Performances VS Reliability: how to exploit Approximate Computing for Safety-Critical applications

Gennaro Rodrigues, Fernanda Kastensmidt, Vincent Pouget, Alberto Bosio

To cite this version:

Gennaro Rodrigues, Fernanda Kastensmidt, Vincent Pouget, Alberto Bosio. Performances VS Reliability: how to exploit Approximate Computing for Safety-Critical applications. IOLTS: International On-Line Testing Symposium, Jul 2018, Platja d'Aro, Spain. pp.291-294, 10.1109/IOLTS.2018.8474122 . hal-02095642

HAL Id: hal-02095642

https://hal.archives-ouvertes.fr/hal-02095642

Submitted on 12 Apr 2019

Performance VS Reliability: How to Exploit Approximate Computing for Safety-Critical Applications

Gennaro S. Rodrigues,
Fernanda L. Kastensmidt
Instituto de Informatica, PGMicro
Universidade Federal do Rio Grande do Sul
{gsrodrigues, fglima}@inf.ufrgs.br

Vincent Pouget
IES
University of Montpellier
CNRS - France
vincent.pouget@ies.univ-montp2.fr

Alberto Bosio
LIRMM
University of Montpellier
CNRS - France
alberto.bosio@lirmm.fr

***Abstract*—Approximate Computing (AxC) paradigm aims at designing energy-efficient systems, saving computational resources, and presenting better execution times. AxC aims to selectively violate the specifications, trading accuracy off for efficiency. It has been demonstrated in the literature the effectiveness of imprecise computation for both software and hardware components implementing inexact algorithms, showing an inherent resiliency to errors. On the other hand, the hidden cost of AxC is the reduction on the inherent resiliency to errors of an application. This paper aims at analyzing the impact of AxC on the reliability.**

***Index Terms*—Fault Tolerance, Approximate Computing, Laser, Fault Injection.**

I. Introduction

Approximate Computing (AxC) techniques [1] execute inexact computations results with acceptable accuracy for many applications [2]. They have been used in many scenarios, from big data to scientific applications [3]. AxC has been proposed as an approach to developing energy-efficient systems [4], saving computational resources and presenting better execution times. Many AxC implementations have been presented so far in the literature.

At software-level, the approximation can be achieved by selectively ignoring specific computations and/or memory accesses while assuring the acceptable quality of results. For these approaches, one has to identify first the less-critical portions of the software, such as variables, function calls, loops and then approximate them by using strategies such as reducing the computation precision, skipping tasks, skipping memory accesses, or skipping some iterations of a loop [2].

At hardware-level, the approximation can be achieved by using approximate circuits, such as approximate adders, multipliers or other logical circuits, or approximate storage strategies, where data can be stored by truncating the lower-bits in the case of floating point data types. Another way to achieve approximation in hardware is to exploit a given hardware component outside its nominal operating conditions, thus generating approximate outputs during its execution.

The common point among the above-listed techniques is the reduction of the "intrinsic redundancy" of the application. Thus, the hidden cost of approximation techniques is the reduction of the inherent fault tolerance of the application itself [5].

This *cost* has never been quantified and taken into account as a metric by AxC techniques. However, it must be considered specifically when the approximated application runs on a safety-critical system.

Safety-critical systems often deal with human lives and high-cost equipment and therefore shall provide high dependability. Those systems are constantly subject to faults, given the dangerous environment they are subjected (e.g., radiation for aerospace systems). When a fault affects the system in a way that it is perceived by the user or other parts of the system, we say that an error occurs [6]. A soft error occurs when it does not permanently damage a system. A soft error is also called a single event upset (SEU). In some cases, such as when exposed to intense radiation environments, electronic systems are affected by multiple bits upset (MBU), but those cases are rare.

In this work, we would like to prove that this cost really exists and we also provide a quantification. This paper is organized as follows. Section II presents the methodology. Section III presents the experiments results and discussions. Finally, the conclusions are presented at Section IV.

II. Methodology

As stated in the previous section, many AxC techniques are available in the literature. In this work, we focus on the so-called successive approximation algorithms based on numerical methods. Numerical methods are algorithms essential for all branches of science and engineering. In a matter of fact, it is the only way to solve many computation problems, such as derivatives and integrals. They make use of mathematical properties and numerical analysis theory to approximate their results to the mathematically expected one.

Successive approximation algorithms are numerical calculations for which an exact, straightforward solution is not computationally achievable. Such is the case of the calculation of the integral of a function. Those algorithms are iteration-based and get closer to an acceptable result on every iteration. Because the value is approximated for each iteration, it is expected that if an error occurs, causing an iteration result that is out of the expected calculation path, it will be corrected in the following iterations. This behavior, however, is not assured. In the event of a permanent hardware fault, all the iterations of the algorithm might be compromised, and the algorithm would not converge. An error affecting one of the last iterations computations may also be not corrected, as it would be too late for the algorithm to converge back to an acceptable result.

A. Case-Study Application

The Newton-Raphson method is an algorithm used to find the roots of a function. It calculates the intersection of the tangent line of the function in an initial guess point x_0 with the x-axis. It is calculated iteratively, as stated in Equation 1 until it reaches a sufficient approximation.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{1}$$

In this work, the Newton-Raphson algorithm is used as application casestudy providing three variants. For each variant, the number of iterations is different (and therefore the accuracy), but the algorithm remains the same.

B. Experimental Environment

In order to evaluate the impact of the approximation on the application fault tolerance, we artificially inject faults by using a laser. The experiments setup is presented in Figure 1. It consists of the DUT, a host computer, and the laser equipment. The host computer is responsible for controlling the laser beam and listening to messages from the DUT. Details about the laser injection facility are out the scope of this paper but are deeply described in [5].

The device under test (DUT) of this work is a Zynq-7000 APSoC, designed by Xilinx. The Zynq board has embedded a high-performance ARM Cortex-A9 processor with two cache levels on the processing system (PS), alongside a programmable logic (PL) layer. The PL presents an FPGA based on the Xilinx 7-Series with approximately 27.7Mb configuration logic bits and 4.5Mb Block RAM (BRAM). The dual-core 32-bit ARM Cortex-A9 processor runs a maximum of 666MHz and is designed with 28nm technology. It counts with two L1 caches (data and instruction) per core with 32KB each, and one L2 cache with 512KB shared between both cores. A 256KB on-chip SRAM memory (OCM) is shared between the PS and PL parts, and so is the DDR (external memory interface). In this work, only the PS part of the board is used. The DUT periodically sends messages to the host computer, to report an error or to confirm it is alive.

Error messages are reported when there is a difference between an execution output and the golden output. The golden output is the result of a fault-free execution at the beginning of the experiment, called golden execution. The alive message is essential because some faults will cause the

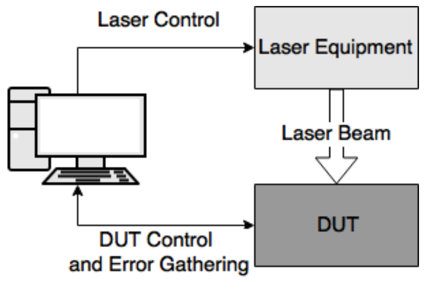

Fig. 1. Laser Experiments Setup

DUT to be irresponsible or hang (errors definitions will be further detailed in Section II-C), needing a reset. A reset consists of re-programming and configuring the DUT and is performed when a timeout occurs while the host computer waits for an alive message from the DUT. This timeout is set to about three minutes but may vary for different experiments with different response times. During a reset, the DUT warns the host computer so that the laser beam is deactivated. It prevents any errors during the system initialization and golden execution. The laser beam is then re-activated after the host computer receives an alive message from the DUT.

The communication between the host computer and the DUT is rather complex and is highly susceptible to errors because it happens during the fault injection. To avoid errors that are not interesting to our experiment and would make it less efficient, we developed a strategy to reduce this communication to a minimum necessary. During an application execution, the algorithm runs N times, filling in an output vector, which will be then compared with the result from the fault-free execution (golden value). This way the DUT only has to send messages to the host computer once per execution, every N runs (and when an error is detected). The value of N may vary for different benchmarks, according to their execution times.

Table I provides some details about the case-study application. The number of runs is the size of the output vector, i.e., the value of N times an algorithm runs per execution. "Workload per Run" represents the size of the output values, while the "Total Workload per Run" is the size in bytes of the output vector. The "Execution Time" is the time of a complete execution (N runs). Finally, the last column presents the average number of laser shots per execution which is calculated dividing the execution time and the time between laser shots (i.e., the inverse value of the laser frequency).

C. Errors

After each application execution, the output vector is compared with the golden value to check for its correctness. When the output value and the golden value are different, the DUT sends to the host computer a message containing the details of the error (position on the output vector and the value of the incorrect output). The host computer receives the error messages and saves them into a log to be further analyzed.

TABLE I
BENCHMARKS DETAILS

Application	Variant	Number of Iterations	Number of Runs (N)	Workload per Run [Bytes]	Total Workload [Bytes]	Execution Time [ms]	Avg. Number of Shots per Execution
Newton-Raphson	1	14	100	8	800	41.72	0.417
	2	37	100	8	800	130.20	1.302
	3	71	100	8	800	337.91	3.3791

TABLE II
ERROR TYPE CLASSIFICATIONS

Type	Error
HANG	Causes the application to be stuck in a certain point.
SDC	Output difference between the golden execution and the exposed one.

The errors are classified into two types: HANG and SDC. They are defined at Table II.

III. EXPERIMENTAL RESULTS

The fault tolerance of the application is evaluated in two aspects. First, we assess how the number of iterations impacts the error susceptibility, i.e., how each variant presented at Table I at Section II-A behaves under fault injection. Variating the number of iterations has a significant impact on fault tolerance. Figures 2 presents the error relative probability (per laser pulse) for each variant. This probability is calculated by normalizing the error occurrence values with their maximum for each variant. The normalization is needed because the error occurrence depends on the execution time and the shots per execution of the application, and those are very different for each variant.

As expected, the variants with larger number of iterations are more fault-tolerant. However, more iterations means more latency. As Table I shows, the variant 3 is almost $10x$ times slower, but Figure 2 shows the error occurrence does not decrease in the same pace. It means that, for this algorithm, increasing the number of iterations improves reliability, but the price is high. More in detail, Figure 2 indicates that the variant 2 already achieves a considerable fault tolerance improvement, having a relative probability two orders of magnitude smaller than the first variant.

The second type of validation is done by relaxing the definition of SDC. Instead to consider all the differences between the golden and the faulty output, we classify the SDC in *critical* and *non-critical*. The classification depends on how much the golden and faulty output differs. If the difference is lower than a certain threshold, the SDC is classified as non-critical.

Figure 3 presents the critical SDC drop when variating the output threshold for the Newton-Raphson algorithm. The variant 3 of Newton-Raphson is the one that had the more erroneous values. The error drop stagnates after around 4% of output variation tolerance.

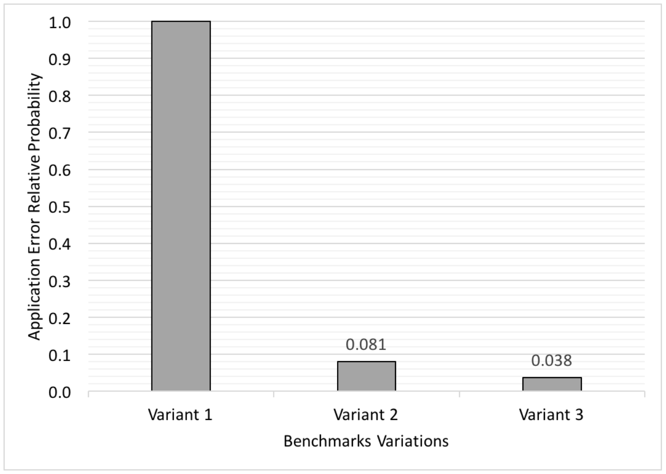

Fig. 2. Newton-Raphson Application Error Relative Probability (per laser pulse)

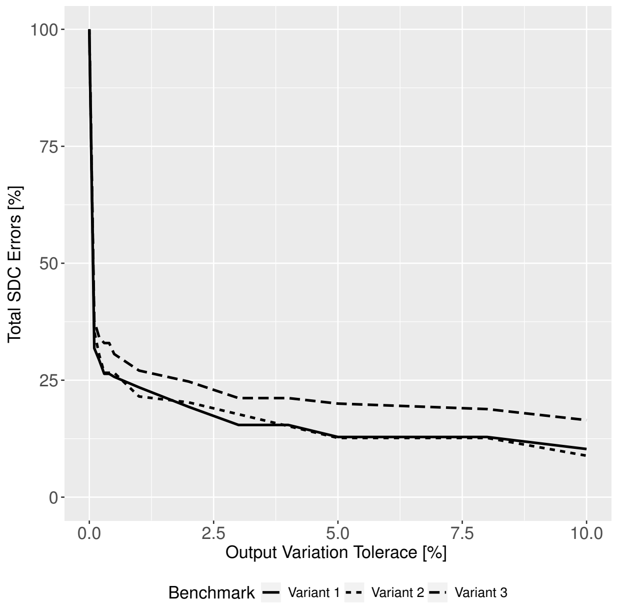

Fig. 3. Error Occurrence Drop in Relation to Output Variation Tolerance for Newton-Raphson Benchmark

IV. CONCLUSION

Approximate Computing as an hidden cost that has to be considered especially for safety-critical systems. Therefore, the study of this kind of approximate computing algorithms is essential before it can be applied to safety-critical systems as reliable software. However, the number of iterations does affect the fault tolerance. Even for an application with a relatively low number of iterations, such as Newton-Raphson, its impact on fault tolerance is noticeable.

All the variants showed a trend of having a significant drop in the number o SDC errors for small output variation tolerances. It shows that most of the SDC type errors affecting approximate computing by successive approximation algorithms have values not very different from the expected one. In other words, the errors are not significant. Many applications that use this kind of algorithm may tolerate small variations on the output without a problem. For those applications, successive approximation arises as the perfect method for approximate computing.

On future works, we will investigate the impact of other Approximate Computing techniques (e.g., precision reduction) on the fault tolerance.

References

[1] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 120:1–120:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2751163

[2] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[3] R. Nair, "Big data needs approximate computing: Technical perspective," *Commun. ACM*, vol. 58, no. 1, pp. 104–104, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2688072

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[5] G. S. Rodrigues and F. L. Kastensmidt, "Evaluating the behavior of successive approximation algorithms under soft errors," in *2017 18th IEEE Latin American Test Symposium (LATS)*, March 2017, pp. 1–6.

[6] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, Sept 2005.