.80×10^{-9}
MM	5.47×10^{-6}	2.30×10^{-6}	7.23×10^{-8}

The comparison of these values to those provided by the avionics standard DO-178B Software Considerations in Airborne Systems and Equipment Certification [3] allows having an idea of the potential use of the approach in the avionic domain depending on the criticity of the tested application.

From the failure rates, the reliability of the MM application under the three mentioned scenarios was obtained. Figure 6.6 compares the results.

[2] Case-study E described in chapter 5

[3] DO-178B defines the failure rates per hour required for certifying avionics applications with different criticity.

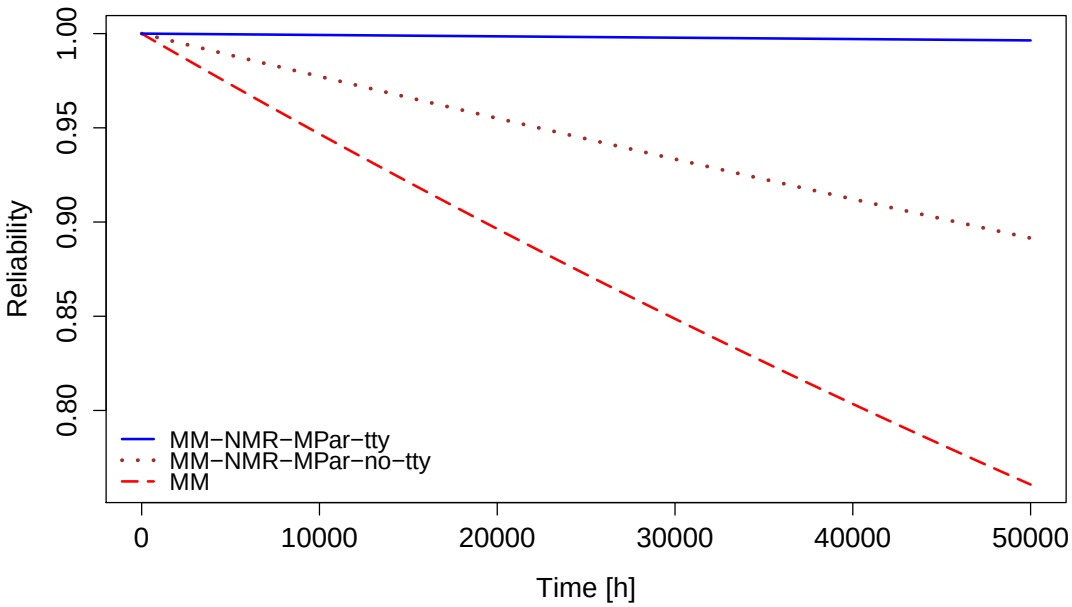

Figure 6.6: Evaluation of reliability for MM application at 35000 ft

A similar comparison was done for the TSP application. Figure 6.7 illustrates the reliability for the three scenarios. From both results, it is possible to confirm the important improvement on the reliability by the use of the NMR-MPar approach.

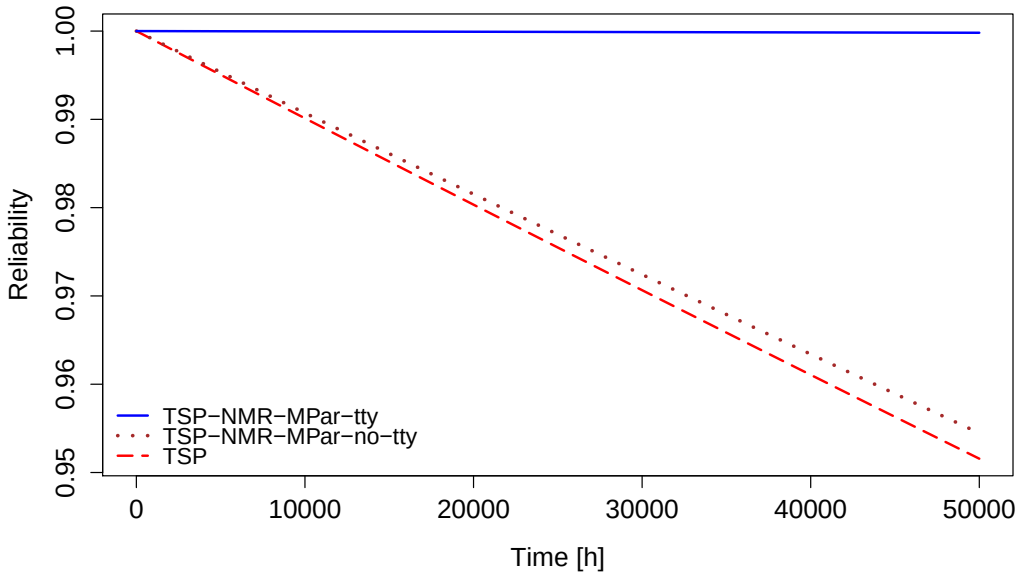

Figure 6.7: Evaluation of reliability for TSP application at 35000 ft

In order to compare the improvement on the reliability of both applications, the best case was considered. Considering the implementation of both applications with NMR-MPar running with TTY scenario, the reliability of the TSP is 0.9998 at 50000 h while

the corresponding in the MM case is 0.9964 [4].

Regarding the FIT value referenced at New York city ($14n\dot{c}m^{-2}\dot{h}^{-1}$), Table 6.8 summarizes the results for both applications.

Table 6.8: FIT values at NYC for 4MR-8PaR applications implemented on the MPPA

Application	*without_FT*	*no_tty*	*tty*
TSP	4.64	4.36	0.02
MM	25.59	10.75	0.34

6.4 Concluding remarks

The NMR-MPar is a generic software-approach developed for improving the reliability of the applications running in multi/many-core processors by using spatial redundancy. It can be easily implemented in other multiprocessors being the implementation specific for each device.

It can be proved that there is a significant decrease of application error-rate about two orders of magnitude when implementing the NMR-MPar approach on the MPPA many-core processor. However, considering the underestimation of the error-rate obtained by fault-injection campaigns, it is advisable to evaluate the approach under neutron radiation.

Taking into account the improvements obtained by applying the N-MoRePAR to the benchmarks, and considering that: (1) there were no *erroneous results* and *timeouts* occurred, (2) all the *errors* were produced by *exceptions*, (3) all the *exceptions* were detected, and (4) *exceptions* can be easily managed by the system through a reset, this approach is very useful for applications that do not have a critical time constraint. Its reliability could be improved by implementing Software Implemented Fault Tolerance (SIFT) techniques on application level. For instance, the results shows that one of the most critical registers is the return address. Consequently, it could be useful to combine this approach with others such as [SAL+08] that proposes selective instruction replication that protects address calculations and conditional branches to reduce segmentation faults or loading incorrect data values.

[4] Only for reference, typical required reliability for space mission applications is > 0.9995 [VRSCH11].

Chapter 7

Related work

This chapter summarizes the most relevant works related to the main topics covered in this thesis. The literature is grouped in two main subjects regarding the SEE sensitivity evaluation and the reliability of multi-core and many-core processors. Concerning the reliability issue, two axes were taken into account: fault-tolerance by software redundancy and fault-tolerance by partitioning. The contribution to the state-of-the-art done during the development of this thesis is also presented.

7.1 SEE sensitivity of multi-core and many-core processors

In the literature, there are a few works dealing with the SEE sensitivity of multi/many-core processors. The research done in [JAC+02] compares the errors issued from fault injection at software level on a single core using Linux, with the errors provided by a Stanford real fault database. This work provides useful guidelines for similar works on multi-cores. Reference [MSM+02] evaluates a COTS system based on two PowerPC750 processors and LynxOS by fault injection. The evaluation comprises OS and user level applications. It concludes that the easiest transient faults to deal with are those related to OS due to its tendency to crash system.

The work presented in [JSKI08] describes the first step towards a framework to evaluate operating system's behavior under faulty conditions (single or multiple bit flips) in multi-core environment. A fault-injection framework is proposed in [LPC+12] for support-

ing dependability analysis of multicore systems. It takes advantage of the error reporting architecture that includes the *machine-check error registers*. It emulates machine-check errors and analyzes how the system responds to them.

Reference [Gue12] presents the SEE test results under 15 and 25 MeV ions of the 49-core Maestro ITC microprocessor. Maestro many-core is a RHBD processor based on the Tilera TILE64 processor intended to be used in space applications. Radiation tests targeted L1 and L2 cache memories as well as registers of the tile core. Results have demonstrated that the L1 and L2 caches are handled by an effective Error Detection And Correction (EDAC) mechanism comprised in the Maestro design.

In [SN12] is presented a relevant work that establishes a dynamic cross-section model for a multi-core server based on quad-core processors built-in 45nm bulk CMOS technology. In addition, it provides a fault handling comparison between Windows 5.2 and Linux 5.1 operating systems.

Reference [ORQ+14] presents the radiation sensitivity evaluation of a modern Graphic Processing Units (GPUs) designed in 28nm technology node, and composed by an array of streaming multi-processors which share the L2 cache memory. It also provides a hardening strategy based on Duplication with Comparison.

7.2 Reliability in multi-core and many-core processors

Since performance and reliability are significant characteristics of systems based on multi-core and many-core processors, it is essential to propose techniques for improving the reliability while keeping the minimum degradation of system performance. Reference [KA11] discusses the possibility of using many-core processors on space applications. They implement a real space sensor application: the third Scalable Synthetic Compact Application (SSCA3) on a platform based on Nehalem CPUs Intel Xeon 6 cores to prove the relevance of using many-cores on space applications.

Reference [Huy12] explores the possibility of using multi-core processor to implement an application that satisfies the standard Avionics Application Standard Software Interface (ARINC) 653. An analysis of both AMP and SMP configurations is considered. It stands out the capabilities of shared resource contention and module-level health monitoring for the AMP case, and concurrency for the SMP case. The author also notes that

the needed preemption could diminish the usage of multi-cores.

Authors of [RRV03] propose to disable cache memories of high-end processors in safety-critical applications in order to gain in reliability in spite of the increase of the execution time. This work also presents an analysis of the effects of soft-error in data and instruction cache memories.

A more recent work [SRN+14] demonstrates that by enabling the L1 cache, it is possible to improve the performance of the system without compromising the reliability. They introduce a generic metric Mean Workload Between Failures (MWBF) for evaluating the reliability of a embedded processor devoted to execute safety-critical applications. This metric takes into account both cross-section and exposure time.

7.2.1 Fault-tolerance by redundancy

There are significant works regarding fault tolerance which include redundant multi-threading techniques for transient fault detection and recovery for multicore processors working on SMP mode [VPC02, BI06, MAAB13, RM00, MKR02]. Some of them are based on Simultaneous multithreading (SMT) where the systems run identical copies of a process as independent threads. However, in these works details about the validation of the proposed techniques by fault-injection campaigns are not provided. Moreover, the principle of redundancy in multi-core processors is used to recover the system from permanent hardware faults [HRI+16],

In reference [SBM+09] is presented a software technique for transient-fault tolerance: the Process-Level Redundancy (PLR), which leverages multiple cores for low overhead. This approach is evaluated by SEU fault-injection on the redundant processes. The faults are injected by means of the Intel tool called *Intel Pin dynamic binary instrumentation* which changes one bit of a randomly selected instruction.

Authors of [WKS+11] implement four independent fault-tolerance techniques on the RHBD Maestro processor aiming at mitigating hardware and software error. This work describes the implementation of these techniques that include a kernel-level checkpoint/ rollback, a process and thread level redundancy and a heartbeat. Nonetheless, there is no evaluation of the effectiveness of their proposal.

Reference [BMS12] proposed a reliable resource management layer applied to many-core architectures. The scheduler applies a TMR on multi-threading applications, detects

error and isolates faulty processing elements. This approach is evaluated at simulator level by injecting one fault per run. The evaluation was done on a single cluster of 16 cores.

Authors from [AML+15] proposes a framework for improving the reliability of COTS many-core processors. The framework is illustrated through an adaptive N-Modular Redundancy (NMR) system that selects the better number of replicas to maximize the system reliability. For achieving it, they present an analysis of the reliability of the system that shows a trade-off between the reliability of the voter and the number of replicas. This analysis is only theoretical and has not been evaluated.

Finally, reference [KS12] proposes a Soft NMR that improves the robustness of classical NMR systems by using error statistics. It effectiveness is illustrates by an example in image coding.

7.2.2 Fault-tolerance by partitioning

Reference [LNT14] recaps the main challenges of the migration from federated systems to IMA in avionics and gives some guidance in temporal partitioning. [MEA+10] proposes temporal isolation between tasks for scheduling mixed-criticality works on multi-core processors.

Authors of [PnZ+14] introduce the concept of parallel Software Partition (pSWP) supported on a Guaranteed Resource Partition (GRP) applied to many-cores. Their proposal consists in defining a set of physical resources where the pSWP runs in order to guarantee the time isolation for IMA systems. The proposal is evaluated in a simulator compatible with PowerPC ISA.

The work presented in [TCAP14] implements multiple partitions with rigorous temporal and spatial isolation. This approach supports mixed-criticality based on the XtratumM virtualization layer. A methodology and tools are described.

Reference [GPm+16] is probably the most related work to the NMR-MPar approach proposed in this thesis. Authors propose an hybridization between duplex execution at task level on COTS and time and space partitioning provided by the bare-board hypervisor XtratuM [1]. The authors have implemented their proposal on a ZYNQ processor that hosts an ARM dual-core Cortex A9. The main differences between this work and our

[1] Xtratum OS a bare-metal hypervisor owned by Fentiss

approach are: (1) redundancy in NMR-MPar approach is totally independent at system and user level, (2) in NMR-MPar there is an independent hypervisor for each partition, (3) our proposal has been implemented on a many-core processor and, (4) our proposal has been evaluated by fault-injection.

7.3 Contributions to the state-of-the-art

Concerning the evaluation of SEE sensitivity and the improvement of reliability on COTS systems based on multi/many-core processors, this thesis contributes to the state of the art as follows:

- In work [RVB+16], the evaluation of the dynamic sensitivity of different scenarios running on the P2041 multi-core was presented. From the results, it can be seen that the dynamic sensitivity of the device strongly depends on the multi-processing mode used. In addition, this work illustrates that the 45nm SOI quad-core processor is about four times less sensitive to SEE that its CMOS counterpart.

- The work [VRR+16] presents the 14 MeV neutron sensitivity of the MPPA-256 many-core processor. The results suggest that ECC and interleaving implemented in the SMEMs of the clusters are very effective to mitigate SEUs consequences as all detected events of this type were corrected. Furthermore, the evaluation of the device dynamic response shows that by enabling the cache memories, it is possible to increase the performance of the application without compromising reliability.

- The results presented in chapters 4 and 5 illustrate the response of different scenarios implemented on COTS multi/many-core processor face to SEE transient consequences. From the results, it was possible to evaluate the effectiveness of the protection mechanisms implemented by manufacturers in such devices. Moreover, it was possible to observe how the algorithm and the programming model affect the reliability of the system. Besides, the evaluation under neutron radiation of COTS system based on multi-core processors at OS and user level was presented.

- Chapter 6 describes NMR-MPar approach, the first work implementing redundancy and partitioning on a many-core processor to improve its reliability. The obtained

results by fault-injection are very promising. For instance, it is possible to observe from figures 6.6 and 6.7 that the worst case system reliability (at $t = 50000h$) were increased from 0.95 to 0.9998 for the TSP, and from 0.75 to 0.9964 for the MM. Therefore, the results persuade the usage of COTS many-core processor in mixed-criticality systems. However for avionics and spacecraft applications, complementary techniques at application level must be adopted to mitigate the *exceptions* that affect their reliability.

- References [CFV+15, CFV+16] aim at evaluating the sensitivity to neutron radiation on devices with emerging technologies. They present the SEE sensitivity of advanced low-power SRAM Renesas memories operating at low bias voltage. The results show that at low voltage, clusters of bit-flips and hard errors can be produced.

- The literature proposes several approaches to improve the reliability through redundancy and partitioning criteria. However, only very few works evaluate them through simulation or fault-injection. This research uses fault injection based on the CEU approach as an evaluation mean for the present proposal for many-core processors.

- Lastly, different variations of the CEU fault-injection method were developed. These variations include a fault-injector based on fork principle, a fault-injector on bare-board based on inter-processor interrupts, and a fault-injector for distributed systems based on NoC and inter-processor interrupts [VRM+14, VRV+15]. The *fault-injectors* also served as *monitor applications* to end the application in case of *time-outs*.

Chapter 8

Conclusions and perspectives

The widespread use of COTS multi-core and many-core processors in computing systems is a consequence of their flexibility, low-power consumption, intrinsic redundancy and high performance. However, there are some dependability issues that have to be overcome. Indeed, the reliability, availability and security of systems based on these devices are reduced due to their increasing complexity, integration scale and sensitivity to natural radiation. Consequently, manufacturers, industrial and academic partners are working together to improve their characteristics to permit its usage in critical embedded systems.

In this context, the work done throughout this thesis provides valuable information about the SEE sensitivity of applications running on multi/many-core processors. The proposed methodology based on multiple case-studies allowed investigating the impact of SEEs on COTS systems based on multi/many-core processors under diverse scenarios. The design of the case-studies has taken into account a wide range of possible configurations in terms of multi-processing mode, programming model, manufacturing technology, number of cores, memory model, and used resources. The obtained results can serve as a helpful guideline to developers for choosing the most reliable system configuration according to their needs. In addition, the proposed fault-tolerant approach reduces significantly the impact of SEE sensitivity and opens up the possibility of its adoption in mixed-criticality systems.

8.1 Contributions

The evaluation done on the Freescale P2041 multi-core under neutron radiation has proved that despite of SOI manufacturing technology and the protection mechanisms implemented on memory, erroneous results occurred. Additionally, results confirm the sensitivity dependence on software environment. It has been demonstrated that the AMP mode is more reliable than the SMP mode. Also, one can verify that the intrinsic application characteristics play an important role on the system sensitivity. In addition, it can be observed a considerable vulnerability of the OS specially concerning system crashes and exceptions. This vulnerability has put in evidence that the implementation of redundancy at user level is not enough to overcome dependability issues. Therefore, it is clear the imperative necessity of using the partitioning concept to guarantee both reliability and availability of the system.

On the other hand, the evaluation under neutron radiation of the implemented scenarios on the KALRAY MPPA-256 many-core processor has proved the effectiveness of the protection mechanisms implemented on its memories. Furthermore, it was also possible to determine the pertinence of enabling the cache memories of this device for improving its performance with minimum reliability consequences. These results also corroborate the SEE sensitivity dependency on software environment. Moreover, through the implementation of parallel applications in the many-core processor, it was possible to confirm that the migration of an application to a distributed system communicating by NoC services is a non trivial task.

Furthermore, one of the mainstay for migrating from uni-processors to multi/many-core processors in criticality systems is the possibility to take advantage of the inherent redundancy to implement fault-tolerance by replication. As expected, the implementation of the TMR on the quad-core has improved significantly the system reliability. However, it has reduced its parallel computing capability. Nevertheless, the redundancy applied to the many-core processor is a promising solution.

The NMR-MPar approach, developed during this thesis, based on NMR and partitioning concepts has proved to be very efficient to improve the reliability on applications running on many-core COTS processors. The obtained results show that only *exceptions* cannot be completely overcome by this approach. This issue can be surpassed by

implementing SIFT tecniques on application and OS level. For instance, [SAL+08] proposes selective instruction replication that protects address calculations and conditional branches to reduce segmentation faults or loading incorrect data values. Therefore, by combining NMR-MPar with complementary SIFT techniques, it will be possible to use multi-core and many-core COTS processors in harsh radiation environments

8.2 Future Works

Regarding future directions, this work opens up various possibilities to continue the investigation in the following areas:

- *Evaluate the NMR-MPar approach under particles radiation.* This step is indispensable to ratify the results obtained from fault-injection. Furthermore, neutron and heavy-ion radiation testing are suitable to validate its applicability to avionics and spacecraft applications.

- *Conjugate this approach with other SIFT techniques.* The implementation of fault-tolerant techniques at OS and user level will deal with the main concern of NMR-MPar approach, the *exceptions*. Some of these techniques comprise compiler techniques and replication of instructions, registers and cache-lines. It is, thus interesting to improve the reliability of the system by combining NMR-MPar with other fault-tolerant approaches.

- *Evaluate NMR-MPar on other system configurations.* Benefiting of the large number of cores in the selected many-core processor, it is suitable to evaluate other systems configurations. For instance, the implementation of two independent TMR running different applications. Also, it can be interesting to implement this approach on other platforms to evaluate its broad applicability on systems based on multi/many-core processors.

- *Appraise its overhead in terms of energy and power consumption.* This is essential to boost its usage on embedded and critical systems.

Annexe

A Implementation details of case-study F

This Annexe presents the details regarding the implementation of the code for the applications implementing the N-MoRePar approach. The pseudo-code of the *master* code that runs on each IO cluster of the MPPA is shown in figure 8.1.

Each IO cluster manages two instances (*NUMBER_LOCAL_IO_INSTANCES*) of the 4-MoRePar implemented on the MPPA many-core processor. Nevertheless, it configures a memory space for saving the results of the four instances (*RESULT [NUMBER_INSTANCES]*). Some details of *main* function of *master* code that runs on the RM0 of each IO cluster are listed in the following:

- The function *create_mppa_connections_for_instance* creates the *portal* and *queue* connections used for the inter-cluster communication of each instance of the application that was detailed in chapter 5 section 5.3.2.

- The function *create_fault_injection_portal_connection* allows opening a channel of communication if fault-injection is performed.

- The function *create_mppa_compl_connections* creates: (1) two *sync* primitive for synchronization with the other IO cluster, (2) two *portals* for sending and receiving the results of each *LOCAL_IO_INSTANCE* to/from the other IO cluster, (3) two *portals* for sending and receiving the results of the voter and the *program_ended* variable to/from the other IO cluster.

- Due to the possibility of having a crash in a part of the system, the application used the principle of *timeout*, so if the execution time is greater than the standard

execution time, the main application continues with the next stage. The different stages of the *main* function are outlined in Figure 8.2.

```
global fault_injection_variables, portals, queues, program_ended

procedure MAIN_SLAVE (nb_threads, application_variables, nb_clusters)
  WAIT_BARRIER( )
  RUN_APPLICATION(nb_threads, application_variables, nb_clusters)
  SEND_RESULTS_OTHER_IO(portal)
  WAIT_FOR (VOTER_OR_TIMEOUT )
  WAIT_FOR(program_ended OR TIMEOUT)
procedure MAIN_VOTER_FAULT_INJECTOR
  WAIT_BARRIER( )
  while program_end != OR NO_TIMEOUT
    do { if cycles_applic == rand_inst AND io_cluster_0
         then { INJECT_FAULT( )
         if every_instance_finished OR TIMEOUT
           then { TASK_VOTER( )
                  SEND_VOTER_RESULTS_TO_OTHER_IO (portal)
                  if other_io_not_log
                    then { LOG_RESULTS( )
  WAIT_FOR(program_ended OR TIMEOUT)
procedure MAIN (fault_injection_parameters, application_variables, nb_clusters)
  for i ← 1 to NUMBER_INSTANCES
    do INITIALIZE_RESULTS(i)
  for i ← 1 to NUMBER_LOCAL_IO_ INSTANCES
    do CREATE_MPPA_CONNECTIONS_FOR_INSTANCES(i,port,nb_clusters )
  if fault_injection = 1
    then { fault_injection_variables ← fault_injection_parameters
           CREATE_FAULT_INJECTION_PORTAL_CONNECTIONS()
  CREATE_MPPA_COMPL_CONNECTIONS( )
  START_RM( 1, & MAIN_SLAVE )
  START_RM( 2, & MAIN_SLAVE )
  START_RM( 3, & MAIN_VOTER_FAULT_INJECTOR )
  WAIT_BARRIER( )
  WAIT_FOR (END_RUN_APPLIC_IN_SLAVES OR TIMEOUT )
  WAIT_FOR (VOTER_OR_TIMEOUT )
  program_ended ← 1
  SEND_TO_OTHER_IO (program_ended)
  WAIT_FOR(program_end_other_io OR TIMEOUT)
  CLOSE_MPPA_CONNECTIONS( )
```

Figure 8.1: Pseudo-code for the implementation of the N-MoRePar applied to the MPPA

Regarding the *main_slave* code executed by the RM1 and the RM2 of each cluster, the details of the code of the function *run_application* were presented in chapter 5 section 5.3.2.

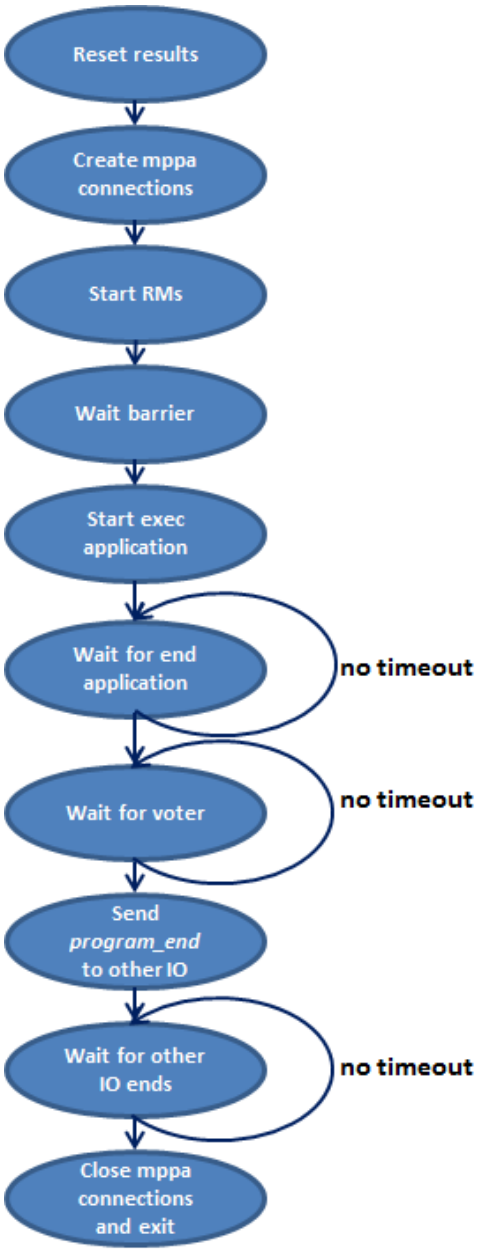

Figure 8.2: Flow diagram of *main* function run in RM0

On the other hand, concerning the function *main_voter_fault_injector* it is important to note that only the RM3 of IO cluster 0 performs the function *inject_fault* at the random instant *rand_inst*. The voter applies majority voting criteria; the correct response can be determined if 3 or 4 results are equal. Also, it is possible to establish the correct response if 2 results are equal and the other 2 are different between them.

However, if there are 2 pairs of 2 equal results, the voter logs an error. It is important to note that the result include the response of the application plus the validation data of the result.

B Publications during this thesis

Journals

1. Vargas, V., Ramos, P., Ray, V., Jalier, C., Stevens, R., Dupont de Dinechin, B., Baylac, M., Villa, F., Rey, S., Zergainoh, N. E., Méhaut, J. F. and Velazco, R., "Radiation Experiments on a 28nm Single-Chip Many-core Processor and SEU error-rate prediction", *IEEE Trans. Nucl. Sci.*, vol.64, no 1, pp . 483-490 , January 2017.

2. Ramos, P., Vargas, V., Baylac, M., Villa, F., Rey, S., Clemente, J. A., Zergainoh, N. E., Méhaut, J. F. and Velazco, R., "Evaluating the SEE Sensitivity of a 45 nm SOI Multi-Core Processor Due to 14 MeV Neutrons", *IEEE Trans. Nucl. Sci.*, vol. 63, no 4, pp. 2193-2200, August 2016.

3. Clemente, J. A., Franco, F., Villa, F., Baylac, M., Ramos, P.,Vargas, V., Mecha, H., Agapito, J. and Velazco, R., "Single Events in a COTS Soft-Error Free SRAM at Low Bias Voltage Induced by 15-MeV Neutrons", *IEEE Trans. Nucl. Sci.*, vol. 63, no 4, pp. 2072-2079, August 2016.

Conferences

4. Vargas, V., Ramos, P., Ray, V., Jalier, C., D. de Dinechin, B., Baylac, M., Villa, Rey, S., Minguez, B. and Velazco, R., "First Results of Radiation Experiments on a 28nm Single-Chip Many-core Processor", *IEEE 53rd Nuclear and Space Radiation Effects Conference (NSREC)*, Portland, Oregon, July, 2016.

5. Vargas, V., Ramos, P., Velazco, R., Méhaut, J. F. and Zergainoh, N. E., "Evaluating SEU fault-injection on parallel applications implemented on multi-core processors", *In Proc. 6th Latin American Symposium on Circuits & Systems (LASCAS)*, February 2015, DOI:10.1109/LASCAS.2015.7250449.

6. Ramos, P., Vargas, V., Baylac, M., Villa, F., Rey, S., Clemente, J. A., Zergainoh, N. E., Méhaut, J. F. and Velazco R., "Sensitivity to Neutron Radiation of a 45 nm SOI Multi-Core Processor", *In Proc. Radiation and its Effects on Components and Systems (RADECS)*, pp. 135-138, September 2015.

7. Clemente, J. A., Franco, F., Villa, F., Baylac, M., Ramos, P., Vargas, V., Mecha, H., Agapito, J. and Velazco, R., "Neutron-Induced Single Events in a COTS Soft-Error Free SRAM at Low Bias Voltage", *In Proceedings of Radiation and its Effects on Components and Systems (RADECS)*, pp 162-165, September 2015.

8. Vargas, V., Ramos, P., Mansour, W., Velazco, R., Zergainoh, N. E. and Méhaut, J. F., "Preliminary results of SEU fault-injection on multicore processors in AMP mode", *In Proc. 20th International On-Line Testing Symposium (IOLTS)*, pp. 194-197, September 2014.

Résumé de la thèse

i. Introduction

Contexte et motivation

De nos jours, les avancées dans les technologies de fabrication de circuits intégrés permettent de disposer de processeurs à multiples cœurs qui offrent une grande capacité de calcul grâce à la mise en œuvre d'un parallélisme massif dans un budget limité en consommation énergétique. Ces dispositifs offrent également une grande flexibilité car ils permettent l'implémentation de différents modes d'exécution parallèle (*multi-processing*), de plusieurs paradigmes de programmation et d'une grande variété d'applications. En outre, leur architecture est basée sur une redondance des ressources, ce qui les rend idéales pour la mise en œuvre de mécanismes de tolérance aux fautes. Le constat est qu'aujourd'hui la plupart des architectures informatiques utilisent des processeurs multi-cœur et many-cœur comme solution standard pour répondre à la demande croissante en performance et en fiabilité, sans augmentation critique de la consommation d'énergie.

Par exemple, dans le domaine du calcul haute performance (*High Performance Computing, HPC*), les deux premières plates-formes du classement Top500 (novembre 2016) sont basés sur des architectures multi ou many cœurs. Par ailleurs, il existe un intérêt croissant pour l'utilisation de processeurs multi-cœur et many-cœur dans des systèmes embarqués et critiques. La tendance à favoriser l'utilisation de composants généralistes et/ou commerciaaux dans les systèmes spécifiques est principalement motivée par des coûts réduits de conception en utilisant des composants standard plutôt que des composants spécialisée, et grâce à la réduction significative du rapport performance vs. consommation d'énergie par l'utilisation de composantes commerciales au lieu de composants résistant aux effets de radiation [MSM+02, KSD+97].

Actuellement, l'utilisation de processueur multi-cœur est largement répandue dans les systèmes embarqués mais lorsque des tâches critiques sont nécessaires, seul un cœurs est utilisé [MEA+10]. Par conséquent, plusieurs projets de recherche internationaux associant des partenaires industriels et universitaires comme ACROSS, ARTEMIS - EMC2, CAPACITES, DECOS, MultiPARTES explorent le développement de nouvelles solutions qui permettent l'utilisation de multi-processeurs dans des systèmes critiques. Ainsi, les industries de l'avionique et spatiales sont intéressées à valider l'utilisation de ces composants pour leurs applications (par exemple, des équipes internationales travaillant sur ces questions sont: CAST, EASA, projet RECOMP, etc.).

Avoir des puces plus performantes nécessitent l'amélioration du processus de fabrication de circuits en augmentant l'intégration d'échelle. Malheureusement, le degré de miniaturisation rend ces circuits potentiellement plus sensibles aux effets des radiations naturelles. L'impact des radiations est aggravé par la complexité de la puce et la quantité énorme de cellules mémoire qu'ils contiennent. Pour cette raison, les concepteurs d'architectures sont continuellement à la recherche de nouvelles méthodes pour améliorer les technologies de fabrication afin de réduire les conséquences des erreurs produites par l'impact d'une particule dans une zone sensible de la puce.

En outre, les fabricants ont mis en place des mécanismes de protection tels que le contrÃťle de parité et des codes de correction d'erreurs (ECC) dans les cellules mémoires des composants. Par ailleurs, certaines techniques d'atténuation basées sur des approches matérielles et logicielles sont proposées dans la littérature pour minimiser les conséquences de ce phénomène. Malgré les efforts de fabrication, il existe certaines zones qui restent particulièrement vulnérables aux effets des radiations naturelles.

Plusieurs travaux pertinents traitant de la sensibilité des composants électroniques sont disponibles dans la littérature scientifique. Cependant, il existe très peu de travaux traitant de la sensibilité aux effets de radiation des processeurs multi-cœur et many-cœur. Par conséquent, il est obligatoire d'évaluer dans quelle mesure cette sensibilité affecte la fiabilité des applications fonctionnant sur de tels circuits. En outre, en raison du parallélisme massif utilisé pour améliorer les performances des systèmes informatiques, l'évaluation de l'impact sur les applications parallèles devient essentielle. La tolérance aux fautes pour les systèmes parallèles et distribués a été largement étudiée au niveau du logiciel. Cependant, les études concernant la tolérance aux fautes de systèmes sur puce

sont limitées. Profitant de la multiplicité de cœurs de processeurs multi/many-cœur, il est tout à fait envisageable d'implémenter des techniques de redondance pour améliorer la fiabilité des applications.

Objectifs de la thèse

Dans ce contexte, les objectifs de cette thèse sont:

1. Évaluer la sensibilité face aux SEE de différentes applications parallèles implémentées sur des processeurs multi/many-cœur.

2. Évaluer l'impact du modèle de programmation et du mode d'exécution (*multi-processing*) sur la sensibilité SEE du système.

3. Développer et évaluer une approche basée sur la redondance et le partitionnement appelée NMR-MPar pour améliorer la fiabilité des applications parallèles tournant sur les processeurs multi/many-cœur.

Contexte de préparation de cette thèse

Cette thèse a été préparée dans le cadre d'une collaboration scientifiques entre deux équipes de recherche de l'Université de Grenoble-Alpes:

1. L'équipe RIS du laboratoire TIMA. Cette équipe RIS possède une solide expérience en matière de fiabilité des circuits et des systèmes

2. l'équipe INRIA CORSE du laboratoire LIG. L'équipe CORSE a une longue expérience sur l'optimisation des compilateurs et des environnements d'exécution pour les processeurs multi-cœur et many-cœur.

La synergie produite par le savoir-faire des deux équipes a été fondamentale dans le développement et la réussite de cette thèse.

Organisation du manuscrit

La suite du document est organisée de la manière suivante. Le chapitre 2 décrit les concepts scientifiques et technologiques nécessaires à la compréhension de cette recherche.

La méthodologie et les outils utilisés pour évaluer l'impact des effets des radiations naturelles sur les applications exécutées sur des processeurs multi-cœur et many-cœur sont décrites dans le chapitre 3. Les chapitres 4 et 5 présentent l'évaluation des impacts des radiations neutroniques sur plusieurs scénarios mis en œuvre sur deux processeurs cibles: le Freescale P2041 multi- cœurs et le many-cœur processeur KALRAY MPPA (Multi Purpose Processing Array) -256. Le chapitre 6 décrit une nouvelle approche logicielle de tolérance aux fautes développée dans le cadre de cette thèse, appelé NMR-MPar, et son évaluation à travers une étude de cas mise en œuvre sur le processeur many-cœur MPPA-256. Les travaux les plus proches sur cette thématique sont résumés dans le chapitre 7. Finalement le chapitre 8 dresse des conclusions générales et propose des perspectives à cette thèse.

ii. Chapitre 2 : Cadre théorique

Ce chapitre décrit les principaux concepts nécessaires à une bonne compréhension de cette thèse. Trois thèmes principaux sont abordés. Le premier est lié à l'environnement de radiation, ses effets et ses conséquences sur les systèmes électroniques. Le second est consacré à la présentation des principales caractéristiques et fonctionnalités des processeurs multi-cœur et many-cœur. Letroisième thème résume les concepts clés sur la fiabilité etquelques techniques pour l'améliorer.

Compte tenu de la multiplicité des cœurs et de l'utilisation de techniques de redondance pour améliorer la fiabilité, il est important de déterminer la possibilité d'utiliser des processeurs généralistes (COTS, Commercial-Off-The-Shelf) pour exécuter des applications dans des environnements avec des radiations sévères.

iii. Chapitre 3: Méthodologie et outils pour évaluer l'impact des SEE sur la fiabilité des applications

Ce chapitre décrit la méthodologie et les outils utilisés pour évaluer l'impact des SEEs (Single Event Effects) sur des applications parallèles développées sur des processeurs multi-cœur et many-cœur. Cette étude propose l'utilisation de la théorie quantitative avec des études de cas multiples appliquées à deux plates-formes expérimentales dif-

férentes. Compte tenu de la large palette de configurations d'environnements logiciels possibles, la sélection des études de cas a été effectuée en utilisant une stratégie bottom-up. Le premier cas, qui est le plus simple, considère un processeur multi-cœur exécutant une application bare-board où chaque cœur s'exécute indépendamment des autres. En revanche, le dernier cas propose un processeur multi-core exécutant une application parallèle massive redondante sous POSIX. Six études de cas ont été évaluées pendant cette thèse:

A) Chaque cœur supporte l'exécution d'une application séquentielle sans aucune coordination entre les différents cœurs.

B) Un multi-cœur en partageant ressources et utilisant des communications inter-cœur.

C) Un multi-cœur sur lequel est implémenté TMR (Triple Modular Redundancy) comme technique de tolérance aux fautes.

D) Un many-cœur qui supporte l'exécution d'une application avec une utilisation minimale des communications NoC (Network-on-Chip).

E) Un many-cœur qui supporte l'éxécution d'une application avec une utilisation intensive des communications NoC.

F) Un many-cœur sur lequel est implémenté la nouvelle approche proposée de tolérance aux fautes sur des applications parallèles.

L'évaluation prend en compte l'utilisation des modes Asymmetric Multi-Processing (AMP) et Symmetric Multi-Processing (SMP) et deux applications de référence: Traveling Salesman Problem (TSP) et Matrix Multiplication (MM). L'évaluation a été effectuée par la technique *Software-Implemented Fault Injection* (SWIFI) et/ou par des expériences de tests sous neutrons en accélérateur de particules. Pour analyser les résultats expérimentaux, les conséquences d'un bit-flip dans une cellule de mémoire ont été classées: *fautes silencieuses, résultats erronés, timeouts* et *exceptions*.

Concernant l'injection de fautes, dans cette thèse, les principes de l'approche *Code Emulating Upset* (CEU), développée à TIMA dans des thèses précédentes, ont été adaptés aux processeurs multi-cœur et many-cœur. Cette approche basée sur des signaux d'interruption fournit des résultats de taux d'erreur proches de ceux obtenus en accélérateur de particules. Ce résultat a été présenté dans [RVE+01, VFP10]. Son applicabilité à un processeur complexe, tel que le Power PC4748, a permis de valider ce dispositif pour des applications aéronautiques [PEP+08].

Il est important de noter que l'objectif de cette approche est de reproduire, sans intrusion, les effets des fautes SEU (Single Event Upset). Cela a été effectué en activant des signaux d'interruption asynchrones. L'exécution du gestionnaire d'interruption produit l'erreur dans une cible choisie aléatoirement. Dans les travaux précédents, l'interruption a été produite par un dispositif externe. Dans le cas de processeurs multi / many-cœur, il est possible de bénéficier de la multiplicité de cœurs pour utiliser l'un d'entre eux comme un injecteur de fautes, tandis que les autres exécutent l'application. Ce type d'injecteur de fautes est dépendant de l'architecture. L'approche a été adaptée dans chaque étude de cas en fonction de l'environnement logiciel et de la plate-forme de développement utilisée.

Concernant l'expérimentation, deux dispositifs COTS ont été considérés. Le premier a été le processeur Freescale PowerPC P2041-core, qui a été sélectionné en raison de sa technologie de fabrication (SOI-Silicon-On-Insulator) et sa capacité à permettre différents modes de *multi-processing*. Le second a été le processeur many-cœur KALRAY MPPA-256, qui a été sélectionné en raison de sa technologie de fabrication avancée CMOS 28nm, de son architecture et du grand nombre de cœurs de traitement (256) qu'il intègre.

Cette recherche considère l'évaluation de la fiabilité d'un système seulement pendant sa durée de vie. Pour calculer le taux d'erreur utilisé pour évaluer la fiabilité, il a été utilisé la limite supérieure de l'intervalle de confiance. En outre, il a été considéré que les applications sont destinées à être utilisées dans le domaine de l'avionique, soit le flux neutronique à 35000 pieds d'altitude avec une durée de vie du système de 50000 h [1].

[1]Les lignes aériennes commerciales volent plus de 10 km et la durée de vie économique est de 50000 heures de vol

iv. Chapitre 4: Études de cas basées sur le processeur multi-cœur

Ce chapitre traite des études de cas réalisées sur le quad-cœur P2041 construit en technologie 45nm SOI. Au début du chapitre, certains détails spécifiques de ce dispositif sont présentés. Ensuite, les trois études de cas permettront d'évaluer la sensibilité aux SEE des applications exécutée sur ce multi-cœur. Chaque cas résume la configuration du système et les résultats obtenus.

L'évaluation réalisée sur le multi-cœur Freescale P2041 a démontré, que malgré la technologie de fabrication SOI, qui est plus robuste que le CMOS traditionnel, et les mécanismes de protection mis en œuvre sur la mémoire, des résultats erronés se sont quand même produits. De plus, les résultats confirment la dépendance de la sensibilité à l'environnement logiciel. Il a été démontré que le mode AMP était plus fiable que le mode SMP. De plus, il a été possible de vérifier que les caractéristiques intrinsèques de l'application jouent un rôle important sur la sensibilité du système.

En outre, il a été observé une vulnérabilité considérable du système d'exploitation, plus particulièrement en ce qui concerne les exceptions du système. Cette vulnérabilité a mis en évidence que la mise en œuvre de la redondance au niveau applicatif ne suffit pas à surmonter les problèmes de fiabilité. Par conséquent, il est clair qu'il est impératif d'utiliser le concept de partitionnement pour garantir, à la fois la fiabilité mais aussi la disponibilité du système.

v. Chapitre 5: Études de cas basées sur le processeur many-cœur

Pour évaluer l'impact des SEE sur des applications fonctionnant sur le processeur MPPA, deux études de cas ont été proposées. La première maximise l'utilisation des ressources internes des clusters de calcul, tandis que la deuxième exploite le parallélisme massif. La plate-forme d'évaluation utilisée pour évaluer le processeur many-cœur est le MPPA Developer.

L'évaluation a été réalisée grâce à des campagnes d'injection des fautes et/ou à des

campagnes de tests sous neutrons. Les détails de chaque cas sont décrits dans les sections correspondantes. L'évaluation des scénarios mis en œuvre sur le processeur many-cœur MPPA-256 a prouvé l'efficacité des mécanismes de protection implémentés dans ses mémoires. De plus, il a été également possible de déterminer la pertinence d'activer les mémoires cache de ce dispositif afin d'améliorer ses performances avec des conséquences minimales sur la fiabilité. Ces résultats corroborent également que la sensibilité aux SEE du dispositif dépend de l'environnement logiciel. De plus, grâce à la mise en œuvre d'applications parallèles sur le processeur many-cœur, il a été possible de confirmer que la migration d'une application vers un système distribué communiquant par les services NoC est une tâche non triviale.

vi. Chapitre 6: NMR-MPar: une approche tolérante aux fautes

Ce chapitre décrit une approche de tolérance aux fautes développée et évaluée dans cette thèse appelée NMR-MPar. Ce nom provient de deux concepts de base utilisés: *N-Modular Redundancy and Partitioning*, pour améliorer la fiabilité des applications s'exécutant sur des processeurs multi-cœur et many-cœur. En combinant ces deux concepts, cette approche permet aux processeurs multi/many-cœur d'effectuer des fonctions critiques dans des systèmes à criticité mixte. Profitant des capacités des processeurs multi/many-cœur, NMR-MPar utilise le principe du partitionnement pour créer différentes partitions qui coexistent dans le même dispositif. Cela est généralement effectué par l'hyperviseur qui fournit des CPUs virtuels à chaque partition.

En revanche, NMR-MPar propose une distribution de ressources physiques à chaque partition, afin de minimiser la propagation de fautes produisant des dysfonctionnements d'autres ressources ou cœurs. Chaque partition peut être configurée en tant que mono ou multi-cœur en mode bare-board, AMP ou SMP. Par conséquent, il existe une grande flexibilité pour la configuration du système qui est améliorée par le nombre de cœurs du dispositif.

En outre, pour des fonctions critiques, il est proposé que N partitions avec la même configuration participent au système NMR. Ainsi, plusieurs partitions exécutent la même

application et leurs résultats sont utilisés par un voteur pour construire un système tolérant aux fautes. Le système voteur peut comprendre un ou plusieurs cœurs du dispositif ou un dispositif externe, si on le souhaite. Selon la partition du dispositif, plusieurs systèmes de redondance N-Modular peuvent s'exécuter simultanément.

Un prototype logiciel a été implémenté sur le processeur MPPA-256, exécutant deux applications de référence. Il met en œuvre l'approche NMR-MPar pour améliorer la fiabilité des applications parallèles exécutées sur le processeur cœurs MPPA-256 avec $N = 4$.

Nos contributions profitent des capacités de l'architecture du MPPA-256 qui permet de diviser la puce en deux parties matérielles indépendantes. Chaque partie est utilisée avec un schéma maître/esclave. Dans ce cas, une séparation symétrique est proposée, de sorte que chaque cluster IO gère une moitié des cœurs. Pour maximiser le parallélisme utilisé et en raison du fait que le partitionnement minimal possible est un *compute cluster*. Le prototype implémente deux modules par IO cluster. De cette façon, c'est comme avoir un système configuré comme un double *Dual Modular Redundancy* (DMR). Par conséquent, si un cluster IO a un dysfonctionnement, l'autre est capable de continuer à travailler. De plus, cette division garantit l'isolation spatiale et temporelle des systèmes critiques.

L'efficacité de cette approche a été analysée à travers l'évaluation par des campagnes d'injection de fautes. D'après les résultats obtenus, l'approche NMR-MPar s'est avérée très efficace pour améliorer la fiabilité des applications testées. Par exemple, on peut observer que la fiabilité au pire des cas du système (à t = 50000h) a été augmentée de 0,95 à 0,9998 pour le TSP et de 0,75 à 0,9964 pour le MM. En outre, la fiabilité du système pourrait être encore améliorée en mettant en œuvre des techniques SIFT (Software Implemented Fault Tolerance) au niveau de l'application. Par exemple, il pourrait être utile de combiner cette approche avec d'autres approches telles que [SAL+08] qui propose une réplication sélective d'instructions qui protège les calculs d'adresses et les branchements conditionnelles qui pourraient générer des erreurs de segmentation ou charger des valeurs de données incorrectes.

vii. Chapitre 7: Travaux connexes

Ce chapitre résume les principaux travaux concernant les thèmes abordés dans cette thèse. La littérature est regroupée autour de deux sujets:

- l'évaluation de la sensibilité face aux SEE.

- La fiabilité des processeurs multi- cœurs et many-cœur. Deux voies ont été explorées: tolérance aux fautes par redondance logicielle et tolérance aux fautes par partitionnement.

La référence [GPm+16] est probablement le travail le plus proche de NMR-MPar développé dans cette thèse. Les auteurs proposent une hybridation entre l'exécution duplex au niveau tâche sur COTS et le temps et l'espace de partitionnement fourni par l'hyperviseur bare-board XtratuM 2. Les auteurs ont mis en œuvre leur proposition sur un processeur ZYNQ qui héberge un ARM à deux cœurs Cortex A9. Les principales différences entre ce travail est le notre sont:

1. la redondance dans l'approche NMR-MPar est totalement indépendante au niveau du système et de l'utilisateur,

2. NMR-MPar a un hyperviseur indépendant pour chaque partition,

3. notre proposition a été mise en œuvre sur un processeur many-cœur,

4. notre proposition a été évaluée par injection de fautes.

La littérature comprend d'autres approches pour améliorer la fiabilité par des critères de redondance et de partitionnement. Cependant, peu de travaux les évaluent par simulation ou par injection de fautes. Cette recherche utilise l'injection de fautes basée sur l'approche CEU comme moyen d'évaluation pour la présente proposition pour les processeurs many-cœur.

viii. Conclusions et perspectives

La méthodologie proposée, basée sur des études de cas multiples, a permis d'étudier l'impact des SEE sur les systèmes COTS basés sur des processeurs multi/many-cœur avec différents scénarios. Les différentes études de cas ont tenu compte d'un large éventail de configurations possibles en termes de mode de multi-processing, modèle de programmation, technologie de fabrication, nombre de cœurs, modèle de mémoire et ressources utilisées. Les résultats obtenus peuvent servir de guide utile aux développeurs pour choisir la configuration de système la plus fiable en fonction de leurs besoins.

L'un des piliers de la migration des processeurs mono-cœurs vers les processeurs multi / many-cœur dans les systèmes critiques est la possibilité de profiter de la redondance inhérente pour implémenter la tolérance aux fautes par réplication. Comme prévu, la mise en œuvre de la TMR sur le quad-core a amélioré de manière significative la fiabilité du système. Cependant, le TMR réduit la capacité de calcul parallèle. A l'opposé, le NMR-MPar est une solution très prometteuse pour améliorer la fiabilité des applications parallèles fonctionnant sur les processeurs COTS many-cœur.

Contributions à l'état de l'art

Les principales contributions de cette thèse peuvent être résumées comme suit:

- L'approche NMR-MPar, développée au cours de cette thèse, basée sur les concepts du NMR et de partitionnement s'est révélée très efficace pour améliorer la fiabilité des applications fonctionnant sur des processeurs COTS many-cœur. Les résultats obtenus montrent que seules les exceptions ne peuvent être complètement surmontées par cette approche. Ce problème peut être surpassé en mettant en œuvre des techniques Software Implemented Fault Tolerance (SIFT) sur l'application et le niveau OS. Par exemple, [SAL+08] propose une réplication d'instructions sélectives qui protège les calculs d'adresses et les branchements conditionnelles afin de réduire les défauts de segmentation ou de charger des valeurs de données incorrectes. Par conséquent, en combinant NMR-MPar avec des techniques SIFT complémentaires, il sera possible d'utiliser des processeurs COTS multi-cœur et many-cœur dans des environnements de rayonnement sévères tels que l'espace et la haute atmosphère.

- Dans les références [RVB+16, RVB+15], l'évaluation de la sensibilité dynamique des différents scénarios du P2041 multi-cœur a été présentée. A partir des résultats, on peut constater que la sensibilité dynamique du dispositif dépend fortement du mode *multi-processing* utilisé. De plus, ces travaux montrent que le processeur quadri-cœurs SOI 45nm est environ quatre fois moins sensible aux SEE que sa contrepartie CMOS.

- Le travail [VRR+16] présente la sensibilité aux neutrons à 14 MeV du processeur many-cœur MPPA-256. Les résultats montrent que l'ECC et le *bit-interleaving* mis en œuvre dans les SMEM des clusters sont très efficaces pour atténuer les effets des SEUs car tous les événements détectés de ce type sont corrigés. En outre, l'évaluation de la réponse du dispositif montre qu'en activant les mémoires caches, il est possible d'augmenter les performances de l'application, sans compromettre la fiabilité.

- Les références [CFV+15, CFV+16] visent à évaluer la sensibilité aux radiations neutroniques sur les dispositifs dotés de technologies émergentes. Elles présentent la sensibilité aux SEE des mémoires avancées *low-power* SRAM Renesas fonctionnant à faible tension de polarisation. Les résultats montrent que, à basse tension, on peut produire une série de retournements de bits et d'erreurs sévères.

- Enfin, différentes variantes d'injection des fautes CEU ont été développées. Ces variantes comprennent un injecteur de fautes basé sur le principe *fork*, un injecteur de fautes en bare-board basé sur des interruptions entre processeurs, et un injecteur de fautes distribués basé sur les interruptions NoC et inter-processeur [VRM+14, VRV+15].

Directions futures

En ce qui concerne les orientations futures, les travaux de cette thèse ouvrent plusiurs pistes sur les aspects suivants:

- *Èvaluer la efficacité de l'approche NMR-MPar avec des tests sous radiation.* Cette étape est indispensable pour raffiner les résultats obtenus par l'injection des fautes.

Les essais de neutrons et d'ions lourds sont appropriés pour valider l'applicabilité aux applications avioniques et spatiales.

- *Conjuguer cette approche avec d'autres techniques SIFT.* La mise en œuvre de techniques de tolérance aux fautes au niveau du système d'exploitation et de l'utilisateur minimisera la principale soucis de l'approche NMR-MPar: *les exceptions.* Certaines de ces techniques comprennent des techniques de compilation et de réplication d'instructions, de la gestion optimisée des registres et des lignes de cache. Il est donc intéressant d'améliorer la fiabilité du système en combinant NMR-MPar avec d'autres approches de tolérance aux pannes.

- *Èvaluer NMR-MPar sur d'autres configurations de système.* Bénéficiant du grand nombre de cœurs dans le processeur many-cœur, il convient d'évaluer d'autres configurations de systèmes. Par exemple, la mise en œuvre de deux TMR indépendants exécutant des applications différentes. En outre, il peut être intéressant de mettre en œuvre cette approche sur d'autres plates-formes pour évaluer son applicabilité à d'autres systèmes basés sur des processeurs multi/many-cœur.

- *Èvaluer l'overhead en termes d'énergie et de consommation de puissance.* Ceci est essentiel pour renforcer son utilisation dans des systèmes embarqués et critiques.

Projects References

ACROSS Advanced Cockpit for Reduction Of StreSs and workload
Online: http://www.across-fp7.eu

ARTEMIS EMC^2 Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real time environments
Online: https://www.artemis-emc2.eu/

CAPACITES Calcul Parallèle pour Applications Critiques en Temps et Sûreté
"Investissement d'Avenir" program
Online : http://capacites.minalogic.net/en/

DECOS Dependable Embedded Components and Systems
Online : http://cordis.europa.eu/project/rcn/71582_en.html

MultiPARTES Multi-cores Partitioning for Trusted Embedded Systems
Online : http://www.multipartes.eu

CAST Certified Associate in Software Testing
Online : https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/

EASA European Aviation Safety Agency
Online : https://www.easa.europa.eu .

RECOMP Reduced Certification Costs for Trusted Multi-core Platforms
Online : https://artemis-ia.eu/project/21-recomp.html .

Bibliography

[ABCC07] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*, pages 49–53. Princeton University Press, Princeton, USA, September 2007.

[AML+15] M. S. Alhakeem, P. Munk, R. Lisicki, H. Parzyjegla, H. Parzyjegla, and G. Muehl. A framework for adaptive software-based reliability in cots many-core processors. In *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*, pages 1–4, March 2015.

[AMRG14] J. Autran, P. Munteanu, P. Roche, and G Gasiot. Real-time Soft- Error Rate Measurements: A Review. *Microelectronics Reliability*, 54(8):1455–1476, Feb. 2014.

[AVTK05] G.H. Asadi, S. Vilas, M.B. Tahoori, and D. Kaeli. Balancing performance and reliability in the memory hierarchy. In *Proc. Performance Analysis of Systems and Software*, pages 269–279, Mar. 2005.

[BDH+12] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for strassen's matrix multiplication. In *Proc. of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 193–204, New York, NY, USA, 2012. ACM.

[BI06] Z. Basile, C.and Kalbarczyk and R. K. Iyer. Active Replication of Multithreaded Applications. *IEEE Trans. Parallel And Distributed Systems*, 17(5):448–465, May 2006.

[BMS12] C. Bolchini, A. Miele, and D. Sciuto. An adaptive approach for online fault management in many-core architectures. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1429–1432, March 2012.

[Cer16] Certification Authorities Software Team (CAST) . Position paper cast-32 multi-core processors, Nov. 2016.

[CFV+15] J. A. Clemente, F. J. Franco, F. Villa, M. Baylac, P. Ramos, V. Vargas, H. Mecha, J. A. Agapito, and R. Velazco. Neutron-Induced Single Events in a COTS Soft-Error Free SRAM at Low Bias Voltage. In *Proceedings of IEEE European Conferences on Radiation Effects on Components and Systems (RADECS2015)*, pages 162–165, Sep. 2015.

[CFV+16] J. A. Clemente, F. J. Franco, F. Villa, M. Baylac, P. Ramos, V. Vargas, H. Mecha, J. A. Agapito, and R. Velazco. Single Events in a COTS Soft-Error Free SRAM at Low Bias Voltage Induced by 15-MeV Neutrons. *IEEE Trans. Nucl. Sci.*, 63(4):2072 – 2079, Aug. 2016.

[CSE+06] Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi, and S. M. Reddy. Cache size selection for performance, energy and reliability of time-constrained systems. In *Asia and South Pacific Conference on Design Automation, 2006.*, pages 6 pp.–, Jan 2006.

[dDAB+13] B. D. de Dinechin, R. Ayrignac, P. E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss, and T. Strudel. A clustered manycore processor architecture for embedded and accelerated applications. In *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sept 2013.

[dDdML+13] Benoît Dupont de Dinechin, Pierre Guironnet de Massas, Guillaume Lager, Clément Léger, Benjamin Orgogozo, Jérôme Reybert, and Thierry Strudel. A distributed run-time environment for the kalray mppaÂő-256 integrated manycore processor. *Procedia Computer Science*, 18:1654 – 1663, 2013. 2013 International Conference on Computational Science.

[DM03] P.E. Dodd and L.W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Trans. Nucl. Sci.*, 50(3):583–602, June 2003.

[Don16] Dongarra, J. Report on the Sunway TaihuLight System, June 2016.

[Dot] B. Dotson. How particle accelerators work. Online: https://energy.gov/articles/how-particle-accelerators-work.

[DWVVL+13] N. De Witte, R. Vincke, S. Van Landschoot, E. Steegmans, and J. Boydens. Comparing Dual-Core SMP/AMP Performance on a Telecom Architecture. *Journal of Electronics*, 7:72–75, Dec. 2013.

[Eri17] Erickson, K. . What is a laser. Online: http://spaceplace.nasa.gov/laser/en/, Jan 2017.

[FCP+15] E. Francesquini, M. Castro, P. Penna, F. Dupros, H. Freitas, P. Navaux, and J.F. Méhaut. On the energy efficiency and performance of irregular application executions on multicore, NUMA and manycore platforms. *Journal of Parallel and Distributed Computing*, 76:32–48, Feb. 2015.

[Fre11] Freescale. P2041 Reference Design Board, 2011.

[Fre12] Freescale. Running AMP, SMP or BMP Mode for Multicore Embedded Systems, 2012.

[Fre13a] Freescale. e500mc Core Reference Manual, 2013.

[Fre13b] Freescale. P2040/P2041 QorIQ Integrated Multicore Communication Processor Family Reference Manual, 2013.

[FVM+07] P.A. Ferreyra, G. Viganotti, C.A. Marques, R. Velazco, and R.T. Ferreyra. Failure and Coverage Factors Based Markoff Models: A New Approach for Improving the Dependability Estimation in Complex Fault Tolerant Systems Exposed to SEUs. *IEEE Trans. Nucl. Sci.*, 54(4):912–919, Aug. 2007.

[FZLF08] X. Fu, W. Zhang, T. Li, and J. Fortes. Optimizing issue queue reliability to soft errors on simultaneous multithreaded architectures. In *2008 37th International Conference on Parallel Processing*, pages 190–197, Sept 2008.

[Gai11] R. Gaillard. *Single Event Effects: Mechanisms and Classification*, pages 27–54. 2011.

[GFCB+02] G. Gasiot, V. Ferlet-Cavrois, J. Baggio, P. Roche, P. Flatresse, A. Guyot, P. Morel, O. Bersillon, and J. du Port de Pontcharra. SEU Sensitivity of Bulk and SOI Technologies to 14-MeV Neutrons. *IEEE Trans. Nucl. Sci.*, 49(6):3032–3037, Dec. 2002.

[GPA+11] D Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. Sastry, D. Sorin, A. Meixner, A. Biswas, and X. Vera. Architectures for online error detection and recovery in multicore processors. In *Proc. Design, Automation & Test in Europe Conference*, March 2011.

[GPm+16] J. Galizzi, M. Pignol, M. masmano, M. Munoz, J. Coronel, T. Parrain, and P. Combettes. Temporal duplex-triplex on cots processors with xtratum. In *2016 Eurospace DASIA Conference*, pages 1–5, May 2016.

[Gue12] S. Guertin. Initial SEE Test of Maestro. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, Jul. 2012.

[HBR11] R. Hyman, K. Bhattacharya, and N. Ranganathan. Redundancy Mining for Soft Error Detection in Multicore Processors. *IEEE Trans. Comp.*, 60(8):1114–1125, Aug. 2011.

[HRI+16] A. Holler, T. Rauter, J. Iber, G. F. H. Macher, and C. J. Kreiner. *Software-Based Fault Recovery via Adaptive Diversity for Reliable COTS Multi-Core Processors*, pages 1–6. 1 2016.

[HTI97] Mei-Chen Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, Apr 1997.

[Huy12] P. Huyck. Arinc 653 and multi-core microprocessors x2014; considerations and potential impacts. In *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pages 6B4–1–6B4–7, Oct 2012.

[IEE01] IEEE Computer Society . 1003.1-2001 - IEEE Standard for IEEE Information Technology - Portable Operating System Interface (POSIX(R)). Online : http://standards.ieee.org/findstds/standard/1003.1-2001.html, 2001.

[Ini11] K. Iniewski. *Radiation Effects in Semiconductors*. CRC Press Taylor 'I&' Francis Group, Boca Raton, FL, USA, 2011.

[JAC+02] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun, and T. Marteau. Analysis of the Effects of Real and Injected Software Faults: Linux as a Case Study. In *Proc. of Pacific Rim International Symposium on Dependable Computing PRDC2002*, pages 51–58, December 2002.

[JFG+12] X. Jean, D. Faura, M. Gatti, L. Pautet, and T. Robert. Ensuring robust partitioning in multicore platforms for ima systems. In *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pages 7A4–1–7A4–9, Oct 2012.

[JP95] D.S. Johnson and C.H. Paadimitriou. *Computational Complexity*, pages 37–85. Wiley Series in Discrete Mathematics and Optimization . Wiley and Sons, Chichester, USA, September 1995.

[JSKI08] G. Jacques-Silva, Z Kalbarczyk, and R. K. Iyer. Dependability Assessment of Operating Systems in Multi-core Architectures. In *Proc. Int. Symp. on Dependable Systems and Networks*, June 2008.

[KA11] F. Kraja and G. Acher. Using many-core processors to improve the performance of space computing platforms. In *2011 Aerospace Conference*, pages 1–17, March 2011.

[Kal13] Kalray. MPPA Process Management and Communication API, 2013.

[Kal15] Kalray. MPPA ACCESSCORE V1.4 Introductory Manual, 2015.

[Kal16] Kalray. MPPA-256 Bostan Cluster and I/O Subsystem Architecture, 2016.

[Kim09] S. Kim. Reducing area overhead for error-protecting large l2/l3 caches. *IEEE Transactions on Computers*, 58(3):300–310, March 2009.

[KN14] M. Kooli and G. Di Natale. A survey on simulation-based fault injection tools for complex systems. In *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, May 2014.

[KS79] I. Koren and S. Y. H. Su. Reliability analysis of n-modular redundancy systems with intermittent and permanen faults. *IEEE Transactions on Computers*, c-28(7):514–520, July 1979.

[KS99] S. Kim and A. K. Somani. Area efficient architectures for information integrity in cache memories. In *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, pages 246–255, 1999.

[KS12] E. P. Kim and N. R. Shanbhag. Soft n-modular redundancy. *IEEE Transactions on Computers*, 61(3):323–336, March 2012.

[KSD+97] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz. Comparing operating systems using robustness benchmarks. In *Proc. Reliable Distributed Systems*, October 1997.

[LNT14] A. Löfwenmark and S. Nadjm-Tehrani. Challenges in future avionic systems on multi-core platforms. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 115–119, Nov 2014.

[LPC+12] A. Lanzaro, A. Pecchia, M. Cinque, D. Cotroneo, R. Barbosa, and N. Silva. *A Preliminary Fault Injection Framework for Evaluating Multicore Systems*, pages 106–116. Springer Berlin Heidelberg, Berlin, Heidelberg, September 2012.

[LSI+06] K. Lee, A. Shrivastava, I. Issenin, N. Dutt, and N. Venkatasubramanian. Mitigating soft error failures for multimedia applications by selective

data protection. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 411–420, New York, NY, USA, 2006. ACM.

[LV62] R.E. Lyons and W. Vanderkulk. The use of triple modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, April 1962.

[MAAB13] H. Mushtaq, Z. Al-Ars, and K. Bertels. Efficient Software-Based Fault Tolerance Approach on Multicore Platforms. In *Proc. Design, Automation & Test in Europe Conference*, pages 921–926, March 2013.

[MEA+10] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1864–1871, June 2010.

[MFMR08] R. Maurer, M. Fraeman, M. Martin, and D. Roth. Harsh Environments: Space Radiation Environment, Effects and Mitigation. *John Hopkins APL Technical Digest*, 28(1):17–29, 2008.

[MKO05] G. Memik, M. T. Kandemir, and O. Ozturk. Increasing register file immunity to transient errors. In *Design, Automation and Test in Europe*, pages 586–591 Vol. 1, March 2005.

[MKR02] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed design and evaluation of redundant multi-threading alternatives. In *Proceedings 29th Annual International Symposium on Computer Architecture*, pages 99–110, 2002.

[MMM12] M. Maniatakos, M. K. Michael, and Y. Makris. Vulnerability-based interleaving for multi-bit upset (mbu) protection in modern microprocessors. In *2012 IEEE International Test Conference*, pages 1–8, Nov 2012.

[MRAV14] W. Mansour, P. Ramos, R. Ayoubi, and R. Velazco. SEU fault-injection at system level: method, tools and preliminary results. In *Proc. Latin American Test Workshop LATW*, March 2014.

[MSM+02] H. Madeira, R.R. Some, F. Moreira, D. Costa, and D Rennels. Experimental evaluation of a COTS system for space applications. In *Proc. International Conference on Dependable Systems and Networks*, pages 325–330, June 2002.

[MV16] S. Mittal and J.S. Vetter. A survey of techniques for modeling and improving reliability of computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1226–1238, April 2016.

[MW79] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. Elec. Dev.*, 26(1):2 – 9, Jan. 1979.

[MWC+13] F. Miller, C. Weulersse, T. Carriere, N. Guibbaud, S. Morand, and R. Gaillard. Investigation of 14 MeV Neutron Capabilities for SEU Hardness Evaluation. *IEEE Trans. Nucl. Sci.*, 60(4):2789–2796, Aug. 2013.

[MWE+03] 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.

[NAR+13] H Naeimi, C. Augustine, A. Raychowdhury, S.L. Lu, and J. Tschanz. Sttram scaling and retention failure. *Intel Technology Journal*, 17(1):54–75, May 2013.

[ND10] E. Normand and L. Dominik. Cross Comparison Guide for Results of Neutron SEE Testing of Microelectronics Applicable to Avionics. In *Proc. Radiation Effects Data Workshop*, pages 50–57, July 2010.

[Nic11] M. Nicolaidis. *Soft Errors in Modern Electronic Systems*. Frontiers in Electronic Testing. Springer US, 2011.

[Nor96] E. Normand. Single-event effects in avionics. *IEEE Transactions on Nuclear Science*, 43(2):461–474, Apr 1996.

[OLA96] I. Osborne-Lee and C. Alexander. CALIFORNIUM-252: A Remarkable Versatil Radioisotope. *ORNL/TM-12706*, pages 1–37, 1996.

[Ope15] OpenMP Architecture Review Board . OpenMP Application Programming Interface. Online : http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf, 2015.

[ORQ+14] D.A.G. Oliveira, P. Rech, H.M. Quinn, T.D. Fairbanks, L. Monroe, S.E. Michalak, C. Anderson-Cook, P.O.A. Navaux, and L. Carro. Modern GPUs Radiation Sensitivity Evaluation and Mitigation Through Duplication With Comparison. *IEEE Trans. Nucl. Sci.*, 61(6):3115–3122, Dec. 2014.

[OTKT12] I. Oz, H. R. Topcuoglu, M. Kandemir, and O. Tosun. Performance-reliability tradeoff analysis for multithreaded applications. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 893–898, March 2012.

[PEP+08] P. Peronnard, R. Ecoffet, M. Pignol, D. Bellin, and R. Velazco. Predicting the SEU Error Rate through Fault Injection for a Complex Microprocessor. In *Proc. 2008 IEEE International Symposium on Industrial Electronics*, pages 2288–2292, Sep. 2008.

[PnZ+14] M. Panić, E. Qui nones, P. G. Zavkov, C. Hernandez, J. Abella, and F. J. Cazorla. Parallel many-core avionics systems. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10, Oct 2014.

[PVH09] P. Peronnard, R. Velazco, and G. Hubert. Real-Life SEU Experiments on 90 nm SRAMs in Atmospheric Environment: Measures Versus Predictions Done by Means of MUSCA SEP3 Platform. *IEEE Transactions on Nuclear Science*, 56(6):3450–3455, 2009.

[RM00] S. K. Reinhardt and S. S. Mukherjee. Transient fault detection via simultaneous multithreading. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pages 25–36, June 2000.

[RRV03] M. Rebaudengo, M. Reorda, and M. Violante. An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor. In *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pages 602–607, 2003.

[RVB+15] P. Ramos, V. Vargas, M. Baylac, F. Villa, S. Rey, J.A Clemente, N.E. Zergainoh, and R. Velazco. Sensitivity to Neutron Radiation of a 45nm SOI Multi-core Processor. In *Proc. Radiation Effects on Components and Systems*, pages 135–138, Sep. 2015.

[RVB+16] P. Ramos, V. Vargas, M. Baylac, F. Villa, S. Rey, J.A Clemente, N.E. Zergainoh, J.F. Méhaut, and R. Velazco. Evaluating the SEE sensitivity of a 45nm SOI Multi-core Processor due to 14 MeV Neutrons. *IEEE Trans. Nucl. Sci.*, 63(4):2193 – 2200, Aug. 2016.

[RVE+01] S. Rezgui, R. Velazco, R. Ecoffet, S. Rodriguez, and J. Mingo. Estimating Error Rates in Processor-Based Architectures. *IEEE Trans. Nucl. Sci.*, 48(5):1680–1687, Dec. 2001.

[SAL+08] A. Sundaram, A. Aakel, D. Lockhart, D. Thaker, and D. Franklin. Efficient fault tolerance in multi-media applications through selective instruction replication. In *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies*, WREFT '08, pages 339–346, New York, NY, USA, 2008. ACM.

[SBM+09] A. Shye, J. Blomstedt, T. Moseley, V. Janapa Reddi, and D. A. Connors. PLR: A Software Approach to Transient Fault Tolerance for Multicore Architectures. *IEEE Trans. On Dependable And Secure Computing*, 6(2):135–148, April 2009.

[Sch10] E. Schoitsch. *Computer Safety, Reliability and Security*. Springer, Vienna, Austria, 2010.

[SEU+15] S. Saidi, R Ernst, S. Uhrig, H. Theiling, and B. Dupont de Dinechin. The shift to multicores in real-time and safety-critical systems. In

Proc. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pages 220–229, Oct. 2015.

[Sho90] M.L. Shooman. *Probabilistic Reliability: An Engineering Approach.* Krieger, Melbourne, FL, USA, 1990.

[Sho02] M.L. Shooman. *Reliability of Computer Systems and Networks: Fault Tolerance,Analysis,and Design.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[SIM07] M. Sugihara, T. Ishihara, and K. Murakami. Task scheduling for reliable cache architectures of multiprocessor systems. In *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007.

[SN12] S.S. Stolt and E. Normand. A Multicore Server SEE Cross Section Model. *IEEE Trans. Nucl. Sci.*, 59(6):2803–2810, Dec. 2012.

[SRN+14] T. Santini, P. Rech, G. Nazar, L. Carro, and F. Rech Wagner. Reducing Embedded Software Radiation-Induced Failures Through Cache Memories. Proc. European Test Symposium, DOI:10.1109/ETS.2014.6847793, May. 2014.

[TCAP14] S. Trujillo, A. Crespo, A. Alonso, and J. Pérez. Multipartes: Multicore partitioning and virtualization for easing the certification of mixed-criticality systems. *Microprocessors and Microsystems*, 38(8, Part B):921 – 932, 2014.

[TS12] H. Tabkhi and G. Schirner. Application-specific power-efficient approach for reducing register file vulnerability. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 574–577, March 2012.

[Vaj11] András Vajda. Multi-core and many-core processor architectures. In *Programming Many-Core Chips*, pages 9–43. Springer, 2011.

[VBR+14] F. Villa, M. Baylac, S. Rey, O. Rossetto, W. Mansour, P. Ramos, R. Velazco, and G. Hubert. Accelerator-Based Neutron Irradiation of Integrated

Circuits at GENEPI2 (France). Proc. Radiation Effects Data Workshop, DOI:10.1109/REDW.2014.7004511, July. 2014.

[VFP10] R. Velazco, G. Foucard, and P. Peronnard. Combining Results of Accelerated Radiation Tests and Fault Injections to Predict the Error Rate of an Application Implemented in SRAM-Based FPGAs. *IEEE Trans. Nucl. Sci.*, 57(6):3500–3505, Dec. 2010.

[Vig] W.J. Vigrass. Calculation of Semiconductor Failure Rates. Online: http://www.intersil.com/content/dam/Intersil/quality/rel/ calculation_of_semiconductor_failure_rates.pdf.

[VPC02] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient Fault Recovery using Simultaneous Multithreading. In *Proc. 29th Annual Int'l Symp. on Computer Architecture*, pages 87–98, May 2002.

[VRM+14] V. Vargas, P. Ramos, W. Mansour, R. Velazco, N. E. Zergainoh, and J. F. Méhaut. Preliminary results of SEU fault-injection on multicore processors in AMP mode. In *Proc. IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 194–197, Sep. 2014.

[VRR+16] V. Vargas, P. Ramos, V. Ray, C. Jalier, R. Stevens, B. Dupont de Dinechin, M. Baylac, F. Villa, S. Rey, N. E. Zergainoh, J. F. Méhaut, and R. Velazco. Radiation Experiments on a 28nm Single-Chip Many-core Processor and SEU error-rate prediction,. *IEEE Trans. Nucl. Sci.*, 99(4):1 – 8, Dec. 2016.

[VRSCH11] C. Villalpando, D. Rennels, R. Some, and M. Cabanas-Holmen. Reliable Multicore Processors for NASA Space Missions. In *Proc. Aerospace Conference*, pages 1–12, Mar. 2011.

[VRV+15] V. Vargas, P. Ramos, R. Velazco, J. F. Méhaut, and N. E. Zergainoh. Evaluating SEU fault-injection on parallel applications implemented on multicore processors. In *Proc. 6th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 181–184, Feb. 2015.

[WKS+11] J. P. Walters, R. Kost, K. Singh, J. Suh, and S. P. Crago. Software-based fault tolerance for the Maestro many-core processor. In *Proc. 2011 Aerospace Conference*, March 2011.

[ZAV04] H. Ziade, R. Ayoubi, and R. Velazco. A survey on fault injection techniques. *The International Arab Journal of Information Technology*, 1(2):171–186, 2004.

[ZGKS03] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. Icr: in-cache replication for enhancing data cache reliability. In *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, pages 291–300, June 2003.

[Zha05] W Zhang. Replication cache: a small fully associative cache to improve data cache reliability. *IEEE Transactions on Computers*, 54(12):1547–1555, Dec 2005.

Approche logicielle pour améliorer la fiabilité d'applications parallèles implémentées sur des processeurs multi-coeur et many-coeur

Résumé - La grande capacité de calcul, flexibilité, faible consommation d'énergie, redondance intrinsèque et la haute performance fournie par les processeurs multi / many-coeur les rendent idéaux pour surmonter les nouveaux défis dans les systèmes informatiques. Cependant, le degré d'intégration de ces dispositifs augmente leur sensibilité aux effets des radiations naturelles. Par conséquent, des fabricants, partenaires industriels et universitaires travaillent ensemble pour améliorer les caractéristiques de ces dispositifs ce qui permettrait leur utilisation dans des domaines embarqués et critiques. Dans ce contexte, le travail effectué dans le cadre de cette thèse vise à évaluer l'impact des SEEs (Single Event Effects) dans des applications parallèles s'exécutant sur des processeurs multi-coeur et many-coeur, et développer et valider une approche logicielle pour améliorer la fiabilité du système appelée NMR-MPar. La méthodologie utilisée pour l'évaluation était fondée sur des études de cas multiples et leur analyses. Les différents scénarios mis en oeuvre envisagent une large gamme de configurations de système en termes de mode de multi-processing, modéle de programmation, modèle de mémoire et des ressources utilisées. Pour l'expérimentation, deux dispositifs COTS ont été sélectionnés: le quadcore Freescale PowerPC P2041 en technologie SOI 45nm, et le processeur multi-coeur KALRAY MPPA-256 en CMOS 28nm. Les études de cas ont été évaluées par l'injection de fautes et par des campagnes des tests sur neutron. Les résultats obtenus servent de guide aux développeurs pour choisir la configuration du système la plus fiable en fonction de leurs besoins. En outre, les résultats de l'évaluation de l'approche NMR-MPar basée sur des critères de redondance et de partitionnement augmente l'utilisation des processeurs COTS multi/many-coeur dans des systèmes qui requièrent de la haute fiabilité.

Mots Clés: Architectures parallèles, Programmation Multi-core et many-core, Fiabilité, Redondance, Partitionnement, Injection de fautes.

Software approach to improve the reliability of parallel applications implemented on multi-core and many-core processors

Abstract -The large computing capacity, great flexibility, low-power consumption, intrinsic redundancy and high performance provided by multi/many-core processors make them ideal to overcome with the new challenges in computing systems. However, the degree of scale integration of these devices increases their sensitivity to the effects of natural radiation. Consequently manufacturers, industrial and university partners are working together to improve their characteristics which allow their usage in critical embedded domains. In this context, the work done throughout this thesis aims at evaluating the impact of SEEs on parallel applications running on multi-core and many-core processors, and proposing a software approach to improve the system reliability called NMR-MPar. The methodology used for evaluation was based on multiple-case studies and their analysis. The different scenarios implemented consider a wide range of system configurations in terms of multi-processing mode, programming model, memory model, and resources used. For the experimentation, two COTS devices were selected: the Freescale PowerPC P2041 quad-core built in 45nm SOI technology, and the KALRAY MPPA-256 many-core processor built in 28nm CMOS technology. The case-studies were evaluated through fault-injection and neutron radiation testing. The obtained results serve as useful guidelines to developers for choosing the most reliable system configuration according to their requirements. Furthermore, the evaluation results of the proposed NMR-MPar fault-tolerant approach based on redundancy and partitioning criteria boost the usage of COTS multi/many-core processors in high-level dependability systems.

Keywords: Reliability, Parallel Architectures, Multi-core and many-core programming, Redundancy, Partitioning, Fault injection

Thèse préparée au laboratoire TIMA (Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés), 46 Avenue Félix Viallet, 38031, Grenoble Cedex, France

ISBN: 978-2-11-129227-7