UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GABRIEL PISCOYA DÁVILA

A Performance, Energy Consumption and Reliability Evaluation of Workload Distribution on Heterogeneous Devices

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Paolo Rech
Coadvisor: Prof. Dr. Philippe Navaux

Porto Alegre
June 2019

“If I have seen farther than others,

it is because I stood on the shoulders of giants.”

— Sir Isaac Newton

ACKNOWLEDGEMENTS

I would like to thank Prof. Paolo Rech, my thesis advisor, and Prof. Philippe Olivier Alexandre Navaux, my thesis co-advisor. Thanks for all the support on the roller coaster path of this research. Thanks to the Radiation Group and the Group of Parallel and Distributed Processing (GPPD) for giving me a fertile soil to develop, discuss and refine my ideas. I would also like to thank my parents Julio and Alejandra, who taught me that there is no insurmountable challenge. Last but not least, I would like to thank all my friends spread around the world.

ABSTRACT

The constant need of higher performances and reduced power consumption has lead vendors to design heterogeneous devices that embed traditional Central Process Unit (CPU) and an accelerator, like a Graphics Processing Unit (GPU) or Field-programmable Gate Array (FPGA). When the CPU and the accelerator are used collaboratively the device computational performances reach their peak. However, the higher amount of resources employed for computation has, potentially, the side effect of increasing soft error rate.
This thesis evaluates the reliability behaviour of AMD Kaveri Accelerated Processing Units (APU) executing four heterogeneous applications, each one representing an algorithm class. The workload is gradually distributed from the CPU to the GPU and both the energy consumption and execution time are measured. Then, an accelerated neutron beam was used to measure the realistic error rates of the different workload distributions. Finally, we evaluate which configuration provides the lowest error rate or allows the computation of the highest amount of data before experiencing a failure.
As is shown in this thesis, energy consumption and execution time are mold by the same trend while error rates highly depend on algorithm class and workload distribution. Additionally, we show that, in most cases, the most reliable workload distribution is the one that delivers the highest performances. As experimentally proven, by choosing the correct workload distribution the device reliability can increase of up to 90x.

Keywords: Performance. Energy consumption. Reliability. Neutrons. Heterogeneous devices. Workload distribution. APU. CPU. GPU.

RESUMO

A constante necessidade de maior desempenho e menor consumo de energia levou aos fabricantes a projetar dispositivos heterogêneos que incorporam uma Unidade Central de Processameno (CPU) tradicional e um acelerador, como uma Unidade de Processamento Gráfico (GPU) ou um Arranjo de Portas Programáveis em Campo (FPGA). Quando a CPU e o acelerador são usados de forma colaborativa, o desempenho computacional do dispositivo atinge seu pico. No entanto, a maior quantidade de recursos empregados para o cálculo tem, potencialmente, o efeito colateral de aumentar a taxa de erros.

Esta tese avalia a confiabilidade das AMD Kaveri "Accelerated Processing Units"(APUs) executando quatro aplicações heterogêneas, cada uma representando uma classe de algoritmos. A carga de trabalho é gradualmente distribuída da CPU para a GPU e o consumo de energia e o tempo de execução são medidos. Em seguida, um feixe de neutrões é utilizado para medir as taxas de erro reais das diferentes distribuições de carga de trabalho. Por fim, avalia-se qual configuração fornece a menor taxa de erro ou permite o cálculo da maior quantidade de dados antes de ocorrer uma falha.

Como é mostrado nesta tese, o consumo de energia e o tempo de execução são moldados pela mesma tendência, enquanto as taxas de erro dependem da classe de algoritmos e da distribuição da carga de trabalho. Além disso, é mostrado que, na maioria dos casos, a distribuição de carga de trabalho mais confiável é a que fornece o maior desempenho. Como comprovado experimentalmente, ao escolher a distribuição de carga de trabalho correta, a confiabilidade do dispositivo pode aumentar até 9 vezes.

Palavras-chave: Desempenho, Consumo Energético, Confiabilidade,Nêutrons, Dispositivos Heterogêneos, Distribuição de Carga, APU, CPU, GPU.

LIST OF ABBREVIATIONS AND ACRONYMS

APU	Accelerated Processing Unit
CPU	Control Process Unit
GPU	Graphics Processing Unit
FPGA	Field Programmable Gate Array
FIT	Failure in Time
MEBF	Mean Execution Between Failure
MTBF	Mean Time Between Failure
EDP	Energy-Delay Product
WD	Workload Distribution
BS	Bezier Surface
SC	Stream Compaction
CED	Canny Edge Detection
BFS	Breadth First Search
CUDA	Compute Unified Device Architecture
DVFS	Dynamic Voltage and Frequency Scaling
HPC	High-Performance Computing
I|O	Input and Output

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Devices integrating cores of different nature in the same chip achieve very high computation efficiency by reducing the power consumption and latencies of moving data from a chip to an external device. However, the higher amount of resources employed for computation has, potentially, the side effect of increasing the soft error rate.

We evaluate the trade-off between execution time and error rate of collaborative workload distribution in heterogeneous devices. While it has already been investigated how workload distribution affects the execution time of heterogeneous devices(GÓMEZ-LUNA et al., 2017), our evaluation also includes reliability and its impact on the identification of the best configuration. We consider AMD Kaveri Accelerated Processing Units (APUs), that integrate CPU and GPU in a single chip, executing four algorithms from different computation classes. We progressively distribute the workload from the CPU to the GPU with the two compute units working collaboratively and measure the overall execution time and power consumption. Fault Injection campaigns were performed to obtain insights of the device fault model, and, through accelerated neutron beam experiments, the realistic device error rate was obtained.

This chapter presents the principal motivators, as well as the scientific contributions of this thesis.

1.1 Motivation

To reduce the execution time of most modern codes, a traditional CPU should be coupled with an accelerator, such as a Graphic Processing Units (GPU) or Field Programmable Gate Array (FPGA). However, the exchange of data between the CPU and the accelerator is a bottleneck that, potentially, could undermine execution efficiency. To reduce the time spent and energy consumed moving data from the computing core to the accelerator, engineers have introduced in the market novel heterogeneous architectures that integrate computing cores of different nature in a single device.

Nowadays heterogeneous devices are commonly adopted in smartphones, tablets, laptops, and other portable applications. Moreover, heterogeneous architectures are viewed as a promising solution, in the High-Performance Computing (HPC) domain, to reach exaFLOPS scale (10^{16} floating point operations per second) while maintaining the computing efficiency higher than 50 gigaFLOPS per Watt (LUCAS, 2014). Furthermore, heterogeneous architectures may ease the adoption of self-driving vehicles for automotive and aerospace applications. To implement self-driving features, in fact, the system must be able to detect objects in real-time in a limited power budget. Heterogeneous systems became then extremely attractive for automotive applications and Unmanned Aerial Vehicles (UAVs). Space agencies, such as NASA and ESA, are working on an autonomous vehicle for deep space exploration.

There are two main ways to take advantage of hardware heterogeneity: to alternatively use the available cores, using the core that is more suitable to execute a given task of the algorithm or use all the available cores concurrently, assigning a portion of the workload to each core. In both cases, the programmer needs to engineer the best workload distribution between the available cores. While the hardware architecture is promising in improving the computing efficiency, it is still unclear how to find the most efficient workload distribution among the available cores.

More powerful cores, like the accelerator, are likely to consume more power but, on the other side, could reduce the execution time significantly. Smaller cores are likely to reduce the power consumption but will take longer to complete the computation. Moreover, while reliability is not a major concern for portable user applications, it definitely is for the automotive, military, aerospace, and HPC markets. Devices for automotive applications, for instance, must be compliant with the ISO2626-2 standard, which requires the hardware error rate to be lower than 10 Failures In Time (FIT, i.e., errors per 10^9 hours of operation). Reliability has also been listed as one of the 10 top challenges to reach exaFLOPS scale (LUCAS, 2014) as transient errors can lead to lower scientific productivity and significant monetary loss (SNIR et al., 2014; RESEARCH, 2016). The error rate becomes then an additional variable to include in the evaluation of the best workload distribution in heterogeneous devices. The error rate of a code depends on the computing core in which it is being executed (FRATIN et al., 2018a). As a general observation, bigger cores like GPUs have a higher error rate but, thanks to the reduced execution time, can produce more correct data before a failure than CPUs.

While the reliability of standalone CPUs and FPGAs has been extensively studied (WANG; LIU; SUN, 2018; VALLERO; CARELLI; CARLO, 2018) and some preliminary research have addressed the reliability of GPUs for HPC and automotive applications (PREVILON et al., 2018; VALLERO et al., 2018), there are few works that target heterogeneous devices reliability.

1.2 Contributions

The contributions of this thesis include:

- A detailed evaluation of the performance, energy consumption, and error rate of collaborative workload distributions in heterogeneous devices.
- An unprecedented evaluation, through accelerated neutron beam experiments, of the realistic error rate of heterogeneous devices
- An evaluation of software fault-injections at source-code level on heterogeneous devices
- Insights on finding the workload distribution that provides the highest reliability regarding error rate and execution time.

1.3 Structure

This thesis is divided in 3 parts: performance, energy consumption and reliability evaluation. Chapter 2 presents background information on heterogeneous devices, starting by explaining the basic, and most common, compute units and then, piece them together to start constructing the heterogeneous concept. Collaborative programming model and workload distribution concepts are also presented. Chapter 3 details the evaluation methodology used on performance and energy consumption experiments, and emphasizes on experimental beam setup and Fault Injection campaign, finally it describes the selected benchmarks and the way the workload is distributed. Chapter 4 presents the results of the fault injection campaign, neutron beam experiments as well as energy consumption and execution time results. Mean Execution Time Between Failure (MEBF) is used to determine the best workload distribution regarding execution time and error rate of all tested benchmarks. At the long last, chapter 5 concludes the thesis and paves a path for future works.

2 BACKGROUND AND RELATED WORK

This chapter reviews background information on computing devices, collaborative programming model, performance, energy and reliability topics on heterogeneous devices as well as related works in the area. First, a brief introduction of compute units is done, followed by examples of heterogeneous devices. Then, collaborative programming model and workload distribution concepts are introduced. Performance and energy consumption challenges on heterogeneous devices are presented, and finally radiation induced errors and fault injection concepts are reviewed. Mean execution between failure metric is explained at the end of the chapter.

2.1 Computer Architectures

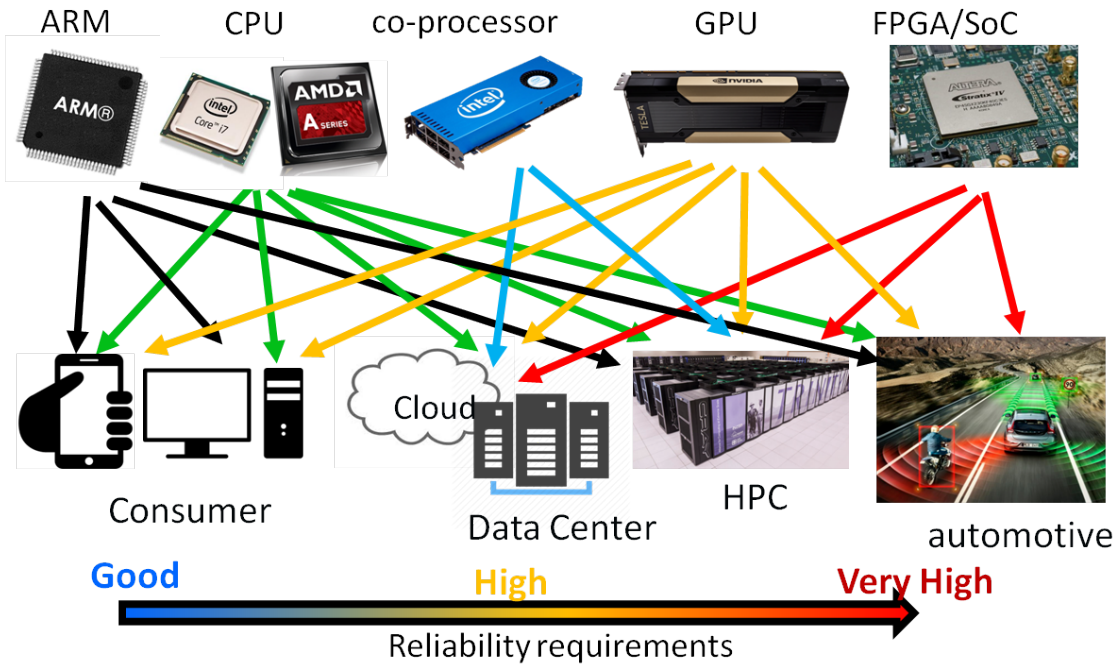
A compute unit is responsible for performing data and I|O operations, as well as data manipulation. Compute units vary in architecture, size, energy consumption, performance and fault model, accordingly to the target market for which they were developed. For High-performance computing market, like Super Computers, performance will be prioritized, and for Safety-Critical and embedded systems, like Autonomous Cars and IOT industry, both low-power budget markets, energy consumptions must be prioritized without neglecting the performance and the reliability. The figure 2.1 shows that modern computer architectures can be adapted and combined to support problem solutions in broad scope, therefore, depending on the target market, the required architecture unit reliability magnifies.

In the following subsections, a characterization of the most common compute units found on heterogeneous devices is presented.

2.1.1 Central Processing Unit

CPUs are general processing units, which perform basic arithmetic, logic and I|O operations. CPUs became popular in the late 1960's with the introduction of Transistor based CPUs, replacing the unreliable and fragile vacuum tubes. Nonetheless, it was not until early 1978, with the Intel 8086 market introduction, that became extremely popular. The figure 2.2 shows the principal architectural components of a CPU, starting with the

Figure 2.1: Reliability requirements variation among computer architecture markets.



Source: Arijit Biswas - SELSE 2018.

Arithmetic and Logical Unit (ALU), where integer arithmetic and bitwise operation are performed. The Control Unit, use electrical signals to orchestrate the entire computer system including control flow and deviation points, finally, the Memory Unit is in charge of granting ALU data feed and the storage of results. CPUs are the most appropriate computational units to perform control tasks and operations with high data dependency . The operating frequency can reach up to 4 GHz and have a high complexity architecture. Techniques to improve CPUs reliability are proposed on(TIMOR et al., 2010).

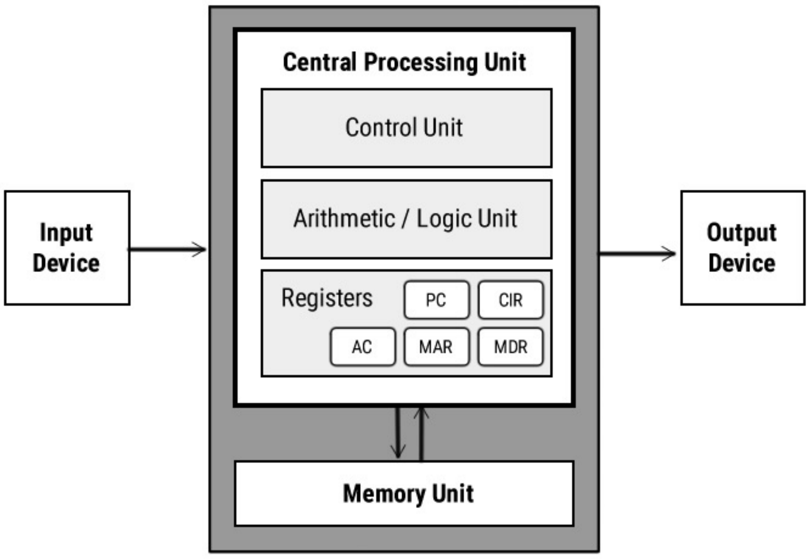
2.1.2 Graphics Processing Unit

Graphics Processing Unit (GPUs) have a Single Instruction Multiple Data (SIMD) architecture that exploits data-level parallelism. Unlike the CPU, the GPU has simple cores in an amount that exceeds a thousand units. These cores work at a low operation frequency and are typically arranged in small groups of 32 units.

Figure 2.3 shows a simplified GPU block diagram. It is possible to distinguish the work-groups formed by 32 work-items. Each Work-Item will be responsible to execute the kernel function.

In the beginning, GPUs were mostly used in computer graphics, since the SIMD

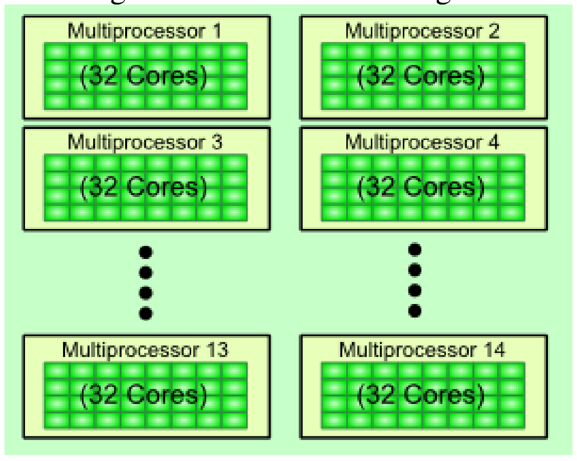
Figure 2.2: Von Neumann CPU Architecture



Source: (RESEARCH, 2019)

architecture is the most suitable to perform the same operation on a mass of data. Over the years, architectural modifications were done to permit the usage of GPUs on scientific calculations. The introduction of GPUs on the scientific area growths the necessity of a parallel programming model to properly leverage the accelerator gains. Compute Unified Device Architecture (CUDA) and OpenCL frameworks were developed to provide the programmer with an API to use the GPU as an accelerator for offloading computations from CPU. One of the major challenges involves the need to develop intrinsic parallel solutions for problems that already have been solved. GPUs fault model and reliability was extensively studied in (RECH et al., 2012; PILLA et al., 2014)

Figure 2.3: GPU Block Diagram



Source: Nvidia

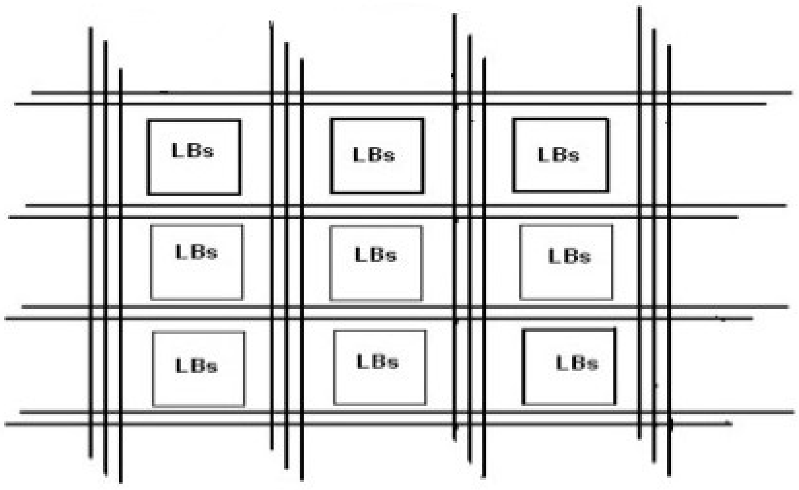
2.1.3 Field-programmable Gate Array

Field-programmable Gate Arrays (FPGAs) are integrated circuits designed to be programmed after manufacturing. Therefore, FPGAs are more flexible compute units than CPUs and GPUs. Used for digital signal processing, the FPGAs consist of an array of logic cells that can be programmed to implement logic functions.

Figure 2.4 shows the FPGA basic architecture, composed by the programmable logic block and the reconfigurable interconnections.

FPGA configurations are usually done employing a hardware description language. Reconfigurable hardware can be adapted to offer higher performances and lower power consumption than general-purpose units. However, FPGAs tend to have a limited amount of logic cells which restricts the problem-solving scope. It is important to highlight that FPGAs must be used on Big-Data processing problems or as a highly specific accelerator. FPGAs fault model and reliability was extensively studied in (WANG; LIU; SUN, 2018; WIRTHLIN et al., 2003)

Figure 2.4: FPGA Block Diagram: Logic blocks and reconfigurable interconnections



Source: Intel

2.2 Heterogeneous Devices

Heterogeneous devices were proposed as a solution to reduce energy consumption while performance is maintained or even increased. Nowadays, the widest variety of compute units are combined. Those units are characterized by having different sizes, architectures, power consumption, and even fault model. Compute Unit selection is done targeting a specific demanding purpose. Most of the heterogeneous devices introduce architectural modifications to enable the proper functioning of the device as a whole. GPUs are the most popular accelerator used in heterogeneous devices.

2.2.1 AMD Kaveri and Ryzen Accelerated Processing Units

Developed by AMD, Accelerated Processing Units (APUs) are composed of a CPU and a GPU embedded on the same die area. Both Compute Units are connected to the same memory bus, performing a Heterogeneous System Architecture (HSA). HSA decreases response time between the compute units as well as energy consumption. The CPU is combined with the GPU massive data processing capability forming a device that delivers up to twice as much performance than a single CPU.

The AMD Kaveri APU family is developed with the Steamroller Micro-architecture and the AMD Ryzen APU family with the Zen Micro-architecture, which is the last company release. Comparing the Ryzen and Kaveri APU, we can notice an increase in GPU cores and a substantial improvement in Thermal Design Power result of the 14nm lithography process. APU applications must be developed using OpenCL programming language combined with the vendor development platform. OpenCL is a C-like language that executes across heterogeneous devices consisting of CPUs, GPUs, FPGAs, and many others.

2.2.2 Nvidia Jetson Tegra and AGX Xavier

Developed by Nvidia, the Jetson Tegra and AGX Xavier are embedded devices used in autonomous vehicles. Tegra X2 was developed to low-energy budget applications that need real-time performances. This device is formed by two CPUs(Dual Denver + Quad ARM), distinguishing on the size and operating frequency, and a GPU with Pascal architecture. Two different CPUs are used to achieve the low-energy budget aims. When the exercising operation is complex, both CPUs and GPU can work together. In the other hand, when the device is performing simple tasks, the small CPU can be used. The AGX Xavier is formed by a 8-core ARM CPU and a Volta GPU with Tensor Cores. AGX Xavier incorporates a set of Deep Learning and Programmable Vision accelerators. As we can observe, newest devices incorporate more specific accelerator to achieve better performances and reduce energy consumption. Jetson applications must be developed using CUDA programming language combined with the Nvidia proprietary development platform. For being a proprietary platform, developed solutions just work under Nvidia devices.

2.2.3 Intel ARRIA 10 SoC

Developed by Intel, the Arria 10 SoC is a medium-sized device within the company's portfolio. Used mostly in telecommunications and machine learning, it can reach 1.5 Tera-FLOPS. This device combines a CPU with the flexibility and high performance that an FPGA can deliver.

FPGA's reconfigurable characteristic is fundamental for the data security area based on cryptography. The same hardware can be used to encrypt and decrypt information, always delivering the highest performances. On Tier 1 Internet server providers, FPGA's can be used as an emergency module that can back up any deficient hardware infrastructure.

On the FPGA side, applications must be developed using purely OpenCL programming language, and on the CPU, Accelerator kernels must be launched using OpenCL API and CPU application can be developed in C language combined with OpenMP framework.

2.3 Workload Distribution

Workload Distribution consists of divide the problem and assigns every portion to the most proper compute unit to be solved. To obtain the best overall performance for heterogeneous devices, one must distribute the workload between CPU and accelerators compute units. The best performance is then when all compute units are working in parallel, and the communication overhead is as low as possible. For heterogeneous devices such as APUs that share the same memory bus, the communication overhead is mainly due to the parallel algorithm structure. The usual approach of attach accelerators to host via PCI-express bus implies in additional memory transfers overhead. Thus, the workload distribution gain can be greatly improved in APUs.

The workload distribution that gives the best performance in one application may not be the best in another one because a lot of variables are involved. Each compute unit have configurable parameters, like Work-Items and Work-Groups, and these parameters directly depend on the underlying hardware. We must consider that depending on the input data size, the portion assigned to the Compute Units may vary. In addition, the number of threads that each compute unit will invoke depends on the problem assigned portion, and these relations show to be neither trivial nor entirely linear. The programmer needs to engineer the best workload distribution between the available compute units and find the best parameter configurations for each compute unit. While the hardware architecture is promising in improving the computing efficiency, it is still unclear how to find the most efficient workload distribution in a generic style.

Previous works demonstrate that an efficient workload distribution between the available compute units that consider their different computing nature can significantly improve performances (CHANG et al., 2017; BARUAH et al., 2018).

2.3.1 Collaborative Programming Model

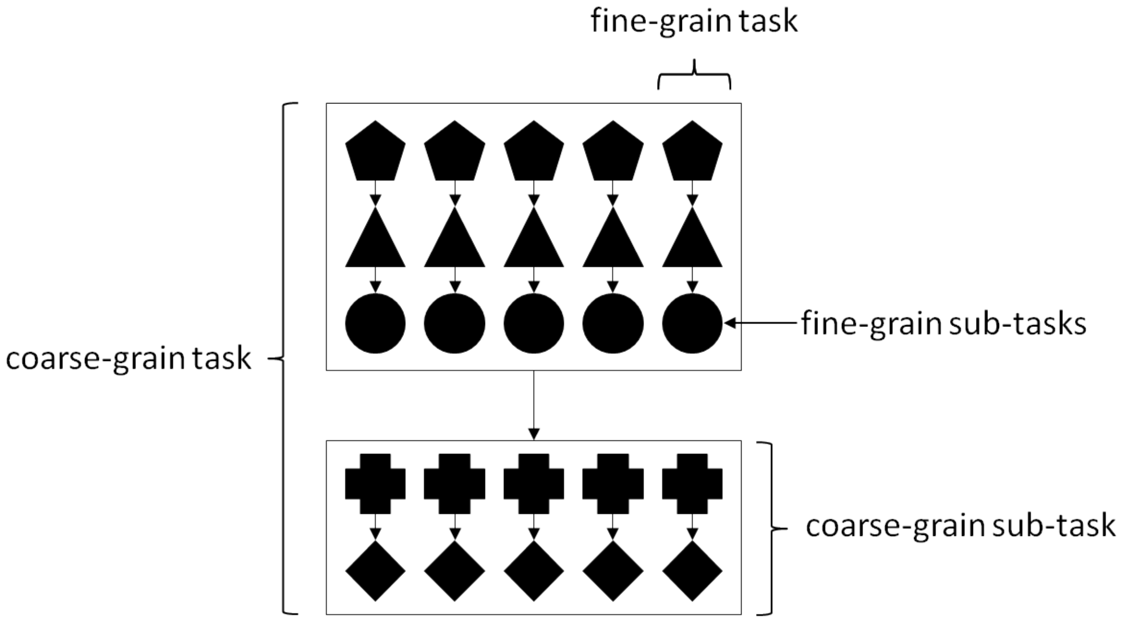
The traditional programming approach uses the accelerator to solve all the problem while the CPU just perform memory copies and control tasks. This programming model was conceived under a line of thought where the accelerator must be an external device attached via PCI-express bus to a General Purpose Unit. With the introduction of Heterogeneous devices, the traditional approach becomes obsolete since it does not take advantages of new heterogeneous features.

Heterogeneous devices performances can be improved by implementing a collaborative programming approach where the CPU is not idle while waiting for the accelerator to complete the job. Thus, the CPU executes relevant parts of the problem and work collaboratively with the accelerator to complete the job in the fastest possible way, reducing the execution time and energy consumption of the device (GÓMEZ-LUNA et al., 2017; BARUAH et al., 2018).

Techniques to divide the problem must be defined with the collaborative programming approach introduction. These can be **Data Partitioning** and **Task Partitioning**. In the former, the available cores execute the same algorithm, each on the assigned portion of data. In the latter, each core executes a task that helps the device to reach the solution(GÓMEZ-LUNA et al., 2017).

It is worth noting that some problems still lack efficient parallel algorithms which

Figure 2.5: Kernel schematic: Two sub-task with linear dependencies



Source: CHAI - ISPAS 2017

take advantage of workload distribution due to data or synchronization dependencies. Another source of inefficiency in the workload distribution is the merge operation between the results of each compute unit when output data partition is used. However, such merge overhead is a problem when the same memory address is not shared, and the host compute unit must execute a task to retrieve and merge the results from other compute units.

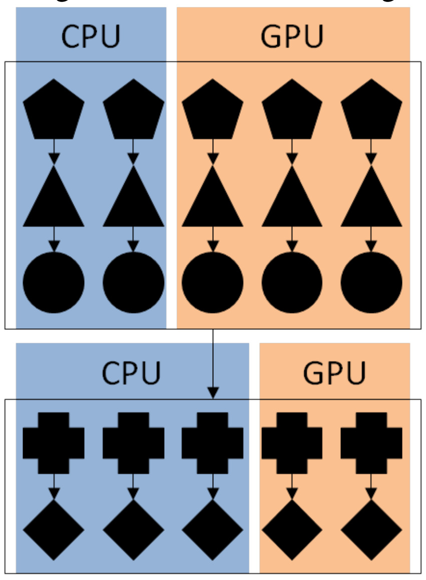
Figure 2.5 shows a Kernel tasks schematic composed of 2 sub-tasks with linear dependencies between them. Each sub-task is composed of operations represented by different geometric shapes. Each geometric shape column is a chain of dependent simple operations with the dependency going from top to bottom. Next sub-section uses Figure 2.5 to exemplify the techniques to divide a problem in order to be distributed among the compute units.

2.3.1.1 Data Partitioning

In data partitioning, all the compute units concurrently execute the same algorithm on different parts of data. Data partitioning benefits from shared memory which helps to avoid unnecessary memory operations. Heterogeneous devices, like APUs, supply the proper architecture to exploit it.

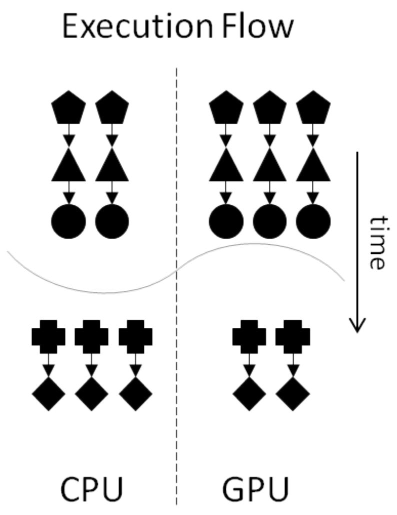
Figure 2.6 shows how data partitioning can be applied to Figure 2.5. For the first Sub-task, two geometric shapes columns were assigned to CPU while three were assigned to GPU, and for the second sub-task, the portion assigned to each compute unit

Figure 2.6: Data Partitioning



Source: CHAI - ISPAS 2017

Figure 2.7: Data Partitioning Flow



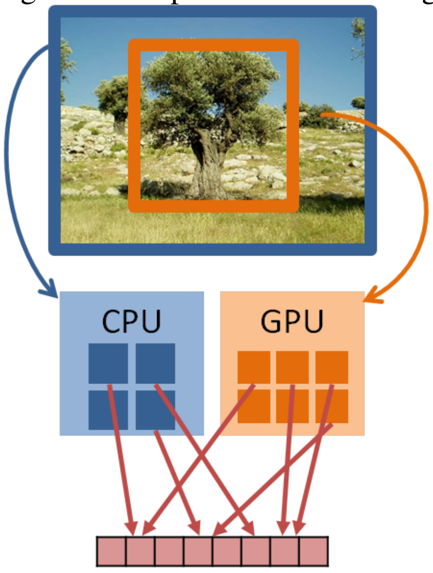
Source: CHAI - ISPAS 2017

was inverted.

Figure 2.7 shows the execution flow, where CPU and GPU concurrently work to solve the problem.

There are two possible approaches to implement data partitioning: **Input data partitioning** and **Output data partitioning**. On Input Data Partitioning, input data is divided among the compute units. For image processing, a portion of input frames can be assigned to GPU while CPU take care of the rest, or even within one frame, some pixels are assigned to the GPU while others to the CPU, this technique can be advantageous on variable rate shading. This technique is represented in Figure 2.8 where CPU take care of edge frame pixels while GPU is used on central ones.

Figure 2.8: Input Data Partitioning



Source: CHAI - ISPAS 2017

On output data partitioning, each compute unit take a part of the output result. Figure 2.9 represents this technique. Imagine we have to present an Image Histogram, to perform an output data partitioning each output bin must be distributed among the compute units. On these approach, some tones will be counted by the CPU and others by the GPU, but each pixel is computed in both compute units.

It worth nothing that on Output Data Partitioning, an extra task can be required: Merge Operation. The merge usually is a very simple task, just like concatenation or is not even necessary because each compute unit can work directly on the output results. In cases where Merge is required, micro-kernels must be executed on the output result. For example, when an image surface is calculated using output data partitioning approach, output data tiles borders must be smoothed to guarantee an optimal result.

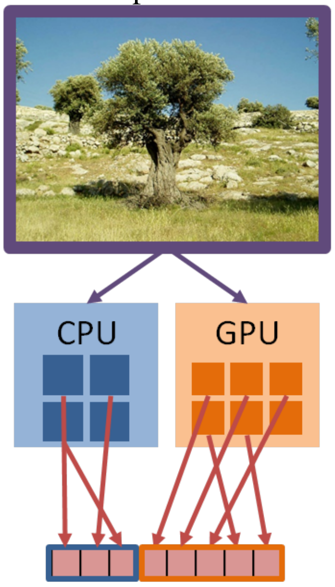
2.3.1.2 Task Partitioning

In Task Partitioning, each compute unit executes a task that helps the device to reach the solution, so different algorithms are executed at the same time. Depending on the granularity,**coarse-grain** and **fine-grain** task partitioning are possible.

Figure 2.10 illustrates how fine-grain task partitioning can be performed. Subtasks within the same fine-grain task cannot be performed in parallel due to linear dependency, subtasks from different fine-grain tasks can be parallelized, which enables concurrency.

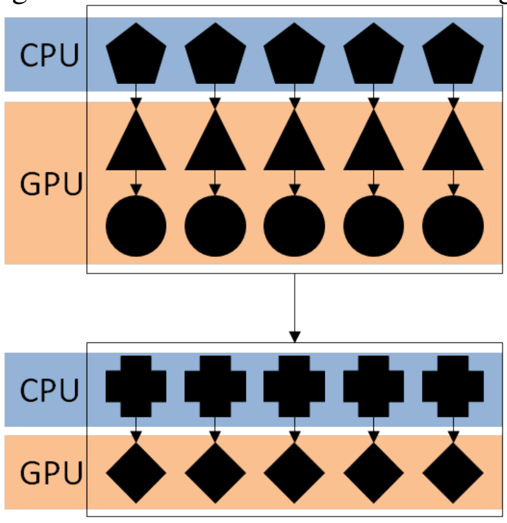
As showed in Figure 2.11, CPU and GPU executes the second coarse-grain subtask just when the first one is completed. Fine-grain task partitioning is effective when the

Figure 2.9: Output Data Partitioning



Source: CHAI - ISPAS 2017

Figure 2.10: Fine Grain Task Partitioning

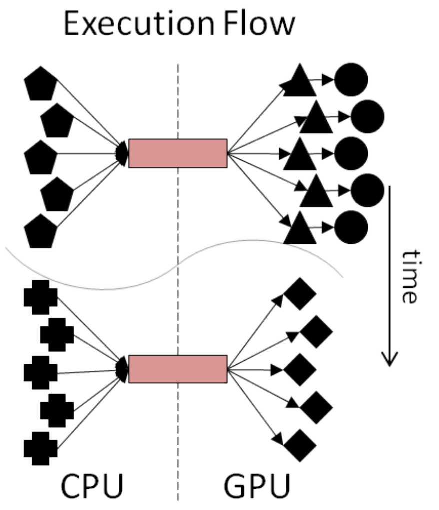


Source: CHAI - ISPAS 2017

heterogeneous device is composed of very specialized compute units like cryptography modules, and 4K video encoder/decoder.

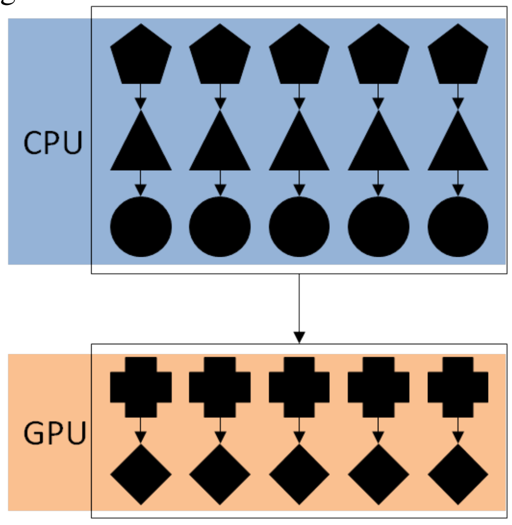
Figure 2.12 illustrates how coarse-grain task partitioning can be performed. Each coarse-grain subtask is assigned to a compute unit. In this partitioning style, various compute units cannot execute the coarse-grain subtasks concurrently due to linear dependency, but concurrency can be achieved using multiple datasets in a pipeline fashion. Coarse-grain task partitioning must be used when the heterogeneous device is composed of more general purpose compute units or when fine-grain task is not reachable.

Figure 2.11: Fine Grain Task Partitioning Flow



Source: CHAI - ISPAS 2017

Figure 2.12: Coarse Grain Task Partitioning



Source: CHAI - ISPAS 2017

2.4 Performance and Energy Consumption

Energy consumption and performance are two of the main constraints for almost all computing devices. Low power is the major concern for embedded systems, like pacemakers and tablets, while performance (i.e., execution time) is fundamental for HPC and real-time application like pedestrian recognition on autonomous cars. In recent years, energy consumption has become a significant concern for HPC systems as well. Once supercomputers approach exaFLOP, the energy amount that can be consumed by the single device must be reduced. The U.S Department of Energy (DOE) imposes a power limit of 20 MW for exascale supercomputers (LUCAS, 2014).

Some works consider the Energy-Delay Product (EDP) (LI, 2008; CONG; YUAN,

2012) of heterogeneous devices and try to improve the overall efficiency, such as in (BARUAH et al., 2018). The authors developed a framework using machine learning prediction models combined with application runtime feedback based on Dynamic Voltage and Frequency Scaling to improve the EDP metric. The usage of this framework derives in approximately 20% of EDP improvements.

For limited energy budget projects like Mars Rover Opportunity, performances cant be neglected due to limited power, so an intermediate point must be found and EDP metric can help on that finding. Unfortunately, the Opportunity rover stops its functioning due to lack of batteries due to environmental problems which affected it recharge solar panels.

Heterogeneous architectures features, like shared memory and system-wide atomic, were mainly planned and designed to improve performance and reduce energy consumption at the same time. It's also important to note that heterogeneous architectures usually demands smaller silicon die areas and those enhancements directly impact on device performance.

There are very established frameworks to measure power and energy consumption. On heterogeneous devices, power/energy efficiency were extensively evaluated by previous works (BARUAH et al., 2018; HU; QUAN; LU, 2018; KLIAZOVICH; BOUVRY; KHAN, 2012).

2.5 Reliability

Nowadays, reliability is one of the major concerns for safety-critical, real-time, and HPC applications. As a reference, the Department of Energy's system, Titan, composed of more than $18,000$ NVIDIA GPUs, has a Mean Time Between Failures (MTBF) in the order of dozens of hours (GOMEZ et al., 2014; TIWARI et al., 2015). In (OLIVEIRA et al., 2017b) the authors project an error rate of one SDC or DUE every day for some applications running in an Intel Xeon Phi Knights Landing accelerator. In this scenario, a lack of understanding of HPC device resilience may lead to lower scientific productivity and significant monetary loss (SNIR et al., 2014). As heterogeneous devices move into automotive, military, aerospace,and HPC markets, reliability questions start to arise. We cannot trade-off performances for reliability in safety-critical applications.

The most common ways to evaluate a device or software reliability are accelerated neutron beam tests and software fault injections. Beam tests emulate the natural radiation effects in all the available computing resources providing the realistic error rate of a device expressed in **Failure in Time** (FIT, errors per 10^9 hours of operation) (BAUMANN, 2005). However, as errors are observed only at the output, beam tests offer limited faults visibility. With fault injections it is possible to calculate the **Program Vulnerability Factor** (PVF), which is the probability for a fault program state to affect the output correctness (SRIDHARAN; KAELI, 2009).

The workload distribution impacts the number of resources required for computation (and, then, the FIT) as well as the probability for a fault to propagate (so the PVF). However, the workload distribution also impacts performances. Neither the FIT or PVF are directly related to the code execution time. A configuration may have a higher error rate but also a shorter execution time, which means that a higher number of executions may be completed between two errors. To also consider execution time in our reliability analysis we use the **Mean Executions Between Failures** (MEBF) (REIS et al., 2005; RECH et al., 2014). The higher the MEBF the higher the code reliability, which is calculated dividing the Mean Time Between Failure (MTBF) (inversely dependent on the FIT) by the code execution time, MEBF is the number of correct executions completed between two output errors.

The authors in (JEON et al., 2013) brought to discussion the reliability of APU hardware structures calculating their Architectural Vulnerability Factor (AVF) when executing a set of heterogeneous benchmarks. These preliminary results show that the CPU core has a significantly higher AVF than the GPU, about $3\times$. However, the study does not take advantage of a collaborative approach as CPU and GPU workloads were executed sequentially (DANALIS et al., 2010; SUN et al., 2016).

2.5.1 Radiation Induced Soft Errors and Fault Injection

Radiation-induced errors are the main source of errors for computing devices (BAUMANN, 2005). About 13 $neutrons/(cm^2 \times h)$ reach ground in consequence of galactic cosmic rays interaction with the terrestrial atmosphere. A neutron strike may perturb a transistor's state generating bit-flips in memory or current spikes in logic circuits that, if latched, lead to an error. A radiation-induced transient error leads to: (1) no effect on the program output (i.e., the fault is masked, or the corrupted data is not used), (2) a Silent Data Corruption (SDC) (i.e., an incorrect program output), or (3) a Detected Unrecoverable Error (DUE) (i.e.,a program crash or device reboot).

The raw radiation sensitivity strongly depends on the device physical implementation (BAUMANN, 2005) while the probability for faults in low-level resources to propagate at visible states depends on both the device architecture (MUKHERJEE et al., 2003a) and on the executed code (SRIDHARAN; KAELI, 2009). However, some reliability behaviors have been shown to be shared among different devices and algorithms (FRATIN et al., 2018a).

Radiation experiments aim at measuring the *cross section*, which is the sensitive area of the device. i.e., that portion of area that, if hit by an impinging particle, causes an observable failure. When an algorithm is tested, the cross section is measured dividing the observed error rate ($errors/s$) by the average particle flux ($particles/(cm^2 \times s)$), yielding an area. The larger the cross section, the higher the use of sensitive resources (BAUMANN, 2005; MUKHERJEE et al., 2003a). So, the cross section depends on the overall amount of resources required for computation and on their criticality, but does not include any information on execution time. Multiplying the cross section (cm^2) with the expected neutron flux on the device ($13n/(cm^2 \times h)$ at sea level (JEDEC, 2006)), we can estimate the device error rate or Failure In Time (FIT), expressed as $errors/10^9h$.

Fault Injection is a technique for improving the device or software reliability that consists of introducing faults on device or software resources to better understand transient errors propagation and provide insights on how to mitigate their effects. Fault Injection can be performed to identify the portions of high-level code which are more critical for application execution. Fault injections are used to better understand the different faults propagation when the code is executed on the CPU or GPU.

3 METHODOLOGY

This Chapter presents the benchmarks and device considered for evaluation as well as the experimental methodology used to measure the execution time, energy consumption, and, mainly, the radiation test setup.

This thesis focus on data partitioning for two main reasons: (1) the architectural characteristics of the APU, specifically the shared memory between CPU and GPU, and (2) in the tasks distribution the CPU and GPU would execute significantly different codes, making it harder to understand if the observed differences in FIT rates are related to the core, the code, or the combination of them.

Data partitioning can be obtained by partitioning the input or the output data. We test both input and output data partitioning. When output data is partitioned, a merge operation is required when both cores complete the execution, which is not required for input data partitioning (GÓMEZ-LUNA et al., 2017).

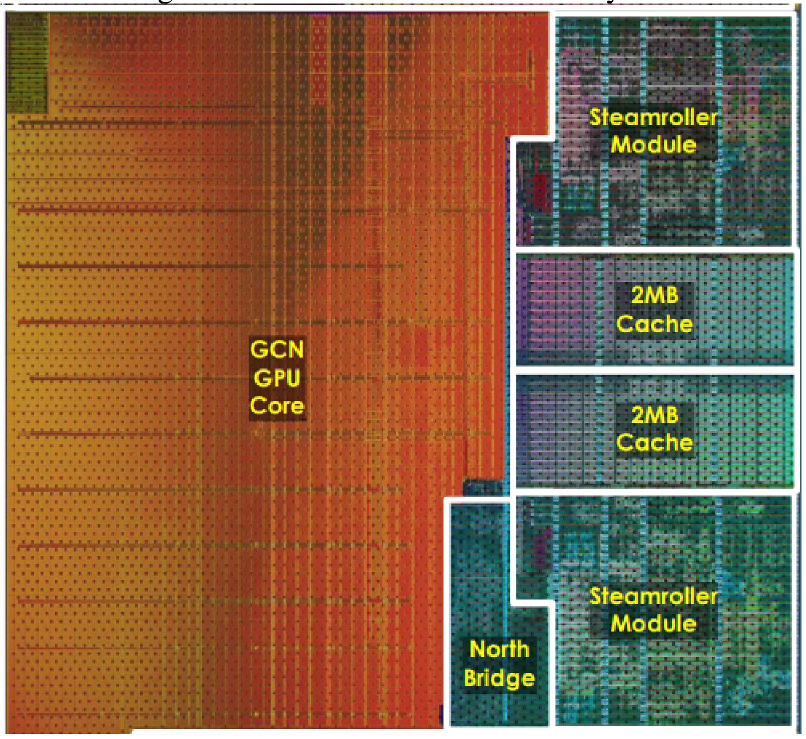
To evaluate the impact of collaborative Worload Distribution on heterogeneous devices reliability we selected a subset of codes from the Collaborative Heterogeneous Applications for Integrated Architectures Benchmarks (CHAI) (GÓMEZ-LUNA et al., 2017). As detailed in the following, each one represents a different class of application and is likely to stimulate specific resources on the tested device. Hence, we believe that the results could be generalized to applications with similar computing characteristics.

3.1 Tested Device

The proposed evaluation is performed on AMD A10 7890K Kaveri APUs. These devices include four Steamroller CPU cores and an AMD Radeon R7 Series GPU containing 512 cores with a nominal working frequency of 866MHZ. The APU has two sets of 2MB shared L2 caches. Both CPU and GPU share 8GB of DDR3 memory and the device power consumption fluctuates between 90 $\sim$ 95W, which makes the AMD APU an optimal choice for low power HPC applications (FRATIN et al., 2018b). The AMD APU peculiarity among other heterogeneous devices is to be Heterogeneous System Architecture (HSA) compliant, as memory is shared between compute units. HSA reduces chip area and power consumption by removing costly device-to-device data transfers.

Figure 3.1 shows AMD Kaveri APU die area layout. The GPU core has a $3\times$ larger area than the two steamroller CPU cores. The cache memory also represents a

Figure 3.1: AMD Kaveri APU Die layout.



Source: (RESEARCH, 2016)

considerable part of the total die area. As discussed previously, the probability of faults may depend on the cores area.

3.2 Tested Benchmarks

We select a subset of codes from the Collaborative Heterogeneous Applications for Integrated Architectures (CHAI) Benchmarks (GÓMEZ-LUNA et al., 2017). We select four codes, each representative of broader classes of applications. For the 4 codes selected, we perform a workload distribution starting at CPU and going towards GPU gradually.

Bezier Surface (BS) is a compute-bound algorithm widely used in computer graphics. The algorithm is a generalization of Bezier curves where a surface is stretched according to control points performing tensor-product operations. The collaborative approach on BS performs *output data partitioning* dividing output data into tiles and assigning them to the CPU or the GPU. In this approach, both compute units work concurrently to achieve the result. A merge operation is required to connect the edges of the output tiles when the workload is distributed. Thus, the program needs to execute more operations when both compute units are used, and the FIT may increase according to the reliability of the merge operation. **Stream Compaction (SC)** is a memory-bound algorithm commonly used in databases and image processing applications. SC is composed of a data manipulation primitive that remove elements from an array, keeping only those satisfying an input predicate. The *input data partitioning* collaborative approach on SC divides the input array into tiles and distribute them among the CPU and GPU. The memory-bound characteristic of this benchmark makes it more suitable for the CPU, which have a more advanced memory hierarchy. The GPU will have to idly wait for data which may increase fault masking resulting in a lower FIT at the cost of execution time. **Canny Edge Detection (CED)** is a technique to extract useful structural information from images and reduce the amount of data to be processed. CED consist of four kernels: (1) a Gaussian filter, (2) a Sobel filter, (3) non-maximum suppression, and (4) hysteresis . The input frames are a subset of Urban Dataset used for training classification neural networks (FRAGKIADAKI et al., 2012). The *input data partitioning* collaborative approach on CED divides the input frames between the CPU and GPU. The CPU and GPU work concurrently to achieve the result and no communication is needed between them. Since all compute unit are executing intensive operations, consequently increasing the resources usage, the FIT may increase when both compute units are working.

Breadth First Search (BFS) is a search in graphs algorithms that performs non-uniform memory access widely used in GPS Navigation Systems. We use a hierarchical queue-based version of graph traversal algorithm. As graphs are irregular, the number of nodes and edges to be processed at a given time may vary. The BFS benchmark invoke continuously the kernel functions when workload distribution occurs. Thus, this increase in synchronization points between CPU and GPU may impact negatively the FIT rate. The input graph we select for our evaluation represents the highways, streets, and bridges of Great Lakes area in the US (9TH..., 2006) It is known that small frontiers are more efficiently processed on the CPU while large ones are better suited for the GPU (LUO; WONG; HWU, 2010). Therefore, the *input data partitioning* collaborative approach is implemented by selecting a threshold, that depends on the nodes that have to be processed on the next frontier. The BFS is the only benchmark where kernel functions are invoked continuously when CPU and GPU are distributing the workload. Thus, this increase in synchronization points between CPU and GPU may lead to a higher DUE FIT rate as well when both compute units are working together.

3.3 Energy and Execution Time Measurements

To measure execution time, we execute 500 times every benchmark configuration, in order to eliminate cold start bias and reduce the measurement error. For energy consumption measurements we use AMD's CodeXL. Code XL is an open-source development tool suite that includes both CPU and GPU profilers (DEVICES, 2018). Average power and total consumed energy of each compute unit can be obtain using CodeXL profilers. For our experiments, CodeXL was configured to perform 10 samples per second on both CPU cores, integrated GPU, PCI-E and memory controllers. It is important to mention that memory RAM consumptions was not included and no PCI-E auxiliary board was attached to the device. To obtain the average power, we calculate the average of selected power counters during the execution time. Finally, with the average power measurement, we can compute the energy consumption.

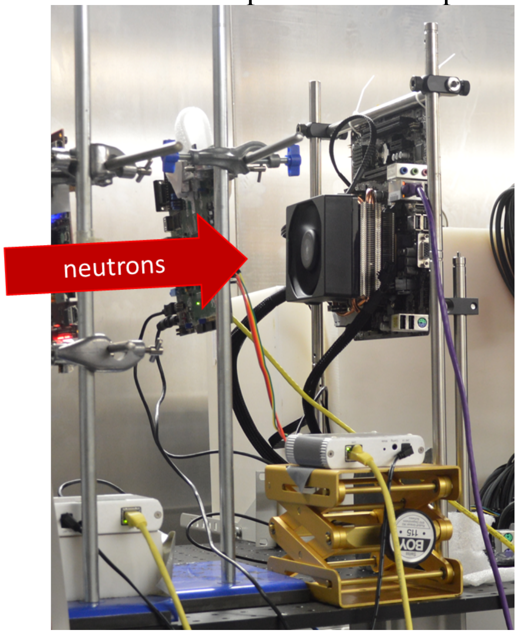
3.4 Fault Injection

For the software fault-injection campaign we use CAROL-FI (OLIVEIRA et al., 2017a). As demonstrated in (OLIVEIRA et al., 2017b), CAROL-FI can obtain useful information about the error propagation paths and the authors also performed radiation experiments alongside fault injection campaigns in the same device. CAROL-FI inject faults at the source code level, measuring the PVF, this allows identifying the portions of high-level code which are more critical for the application execution. It is worth noting that, as we do not have access to the AMD proprietary architectural-level fault injector, we can inject only at source code level, accessing the PVF and not the AVF. However, the PVF still can predict the vulnerability of a code executed in a specific architecture (SRIDHARAN; KAELI, 2009).

CAROL-FI is built upon GNU GDB and have four different fault models to simulate error propagation: (1) **Single** that flips a single random bit, (2) **Double** that flips two random bits from the same variable,(3) **Random** that overwrite every bit by a random bit and (4) **Zero** that set every bit to zero.

For the fault injection campaign, we configure CAROL-FI on **Random** fault model and inject 80000 faults, proportionally distributed among all tested configuration.

Figure 3.2: Part of the experimental setup at ChipIR



Source: The Author

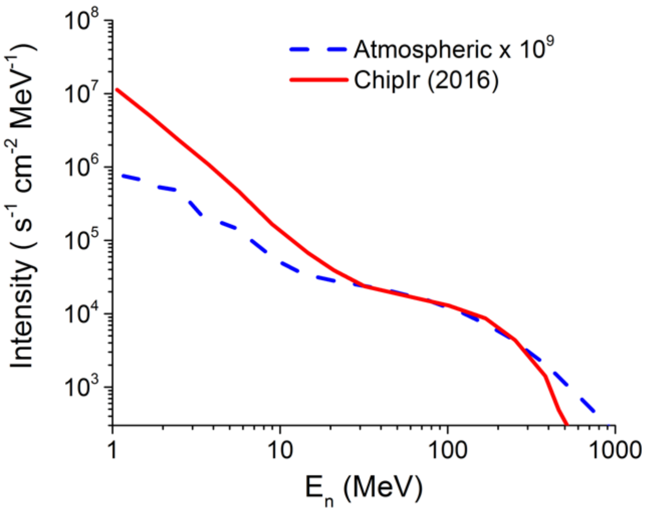
3.5 Radiation Beam Experiments

The error rate of a device is strictly related to the number of resources used for computation (BAUMANN, 2005). From Figure 3.1 we can predict that the GPU is expected to have a higher fault rate than the CPU, and a significant amount of faults may occur in the caches. However, a fault does not necessarily propagate to the output. Faults in unused resources or obsolete data will not impact computation. Moreover, some faults can be masked during computation, without affecting the output. The probability for a fault to affect the output depends on the architecture itself (MUKHERJEE et al., 2003b). In other words, each core has a different architecture, therefore different fault model.

To have a realistic evaluation of the influence of workload distribution in the device error rate we take advantage of particle accelerator facilities. These accelerators produce a neutron beam that mimic the energy spectrum of atmospheric neutrons. The flux of particle provided by accelerators is millions of times higher than the terrestrial one: one hour of a test at these facilities represents many hundreds of years of natural exposure, allowing to gather a significant amount of data in a short time.

The radiation experiments presented in this thesis were performed at the ChipIR facility at the Rutherford Appleton Laboratories, Didcot, UK. Figure 3.3 shows that ChipIR provides a white neutron source that emulates the energy spectrum of the atmospheric neutron flux from 10 to 750 MeV. Moreover, the beam at ChipIR was empirically demonstrated to mimic the terrestrial radiation environment (CAZZANIGA;

Figure 3.3: ChipIR neutrons spectrum of energy compared to the terrestrial one.



Source: The Author

PLATT; FROST, 2013). The flux of neutrons was about $2.5 \times 10^6 n/(cm^2/s)$, while the atmospheric neutron flux at sea level is about $13n/(cm^2/h)$ (JEDEC, 2006). To scale our results to the natural environment, we tune our setup to avoid the possibility of more than one neutron to generate an error in the device per code execution. Tests were conducted for more than 100 hours per configuration. When scaled to the natural environment, our results cover at least 1×10^8 hours of normal operations per configuration. As we tested 20 configurations, our data covers at least 220,000 years of normal operation.

Figure 3.2 shows part of our setup at ChipIR. We have aligned two APU with the neutron beam to increase statistics, de-rating flux intensity based on distance. To evaluate the error rate of the device we run the selected codes and configurations while the device is irradiated with neutrons. We then compare each output with a pre-computed golden copy to detect Silent Data Corruption (SDC). A software and hardware watchdog detects eventual application crashes or device hangs, triggering a new execution of the application or a device power-cycle.

4 RESULTS

In this Chapter, we report and discuss collaborative Workload Distribution performances and error rates measured by radiation experiments and fault injections. For each benchmark, we test from three up to six different workload distribution configurations with a total of 20 different configurations. Due to beam time restriction, we were unable to gather results for more configurations. Additionally, using the MEBF metric we can identify the most reliable workload distribution.

Table 4.2 depicts the radiation experiments result for the four benchmarks tested. The second column shows the Workload Distribution between CPU and GPU. The third column presents the energy consumption in joules. The fourth column presents the execution time in seconds. The fifth and sixth columns present the SDC FIT and DUE FIT rate. Finally, the last column shows the MEBF. The FIT rate data have been normalized to the lowest SDC FIT rate we have measured. This normalization allows a direct comparison of the FIT rates of the different codes and different configurations without revealing business-sensitive data. The MEBF metric is computed using the normalized FIT rate allowing only a comparison within the presented results.

Table 4.1: PVF for the tested codes

Code		CPU PVF	GPU PVF
	SDC	3.05%	0.40%
BS	DUE	9.38%	28.50%
	SDC	0.35%	0.93%
SC	DUE	39.7%	48.8%
	SDC	2.5%	1.03%
CED	DUE	17.70%	14.07%
	SDC	1.4%	2.3%
BFS	DUE	13.2%	28.1%

Source: The Author

4.1 Energy Consumption and Execution Time

The energy consumption and delay for all four applications are presented in Figures 4.1, 4.2, 4.3 and 4.4. The X-axis starts with the workload entirely assigned to CPU (100%CPU, 0%GPU) and gradually shares the workload with the GPU up to the other ex-

Table 4.2: Energy Consumption,Execution Time, FIT, and MEBF for the tested codes.

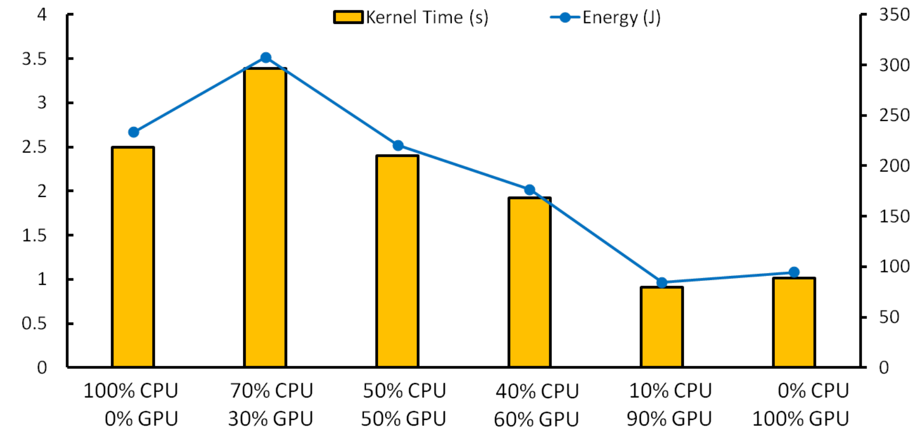
Code	WD CPU-GPU	Energy Consump. [J]	Exec. Time [s]	SDC FIT [a.u.]	DUE FIT [a.u.]	MEBF [a.u.]
BS	100%-0%	234	2.5	19.3	1.35	74.5
	70%-30%	308	3.4	21.1	1.36	50.3
	50%-50%	220	2.4	20.1	1.88	74.3
	40%-60%	177	1.9	26.4	2.74	70.7
	10%-90%	84	0.9	26.5	3	149.7
	0%-100%	94	1	14.6	2.9	242.3
SC	100%-0%	17	0.17	83	3.77	261.1
	70%-30%	11	0.11	50.4	2.47	620.3
	50%-50%	19	0.19	38.1	2.19	507.1
	40%-60%	22	0.22	34.3	2.14	485
	0%-100%	37	37	37.7	1.01	263.3
CED	100%-0%	525	5.9	3.7	3.46	165.8
	70%-30%	411	4.6	4.9	3.44	159.1
	40%-60%	242	2.7	4.3	3.57	312.4
	10%-90%	67	0.7	1.3	3	3549.4
	0%-100%	10	0.11	1.7	3.68	14749.5
BFS	100%-0%	79	0.94	1.2	0.53	3160.8
	50%-50%	31	0.35	2.5	4	4136.2
	0%-100%	28	0.30	1	5.1	11920.5

Source: The Author

treme where the entire workload is now assigned to the GPU (0%CPU, 100%GPU). The trend between the two metrics is very similar for all benchmarks. As demonstrated in previous work (BARUAH et al., 2018), power and energy management systems have inefficient performances when collaborative workloads are executed. For instance, when workload distribution is assigned entirely to CPU (100%GPU, 0%GPU) or GPU (0%CPU, 100%GPU), both compute units are powered ON even if no job is assigned to them. This power management limitation is also presented in other SoC Devices (NVIDIA, 2018). Furthermore, the frequency of activation of the power management system is higher than many benchmarks execution time. As a result, the benefits of a dynamically reduced power due to idle compute units is negligible for codes with short execution times. These issues result in approximately constant power usage during benchmarks execution leading to the execution time as the main factor for the energy consumption computation.

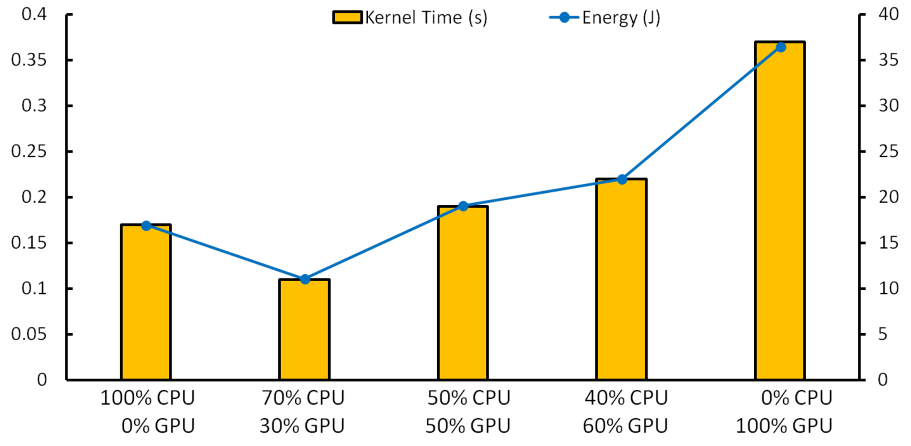
For three out of four benchmarks, the best Workload Distribution configurations regarding energy and execution time are the closest to the workload entirely assigned to the GPU (0%CPU, 100%GPU). For BS, when CPU is working with 10% and the GPU with 90% of the workload (10%CPU, 90%GPU), it is possible to obtain 10% of performance gain comparing to the workload entirely executed in the GPU. The heterogeneous device architecture with shared memory spaces providing negligible communication overhead makes it possible to obtain such gains, which would be much less efficient if memory transfers are needed. However, for CED and BFS the configuration that assigns the entire

Figure 4.1: Energy Consumption and Execution Time for Bezier Surface.



Source: The Author

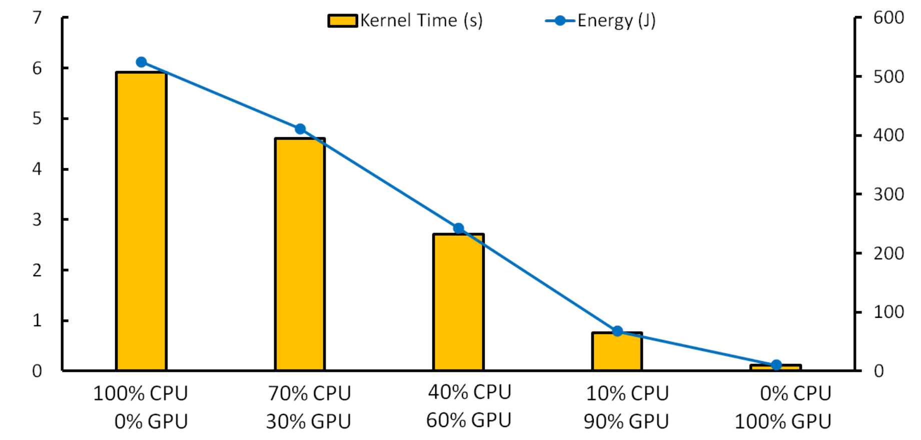
Figure 4.2: Energy Consumption and Execution Time for Stream Compaction.



Source: The Author

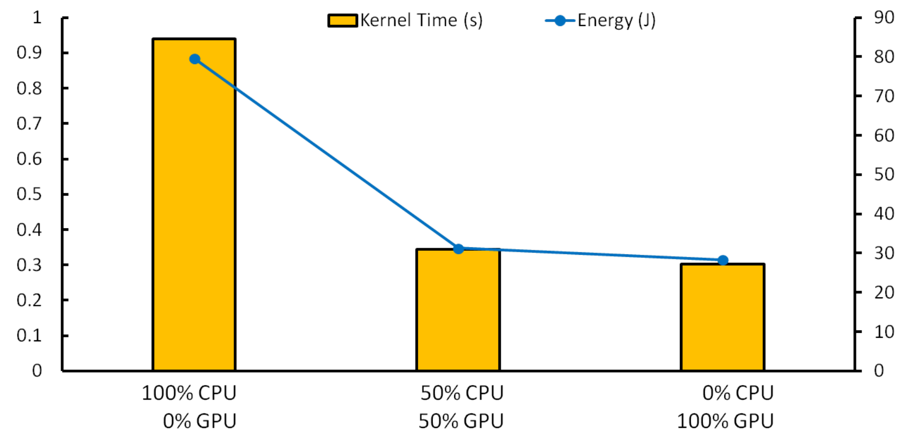
workload to GPU presents the best performances. CED is an image processing algorithm and, then, have a much higher performance in a Single Instruction Multiple Data (SIMD) compute unit as the GPU. The BFS algorithm implementation highly benefits from parallel structures, and the compute-bound characteristics are better explored in GPUs (LUO; WONG; HWU, 2010). Finally, for SC we have a mix of memory- and compute-bound operations and the best Workload Distribution is at 30%GPU workload distribution configuration (70%CPU, 30%GPU), the balance is found delivering 30% of performance gains regarding the second best configuration.

Figure 4.3: Energy Consumption and Execution Time for Canny Edge Detection.



Source: The Author

Figure 4.4: Energy Consumption and Execution Time for Breadth First Search.

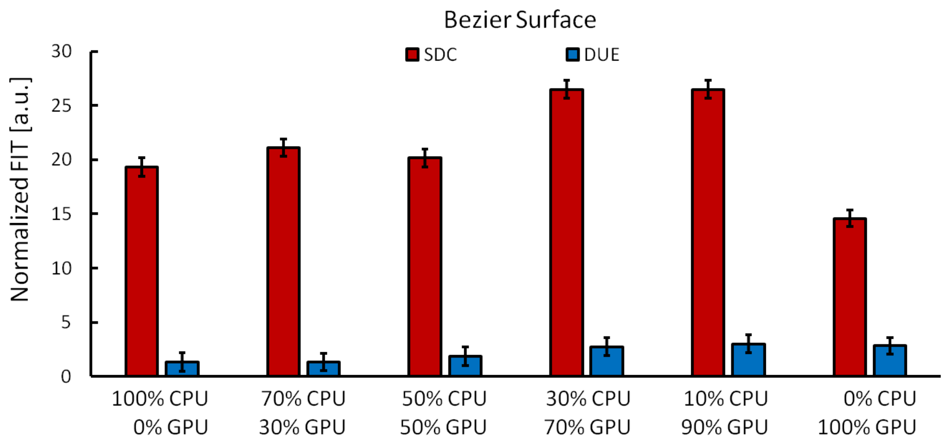


Source: The Author

4.2 Error Rate

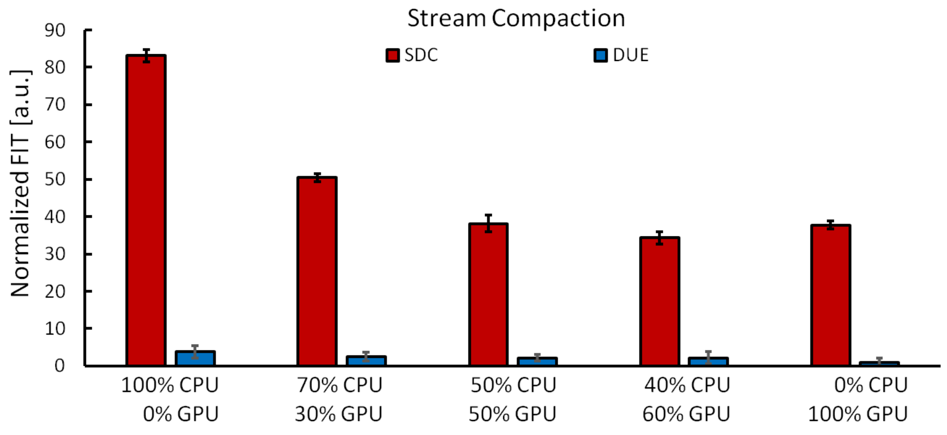
As discussed in Section 2.5, the reliability of a device depends both on the amount of resources required for computation and on how faults in a resource propagate to the output. The SDC FIT rate, then, strongly depends on the algorithm that is being executed (FRATIN et al., 2018a; SRIDHARAN; KAELI, 2009; MUKHERJEE et al., 2003b) and on the workload distribution. When the workload is assigned entirely to one of the compute units, the other compute units structures are in idle, and impinging particles in such structures are not expected to impact the code output. Using both CPU and GPU collaboratively increases the number of exposed resources which may incur in an increase of the FIT rate. However, both compute units have different architectures, and thus different radiation sensitivity (MUKHERJEE et al., 2003b). Faults in GPU structures are less likely to impact the output than a fault in the CPU structures (JEON et al., 2013).

Figure 4.5: FIT rates for Bezier Surface



Source: The Author

Figure 4.6: FIT rates for Stream Compaction

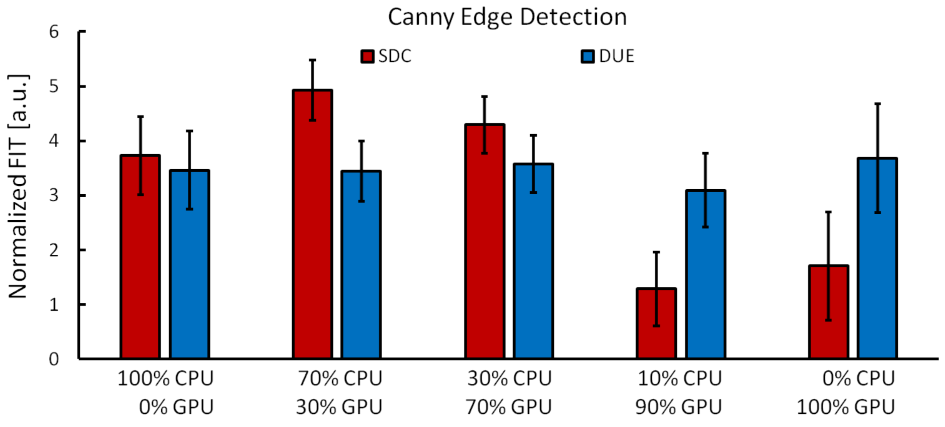


Source: The Author

We have performed fault-injection experiments using the CAROL-FI as described in Section 2.5. Table 4.1 shows the SDC and DUE PVF rates for all codes with the workload assigned entirely to one of the cores (100% CPU or 100% GPU).

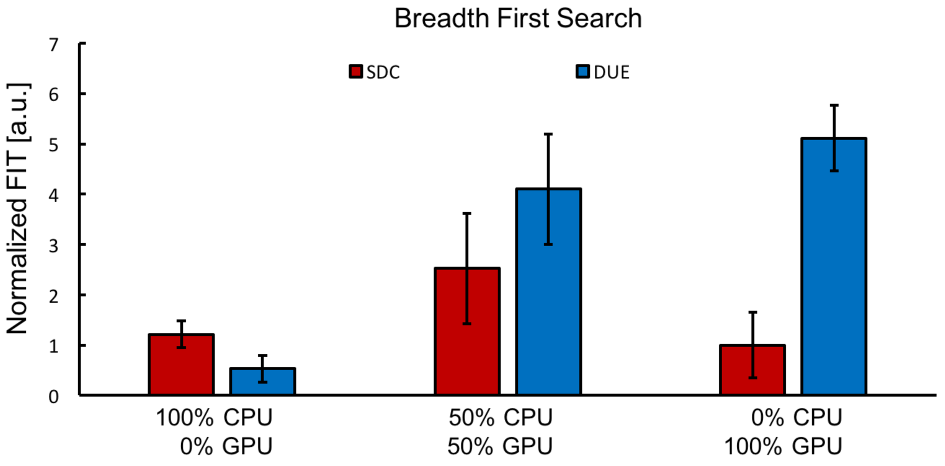
Figures 4.5, 4.6, 4.7, and 4.8 show the SDC and DUE FIT rates for the experimental results of each tested code with different collaborative Workload Distributions, from the whole workload assigned to the CPU (100% CPU, 0% GPU in the Figures) to the whole workload assigned to the GPU (0% CPU, 100% GPU in the Figures). Not surprisingly both the SDC and DUE rates change from code to code and from configuration to configuration. However, the DUE rate is less dependent on the executed code or configuration than the SDC rate. The average DUE FIT rate variation among the tested codes is about 35% while the average SDC rate varies of more than 100%. The DUE rate has a component that is independent of the executed code and derives from the underlying hardware characteristics such as control logic, synchronizations, and interfaces (FRATIN et al., 2018a). DUEs can also be generated by faults affecting the instruction cache or the

Figure 4.7: FIT rates for Canny Edge Detection



Source: The Author

Figure 4.8: FIT rates for Breadth First Search



Source: The Author

code execution but, as shown in (FRATIN et al., 2018a), DUE is mostly independent on the executed code. The DUE PVF shown in Table 4.1 could not replicate the behaviour observed during radiation experiments, confirming that one must access the hardware components to replicate the reliability behaviours.

BFS DUE FIT rate shows an increasing trend when we increase the workload computed by the GPU (refer to Fig. 4.8). The BFS code executes the GPU kernel only when the number of nodes to be processed surpasses a predefined threshold, resulting in several GPU kernel launches and stressing the synchronization between the CPU and GPUs as explained in Section 3.2. Thus, as the GPU workload increases, the number of kernel launches increases as well leading to more DUEs. On the other hand, the remaining codes DUE rate have no statistical difference since we have only one kernel launch independent of Workload Distribution.

SDC FIT rate, in contrast to DUE FIT rate, strongly depends on the algorithm that is being executed (FRATIN et al., 2018a; MUKHERJEE et al., 2003b; SRIDHARAN;

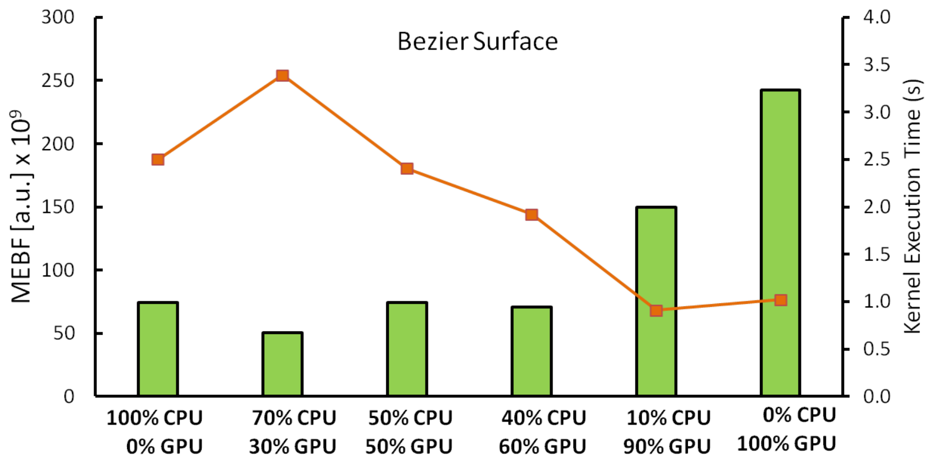
KAELI, 2009) and on how the workload is being distributed. First, if the workload is assigned entirely to one of the cores (100% CPU or 100% GPU) the other core is in idle, meaning that data errors are not expected to impact the code output. In other words, using collaboratively the CPU and GPU is likely to increase the number of exposed resources and, thus, the SDC rate. Second, even if the executed code is the same, CPU and GPU have remarkably different AVFs (JEON et al., 2013). This different AVF means that a fault in the GPU resources is less likely to impact the output than a fault in the CPU.

Figures 4.5 and 4.7 shows the BS and CED SDC FIT rate respectively, we can see that GPU is more reliable than CPU when there is no Worload Distribution, which agrees with the PVF results in Table 4.1 and the AVF results in (JEON et al., 2013). When the workload is distributed between CPU and GPU, the SDC rate increases since the code executes concurrently and more resources are now used increasing the probability of a fault to cause SDCs. The only exception is for CED with the Workload Distribution of 10% CPU and 90% GPU which presents a lower SDC FIT rate than CPU or GPU. However, as the error bars demonstrate, due to time limits for radiation experiments we could not gather enough data to have a statistical sound analysis for this configuration.

SC SDC FIT rate can be seen in Figure 4.6. GPU also prove to be more reliable than CPU, but sharing the workload between CPU and GPU does not increase the overall SDC FIT rate, but we can see a downward trend towards the GPU SDC rate. The SC code is extremely simple, and memory operations dominate most of the time. Thus, when we use more the GPU, we only increase the amount of time spent with memory operations while decreasing the time with other operations that could cause SDCs, then, faults in unused resources during idle time will likely be masked. Additionally, the SC SDC PVF in Table 4.1 shows an opposite trend where the SDC PVF is lower for CPU. However, since SC depend heavily on memory operations which is not accessible at source code level, the PVF analysis cannot observe architectural trends that compose the most significant portion of this benchmark.

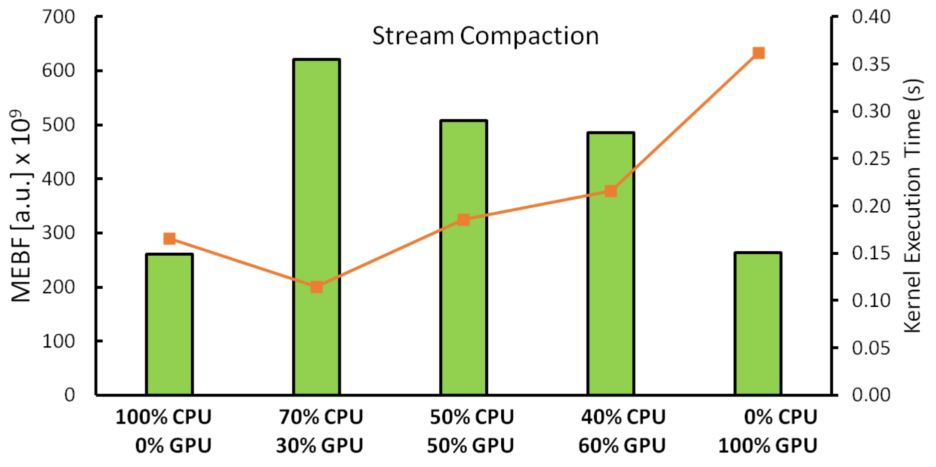
Figure 4.8 present the SDC FIT rate for the BFS benchmark. BFS shows a similar trend to BS and CED when the workload is shared between CPU and GPU. Thus, using more compute units concurrently we are increasing the area leading to a higher SDC FIT rate. We can also see in Figure 4.8 the GPU as the most reliable compute unit, but the PVF results in Table 4.1 shows the opposite trend. The BFS, similar to SC, relies extensively on architectural characteristics since BFS invokes the GPU kernel for each step of the computation, even if the workload is performed 100% in CPU. Thus, the reliability of this

Figure 4.9: MEBF for Bezier Surface



Source: The Author

Figure 4.10: MEBF for Stream Compaction



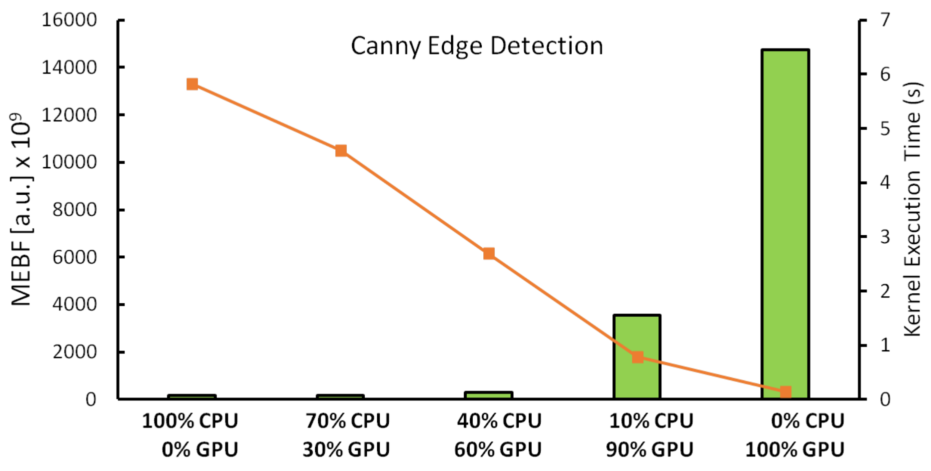
Source: The Author

synchronization at each step cannot be captured by PVF analysis since the resources are not accessible at source code level.

4.3 Mean Execution Between Failure

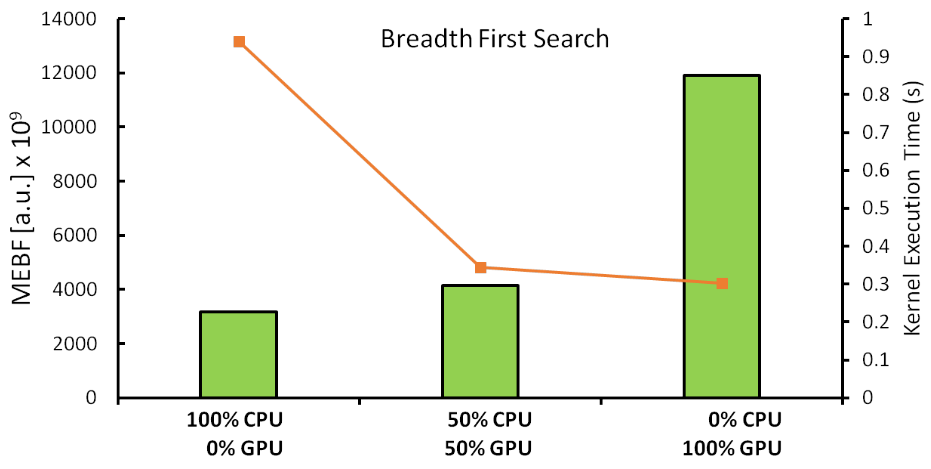
FIT is the universally used metric to compare the reliability of devices and algorithms (JEDEC, 2006). However, FIT depends only on the sensitivity of the used resources and on the probability of faults to affect the output without directly considering other important factors as execution time. To have an operative evaluation of the workload distribution that guarantees highest reliability we consider the Mean Executions Between Failures (MEBF) metric. MEBF is the number of correct executions completed between two failures and is measured dividing the Mean Time Between Failure (MTBF) (inversely proportional to FIT) with the code execution time (reported in Table 4.2). The higher the

Figure 4.11: MEBF for Canny Edge Detection



Source: The Author

Figure 4.12: MEBF for Breadth First Search



Source: The Author

MEBF the higher the amount of useful data produced between failures and, thus, the higher the reliability.

As detailed in the previous chapters, the Workload Distribution affects both the execution time and the FIT. Through the MEBF shown in Figures 4.9, 4.10, 4.11, and 4.12 we can identify the configuration that delivers higher performances and lower error rate. A proper workload distribution can significantly increase the amount of data correctly produced by the device. For BS, CED, and BFS (Figures 4.9, 4.11, and 4.12 respectively), choosing to use GPU-alone (100% GPU) can improve the reliability of 3.8x, 90x, and 2.8x, respectively. For SC shown in Figure 4.10, the most reliable configuration is achieved when 70% of the workload is assigned to the CPU with a 1.4x higher MEBF concerning CPU-alone (100% CPU). A promising result we can gather from our analysis is that the Workload Distribution that guarantees shorter execution time is also the configuration with higher MEBF and, thus, the most reliable. This statement holds for all the tested codes but BS, for which best performances are delivered with the 90% GPU configuration, but the highest MEBF is achieved using the GPU-alone. BS is faster on GPUs than CPUs (refer to Table 4.2), but CPU can still complete 10% of the workload faster than the GPU completes 90% resulting in better execution time when compared to GPU-alone. Nevertheless, the difference in the execution time between 90% and 100% GPU is less than 10% (please refer to Table 4.2). The significant FIT increase due to the additional number of resources required for computation when both GPU and CPU are used is not compensated by the slightly reduced execution time. For all the other codes the MEBF is higher when the execution time is lower, regardless of the FIT rate.

For SC the Workload Distribution that delivers higher MEBF is not the one with the lowest FIT rate, which means that the benefit regarding reduced execution time brought by the additional resources used for computation is higher than the drawback of having a higher FIT rate. The Workload Distribution impacts positively, with gains up to 125%. The FIT rate decreases towards the GPU as explained in Section 4.2 and it would expect the MEBF follows the same course. Due to the inefficiency of the GPU core to perform memory manipulations, the configuration that presents the best MEBF is the one that beat others in performance.

5 CONCLUSIONS

Fault injection is one of the methodologies to investigate the reliability of devices and applications (MUKHERJEE et al., 2003b; SRIDHARAN; KAELI, 2009). By injecting faults through software frameworks it is possible to track faults propagation and evaluate the vulnerability of architectures and algorithms. However, as faults are injected with user-defined probabilities, fault injection does not provide any information about the natural probability of faults generation. As such, fault injection fails in evaluating the different sensitivities of cores of different kind and size.

Radiation beam experiments are the most realistic way to measure the error rate of a device. CHIPIR facilities allow performing experiments resembling accurately the natural conditions to which devices are exposed. However, beam experiments treat the device as a black box, i.e., errors can be observed only when they exhibit at the application output as an SDC or when they compromise the system responsiveness, as a DUE. Therefore, no information that helps to understand the fault propagation pattern is available. Relying only on beam experiments hinders the identification of the most critical and vulnerable software and hardware elements. Beam experiments and fault injection data complement each other and together delivers deep insights about devices reliability.

We have seen that the workload distribution impact not only the execution time and energy consumption of heterogeneous devices, as already demonstrated in previous works, but also their error rate. This thesis presents a reliability evaluation of Workload Distribution on heterogeneous devices through radiation experiments and software fault-injection. Regarding silent errors, we show that software fault-injection can correctly identify the most reliable cores for compute-bound codes, but radiation experiments are needed for memory-bound codes where memory operations or kernel synchronization dominate execution time using hardware resources inaccessible to fault injection. We also identify the Workload Distribution that achieves the highest reliability and show that in most cases it corresponds with the distribution that delivers the highest performances. While the higher amount of resources used to speed up computation increases the device faults sensitivity, the benefits regarding higher performances are higher than the drawback of having a higher error rate. Finally, DUE FIT rates depends on the collaboration pattern of Workload Distribution used, showing that several kernel invocations and synchronization among cores have an adverse impact.

REFERENCES

9TH DIMACS Implementation Challenge - Shortest Paths. [S.l.]: DIMACS, 2006. Urlwww.dis.uniroma1.it/challenge9/download.shtml.

BARUAH, T. et al. Airavat: Improving energy efficiency of heterogeneous applications. **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**, p. 731–736, 2018.

BAUMANN, R. Radiation-induced soft errors in advanced semiconductor technologies. **Device and Materials Reliability, IEEE Transactions on**, v. 5, n. 3, p. 305–316, Sept 2005. ISSN 1530-4388.

CAZZANIGA, C.; PLATT, S.; FROST, C. Preliminary results of chipir, a new atmospheric-like neutron beamline for the irradiation of microelectronics. **Proceedings of SELSE**, v. 13, 2013.

CHANG, L.-W. et al. Collaborative computing for heterogeneous integrated systems. In: ACM/SPEC. **International Conference on Performance Engineering (ICPE), 8th ACM/SPEC**. [S.l.], 2017.

CONG, J.; YUAN, B. Energy-efficient scheduling on heterogeneous multi-core architectures. In: ACM. **Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design**. [S.l.], 2012. p. 345–350.

DANALIS, A. et al. The scalable heterogeneous computing (shoc) benchmark suite. In: ACM. **Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units**. [S.l.], 2010. p. 63–74.

DEVICES, A. M. **CodeXL-Whitepaper**. 2018. <https://gpuopen.com/codexl-2-6-released/>. [Online; accessed 20-October-2018].

FRAGKIADAKI, K. et al. Two-granularity tracking: Mediating trajectory and detection graphs for tracking under occlusions. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2012. p. 552–565.

FRATIN, V. et al. Code-dependent and architecture-dependent reliability behaviors. In: **2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2018. p. 13–26.

FRATIN, V. et al. Energy-delay-fit product to compare processors and algorithm implementations. **Microelectronics Reliability**, v. 84, p. 112 – 120, 2018. ISSN 0026-2714. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S002627141830132X>.

GOMEZ, L. A. B. et al. GPGPUs: How to Combine High Computational Power with High Reliability. In: **2014 Design Automation and Test in Europe Conference and Exhibition**. Dresden, Germany: [s.n.], 2014.

GÓMEZ-LUNA, J. et al. Chai: Collaborative heterogeneous applications for integrated-architectures. In: IEEE. **Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on**. [S.l.], 2017.

HU, F.; QUAN, X.; LU, C. A schedule method for parallel applications on heterogeneous distributed systems with energy consumption constraint. In: ACM. **Proceedings of the 3rd International Conference on Multimedia Systems and Signal Processing**. [S.l.], 2018. p. 134–141.

JEDEC. **Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices**. [S.l.], 2006.

JEON, H. et al. Architectural vulnerability modeling and analysis of integrated graphics processors. In: **IEEE 10th Workshop on Silicon Errors in Logic - System Effects (SELSE)**. [S.l.: s.n.], 2013.

KLIAZOVICH, D.; BOUVRY, P.; KHAN, S. U. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. **The Journal of Supercomputing**, Springer, v. 62, n. 3, p. 1263–1283, 2012.

LI, K. Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 19, n. 11, p. 1484–1497, 2008.

LUCAS, R. Top ten exascale research challenges. In: **DOE ASCAC Subcommittee Report**. [S.l.: s.n.], 2014.

LUO, L.; WONG, M.; HWU, W.-m. An effective gpu implementation of breadth-first search. In: ACM. **Proceedings of the 47th design automation conference**. [S.l.], 2010. p. 52–55.

MUKHERJEE, S. S. et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: IEEE COMPUTER SOCIETY. **Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture**. [S.l.], 2003.

MUKHERJEE, S. S. et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: **Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture**. Washington, DC, USA: IEEE Computer Society, 2003. (MICRO 36), p. 29–. ISBN 0-7695-2043-X. Available from Internet: <http://dl.acm.org/citation.cfm?id=956417.956570>.

NVIDIA. **TegraX2-Whitepaper**. 2018. <https://www.nvidia.com/>. [Online; accessed 20-October-2018].

OLIVEIRA, D. et al. Carol-fi: An efficient fault-injection tool for vulnerability evaluation of modern hpc parallel accelerators. In: **Proceedings of the Computing Frontiers Conference**. New York, NY, USA: ACM, 2017. (CF'17), p. 295–298. ISBN 978-1-4503-4487-6.

OLIVEIRA, D. et al. Experimental and analytical study of xeon phi reliability. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. New York, NY, USA: ACM, 2017. (SC '17), p. 28:1–28:12. ISBN 978-1-4503-5114-0. Available from Internet: <http://doi.acm.org/10.1145/3126908.3126960>.

PILLA, L. L. et al. Software-based hardening strategies for neutron sensitive fft algorithms on gpus. **IEEE Transactions on Nuclear Science**, IEEE, v. 61, n. 4, p. 1874–1880, 2014.

PREVILON, F. G. et al. Evaluating the impact of execution parameters on program vulnerability in gpu applications. In: IEEE. **2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.], 2018. p. 809–814.

RECH, P. et al. Neutron-induced soft errors in graphic processing units. In: IEEE. **2012 IEEE Radiation Effects Data Workshop**. [S.l.], 2012. p. 1–6.

RECH, P. et al. Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability. In: **IEEE International Conference on Dependable Systems and Networks (DSN 2014)**. Atlanta, USA: [s.n.], 2014.

REIS, G. et al. Design and evaluation of hybrid fault-detection systems. In: **Proceedings of the 2005 International Symposium on Computer Architecture, ISCA'05**. [S.l.]: IEEE Press, 2005. p. 148–159.

RESEARCH. **APU Layout**. 2016. <https://pc.watch.impress.co.jp/video/pcw/docs/630/741/p04.pdf>.

RESEARCH. **CPU Diagram**. 2019. <https://www.computerscience.gcse.guru/theory/von-neumann-architecture>.

RESEARCH, A. S. C. **Scientific Discovery through Advanced Computing - The Challenges of Exascale**. 2016. <https://science.energy.gov/ascr/research/scidac/exascale-challenges/>. [Online; accessed 5-March-2016].

SNIR, M. et al. Addressing failures in exascale computing. **International Journal of High Performance Computing Applications**, SAGE Publications, p. 1094342014522573, 2014.

SRIDHARAN, V.; KAELI, D. R. Eliminating microarchitectural dependency from architectural vulnerability. In: **2009 IEEE 15th International Symposium on High Performance Computer Architecture**. [S.l.: s.n.], 2009. ISSN 1530-0897.

SUN, Y. et al. Hetero-mark, a benchmark suite for cpu-gpu collaborative computing. In: **2016 IEEE International Symposium on Workload Characterization (IISWC)**. [S.l.: s.n.], 2016. p. 1–10.

TIMOR, A. et al. Using underutilized cpu resources to enhance its reliability. **IEEE Transactions on Dependable and Secure Computing**, IEEE, v. 7, n. 1, p. 94–109, 2010.

TIWARI, D. et al. Understanding GPU Errors on Large-scale HPC Systems and the Implications for System Design and Operation. In: ACM. **Proceedings of 21st IEEE Symp. on High Performance Computer Architecture (HPCA)**. [S.l.], 2015.

VALLERO, A.; CARELLI, A.; CARLO, S. D. Trading-off reliability and performance in fpga-based reconfigurable heterogeneous systems. In: **2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)**. [S.l.: s.n.], 2018. p. 1–6.

VALLERO, A. et al. Multi-faceted microarchitecture level reliability characterization for nvidia and amd gpus. In: IEEE. **VLSI Test Symposium (VTS), 2018 IEEE 36th**. [S.l.], 2018. p. 1–6.

WANG, G.; LIU, S.; SUN, J. A dynamic partial reconfigurable system with combined task allocation method to improve the reliability of fpga. **Microelectronics reliability**, Elsevier Science, v. 83, n. 1, p. 14–24, 2018.

WIRTHLIN, M. et al. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In: IEEE. **11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.** [S.l.], 2003. p. 133–142.