Radiation-Induced Error Criticality in Modern HPC Parallel Accelerators

*Daniel Oliveira, †Laercio Pilla, ‡Mauricio Hanzich, *Vinicius Fratin,*Fernando Fernandes, *Caio Lunardi,
‡José María Cela, *Philippe Navaux, *Luigi Carro, *Paolo Rech
*Institute of Informatics †Department of Informatics and Statistics ‡CASE Department
UFRGS UFSC Barcelona Supercomputing Center
Porto Alegre, Brazil Florianópolis, Brazil Barcelona, Spain

***Abstract*—In this paper, we evaluate the error criticality of radiation-induced errors on modern High-Performance Computing (HPC) accelerators (Intel Xeon Phi and NVIDIA K40) through a dedicated set of metrics. We show that, as long as imprecise computing is concerned, the simple mismatch detection is not sufficient to evaluate and compare the radiation sensitivity of HPC devices and algorithms. Our analysis quantifies and qualifies radiation effects on applications' output correlating the number of corrupted elements with their spatial locality. Also, we provide the mean relative error (dataset-wise) to evaluate radiation-induced error magnitude.**

We apply the selected metrics to experimental results obtained in various radiation test campaigns for a total of more than 400 hours of beam time per device. The amount of data we gathered allows us to evaluate the error criticality of a representative set of algorithms from HPC suites. Additionally, based on the characteristics of the tested algorithms, we draw generic reliability conclusions for broader classes of codes. We show that arithmetic operations are less critical for the K40, while Xeon Phi is more reliable when executing particles interactions solved through Finite Difference Methods. Finally, iterative stencil operations seem the most reliable on both architectures.

I. Introduction

Today, reliability is one of the major concerns for HPC systems. Due to the large-scale and runs duration, leadership scientific applications may encounter interruptions as a result of application crashes or system hangs as well as Silent Data Corruption (SDC) in the output. As a reference, the U.S. Department of Energy's (DOE) Titan, today's third most powerful supercomputer [15], which is composed of more than $18,000$ $Kepler$ GPUs, has a radiation-induced Mean Time Between Failures (MTBF) in the order of dozens of hours [18], [41]. As we approach exascale, the resilience challenge will become even more critical due to an increase in system scale [24], [36].

The high computational power of modern accelerators combined with their low cost, reduced energy consumption, and flexible development platforms are pushing their adoption in HPC applications. In this paper we analyze the radiation reliability of Intel *Xeon Phis* and NVIDIA *Kepler* GPUs, which dominate the HPC market. Tianhe-2, today's second powerful supercomputer [15], and Trinity, Los Alamos National Laboratory's (LANL) new cluster, are powered by Xeon Phis, while NVIDIA Kepler GPUs act as accelerators in 2 of the top 10 supercomputers, including Titan. In this scenario, a lack of understanding of HPC devices resilience may lead to lower scientific productivity, lower operational efficiency, and even significant monetary loss [36].

SDCs are typically studied by comparing the code experimental output with the expected output [6], [26]. We aim at going a step beyond and consider *error criticality*, i.e., how SDCs impact the application or system. On parallel architectures, a soft-error can either corrupt only a thread/data value, corrupting the output in an insignificant manner, or propagate to several threads/data values, leading to a significant corruption in the output. Moreover, the effect of a radiation-corrupted output strictly depends on the application and may be, under certain circumstances, tolerated. For instance, errors affecting the least significant positions of the mantissa could be considered inside the intrinsic imprecision of floating-point operations. Additionally, wave simulations may accept misfits of about 4% [14] and imprecise computation can be generically applied to various HPC applications [10]. In other words, HPC output errors should not be equally considered, but their criticality should be taken into account.

To have a qualitative and quantitative evaluation of error criticality we propose the use of four metrics: (1) number of errors, (2) relative error, (3) mean relative error and (4) spatial locality. The number of errors indicates how many output elements differ from the expected values. Relative and mean relative errors are used to filter those elements whose values have a relative difference from the expected ones which is lower than a parametrized threshold. Spatial locality is used to describe the SDCs distribution in the output. Our criticality study discloses interesting peculiarities of Xeon Phi and Kepler architectures. Thanks to the proposed methodology we identify which architecture is more likely to produce critical errors in the tested codes and which parallelism management philosophy is more reliable.

We benefit from the accelerated high energy neutron beams available at Los Alamos Neutron Science Center (LANSCE) at LANL, Los Alamos, NM, USA and at ISIS, Rutherford Appleton Laboratories (RAL), Didcot, UK to experimentally evaluate the neutron-induced errors impact on Xeon Phi and Kepler devices. By inducing failures in all the components of the device, including the scheduler, dispatcher, and control logic, our neutron beam experiments provide deep insights on the resilience characteristics of HPC accelerators that are, otherwise, difficult to obtain.

We select a representative and heterogeneous set of codes to evaluate radiation effects in modern HPC accelerators: DGEMM (arithmetic operations), LavaMD (particles interactions), HotSpot (physical simulations), and a DOE's proprietary workload named CLAMR (fluid dynamics). We report data obtained in various test campaigns for an overall beam time of more than 400 hours per device, resulting in a wide set of possible radiation-induced failures. For DGEMM and LavaMD we test also different input sizes to evaluate how the increased number of parallel processes, workload, and throughput impact the accelerators reliability. The selected codes are representative of wider classes of HPC algorithms, which allows our error criticality to be generalized. Whenever possible, we correlate the observed error criticality with algorithm characteristics and compute or memory requirements.

The main contributions of this paper are: (1) a novel error criticality evaluation methodology based on specific metrics, (2) the reliability analysis of HPC applications based on beam experiments, whose logs are made publicly available in [1] to ease reproducibility and third party analysis, (3) a broad discussion on the error criticality of wide classes of HPC algorithms, and (4) the comparison between errors criticality in Xeon Phis and K40s.

The remainder of the paper is organized as follow. Section II gives a brief background on the impact of radiation in HPC. Section III details the metrics used throughout this paper. Section IV describes the adopted methodology. Section V presents the evaluation based on the proposed metrics for errors criticality in modern HPC accelerators. Finally, Section VI concludes the paper and proposes future work.

II. Background

A. Radiation-Induced Effects

The impact of galactic cosmic rays with the upper level of the terrestrial atmosphere generates a great number of high energy neutrons (i.e., neutrons with an energy higher than 10MeV).A flux of about 13 $n/(cm^2 \times h)$ reaches the ground, and the number of neutrons increases exponentially with altitude [23]. The interaction of a neutron may perturb the transistor state, generating (single or multiple) bit-flips in memory as well as a current spike in logic circuits that, if latched, leads to an error. The impact of radiation-induced errors in logic is expected to increase in future technology, eventually becoming more probable than errors in memory elements [11], [25].

A radiation strike in modern HPC accelerators leads to one of the following outcomes: (1) no effect on the program output (the failure is masked, or corrupted data is not used), (2) Silent Data Corruption (incorrect program output), (3) application Crash, (4) system Hang (the node has to be rebooted to restore its functionality). Among these outcomes, (2) is harmful as it remains undetected and unpredictable while (3) and (4) lead to performance penalties and eventual data loss if a checkpoint was not performed. In this paper we concentrate on SDCs, as Crash or Hangs, for their nature, are at least detectable. Specifically, we will analyze the impact of incorrect program output in the correctness of the simulation or code solution.

B. Output Correctness and Error Criticality in HPC

The impact of radiation corruption in the output may vary depending on the architecture and application class. One impinging particle may alter single or multiple bits in one or multiple words. However, depending on how the device or algorithm digests the corrupted data, the outcome can vary widely. Scientific applications with filter phases, like stencils, may mask or lower the magnitude of errors, but still spread them among several elements in the multiple dimensions of the output due to repetitive operations with the corrupted data. If the scheduler or a hardware routine are corrupted, the outcome could range from the crash of a device to several improperly scheduled threads producing incorrect data.

A corrupted output is not always critical in HPC applications. Floating-point operations' results, for instance, have an intrinsic variance [44]. A radiation-induced error that affects the least significant bits of the mantissa of a float output element could then have little to no effect on the application correctness. Additionally, physical simulations accept them as correct values in a given range [14], [19]. If the value of the corrupted output element is inside the accepted margin we may still consider it as correct. Lately, various works have discussed the preference of not pursuing strict output correctness in HPC to improve performance and efficiency [13], [10], [16].

In this paper, we aim at applying the concept of inexactness to radiation-induced errors. In fact, some corruption could still be tolerable if we consider imprecise computations on HPC systems. Understanding how off this incorrect output could be from the correct one, and how errors may affect the output on different architectures is essential to precisely evaluate the effect of radiation in current and future HPC machines.

III. Metrics

We select four metrics to characterize radiation-induced output errors and to discuss their criticality in HPC applications: the number of incorrect elements, relative error, mean relative error, and spatial locality.

As will be detailed in Section IV-D, we design our experiments to have at most one neutron generating a failure per execution. When multiple elements in the output are corrupted it means that the effect of that single impinging neutron propagates and spreads affecting multiple processes or values. The higher the **number of incorrect elements** in the output data, the more likely for a code to propagate the error and exacerbate the number of incorrect elements in the output. The number of incorrect elements, then, correlates well with the algorithm and architecture sensitivities.

The number of incorrect elements is especially significant for parallel architectures. Accelerators like Intel Xeon Phi and NVIDIA GPUs have dozens or thousands of cores that share different levels of resources. If a shared resource is corrupted, several threads may produce incorrect data. Moreover, each device handles parallelism differently. NVIDIA has simple cores and a hardware scheduler [31], [30] while Intel uses more complex cores and a complete operating system with software scheduler [21], [22]. The corruption of the scheduler or operating system is likely to affect the execution of multiple threads.

To measure output error magnitude we calculate the **relative error** which is given by the following equation:

$$relative\,error = \frac{|read - expected|}{|expected|} \times 100.$$

Where *read* is the value of the corrupted element and *expected* is the correct one. Relative error is a measure of how off the corrupted result is from what is expected, expressed in percentage. The relative error of a corrupted element that has a value which is ten times the expected will be 900%. If an algorithm produces text as output, one could apply **relative error** treating the output as integer.

The **mean of relative errors** is obtained averaging the relative errors of all the corrupted elements in the output. The mean of relative errors gives an overview of how much the overall corrupted output differs from the expected one. In our analysis we correlate the mean of relative errors with the number of incorrect elements. This correlation highlights how many elements were corrupted and how much those elements differ from the expected value. We can then distinguish situations in which radiation produces few corrupted elements that are significantly different from the expected value, and situations in which the corruption affects a lot of elements which are only slightly different from the expected value.

We also use the relative error to filter those errors that significantly impact the results and those errors that could be ignored. As discussed in Section II-B, some applications may accept as correct results that slightly wander off the precise value. For instance, a seismic wave application accepts misfits of about 4% [14]. Additionally, the relative error becomes fundamental for imprecise computations [2], [16], [10]. In this work, being conservative, we chose to consider only mismatches with relative errors greater than 2%. We are aware that the accuracy or relative error allowed by a scientific application or imprecise computations may vary widely. Hence, we made available all our corrupted outputs in a publicly accessible repository [1] so to allow users to apply different filters.

When we apply the filter, we ignore all incorrect elements whose relative error is lower than 2%. We remove faulty executions where there are no mismatches left after the filter. As will be shown in Section V, several errors could have a relative error inferior to some parameterized threshold. Considering every mismatch as an error would be, then, an ineffective evaluation of resilience and error criticality. Therefore, the reliability of architectures and algorithms that produce low relative errors could be immensely far from the real one.

It is common for HPC output data to be structured as two or three-dimensional arrays. The **spatial locality** of errors, then, identifies the output errors pattern. When several elements are corrupted, but they do not share the same position in one of the axis, they are tagged as random errors. When the corrupted elements share one, two, or three dimensions of the axis we classify them as line, square, or cubic respectively. The spatial locality of errors is important to understand error propagation in the considered architecture and how data is actually used in the device. Locality information can be fruitfully used to evaluate software-based hardening strategies detection efficacy. For example, the Algorithm-Based Fault Tolerance (ABFT) *DGEMM* can detect and correct single and line errors [20], [33] but not square errors. Therefore, by knowing the spatial locality we can evaluate if it is wise to implement ABFT. It is worth noting that the spatial locality can be deeply affected by the relative error, as the number of incorrect elements can decrease as we apply the filter.

The four presented metrics can be used in conjunction to better understand the reliability of an algorithm or architecture and conceive a solution to improve their resilience. The number of incorrect elements in the output can indicate the magnitude of error propagation. Correlating the number of incorrect elements with the mean relative error provides an overview of output correctness. Locality can give insights on errors propagation and help to understand data placement or organization. Locality could also contribute to the development and use of detection and correction strategies [8].

IV. Methodology

In this section, after introducing the tested devices, we present the selected codes and the reason for the chosen input sizes. Finally, we describe the experimental procedure adopted for our studies.

A. Tested Devices

The K40 board includes a Kepler architecture based on the GK110b GPU chip [29]. The GK110b is fabricated using 28nm planar bulk technology from TSMC and includes 15 Streaming Multiprocessors (SMs), up to 2048 threads/SM, 30 Mbit total register file (RF), 960 KB total L1 cache/Shared memory (64 KB per SM), 1536 KB L2 cache, and 12 GB GDDR5 (which is not irradiated).

The Xeon Phi board, codenamed Knights Corner, is the coprocessor 3120A [21], [22]. The coprocessor is fabricated using 22nm with the Intel 3-D Trigate transistors. The chip includes 57 physical in-order cores with four hardware threads and 32 512-wide vector registers per core. The board has 6GB GDDR5 (which is not irradiated) with 64 KB L1 cache and 512 KB private L2 cache for each core (a total of 3648 KB and 29184 KB for L1 and L2 caches, respectively). L2 caches are fully coherent and connected using a bidirectional 64 bytes wide data ring.

The physical implementations of Intel and NVIDIA devices are extremely different. 3-D transistors have shown a $10\times$ reduced per bit sensitivity to neutron compared to planar devices [28]. The raw resources corruption probability for the Xeon Phi is then expected to be lower then for the K40. Unfortunately, as circuit level details are proprietary, it is not possible to evaluate the devices low-level resources sensitivity. A direct comparison between NVIDIA and Intel devices physical implementation reliability is then unfeasible and out of the scope of this paper. We focus on the criticality of radiation-induced error, which depends on how the errors propagate till the application output and is related to the device architecture.

NVIDIA's and Intel's management of parallel processes are extremely different and may impact both the device efficiency and reliability. NVIDIA has a hardware scheduler while Intel relies on a dedicated Operating System (OS) to orchestrate execution. The characterization of the parallel threads management is part of the goal of our test procedure (details in Section IV-C).

B. Selected Algorithms

Several benchmark suites are available for performance and efficiency evaluation of computer architectures [5], [46], [9], [12]. A standard set of benchmarks for the reliability evaluation of HPC devices has not been establish, yet. General guidelines for reliability evaluation of computing devices suggest to consider codes from different domains and comprising different computation and communication patterns [3], [32]. Hence, we choose: a **Matrix Multiplication (DGEMM)** benchmark, **LavaMD**, **HotSpot** (mini-apps from the Rodinia benchmark suite) [12], and **CLAMR** (a DOE mini-app) [19]. The benchmarks are selected also as they are representative of different application classes: algebraic applications, particles simulation, physics simulation, and fluid dynamics. Additionally, as detailed in the following, each code is likely to stimulate specific resources on the Xeon Phi and K40. Hence, we believe that results obtained with the selected benchmarks could be, under certain premises, generalized to similar applications. It is worth noting that radiation experimental evaluation should be restrained to few benchmarks because of beam time limitations and the need to gather a statistically significant amount of data.

To broaden the representativeness of the selected applications, we have classified each code using some general parameters such as: resources bounding the execution (i.e. either CPU or memory), load balance (balanced or imbalanced), and the regularity of the memory access pattern – which affects the capacity of the algorithm to profit from the memory hierarchy (e.g., coalesced accesses). Table I shows the classification for the selected applications.

DGEMM is a Dense Linear Algebra [3] code that implements an optimized Matrix Multiplication, which serves as a cornerstone code for several applications and performance evaluation tools. Memory accesses are coalesced or vectorized, which results in a better memory locality and a high device utilization, but also stresses the register file, local memory, and Floating Point Unit (FPU). *DGEMM* is selected as representative of highly arithmetic codes performing compute-bound tasks with $O(N^3)$ in compute and $O(N^2)$ in space. Also, *DGEMM* is representative of those applications with *static* partitioning of the data among the computational resources, working on a data structure with a *regular* access pattern. We believe the error criticality of *DGEMM* to be of great importance for the HPC community as several HPC applications employ *DGEMM*, including *Linpack*, which is used to rank supercomputers [15].

LavaMD is a solver that uses Finite Difference Methods (FDM) to calculate particle potential and relocation due to mutual forces between particles within a large 3D space, behaving as an N-Body Method [3]. This space is divided into large boxes, that are allocated to individual blocks of threads [12]. The main computation in *LavaMD* lies on dot products with floating-point data, where each thread computes the interaction of one particle with all particles in neighboring boxes (26 neighbor boxes in the cutoff radius plus the home box allocated to the block of threads). As the home box and a neighbor box are kept at all times in local memory, *LavaMD* stresses local memory the most. *LavaMD* is selected as a benchmark representative of *Multi-physics Particle Dynamics Code (ddcMD)* applications [39], [38]. Also, as the application calculates the interactions with the particles in neighboring boxes, boxes in the borders of the simulated space will have less neighbors to compute, generating some load imbalance. Finally, *LavaMD* has a regular access pattern for calculating the interaction of particles inside a single box that produces coalesced accesses to memory.

HotSpot is a physics simulation code that simulates the energy dissipation on an architectural floor plan to estimate processor temperature [12]. At each iteration, *HotSpot* computes the average temperature on areas of the chip based on their previous temperature and power input. The code is a 2D stencil computing on single precision floating-point values. Given its small local memory footprint, a high number of iterations using local memory and registers only, and use of single-precision instead of double-precision, *HotSpot* achieves the highest occupancy among tested codes. We used *HotSpot* as a representative Structured Grid code [3], a pattern present in several applications used in HPC systems that solves Partial Differential Equations, like in geophysics [14], and deeply studied by the community [27]. The problem being solved by *HotSpot* is representative of a more general class of problems, where the *operational intensity*[1] is low enough to make the problem memory bound. Also, *HotSpot* does not suffer from any kind of load imbalance, as the calculation remains the same along its domain, which also favors a regular memory access pattern.

TABLE I: Classification of parallel kernels.

	Bound by	Load Balance	Memory Access
DGEMM	CPU	Balanced	Regular
LavaMD	Memory	Imbalanced	Regular
HotSpot	Memory	Balanced	Regular
CLAMR	CPU	Imbalanced	Irregular

CLAMR is a DOE homemade fluid dynamics application, representative of classified LANL supercomputers workloads. *CLAMR* simulates the long range propagation of waves using a cell-based adaptive mesh refinement implementation [19]. By using the shallow water equations (conservation of mass, x momentum, and y momentum) and by assuming that the fluid bottom is flat and that the flow in the vertical direction is negligible, the simulation is implemented by having each cell of the 2D space computed by a thread. *CLAMR* stresses FPU resources (by being compute-bound and working over double-precision floating-point data), control flow resources (the kernel uses several tests to handle questions like border conditions), and device control resources due to its large number of kernel calls and changes in number of threads between time steps to re-balance the load among computational resources.

It is worth noting that even if the high level code of the selected algorithms is the same for both devices, the post compiler code may be very different between the K40 and the Xeon Phi. This is due to different architectures and compilers. Nevertheless, as highlighted earlier in the section, the selected set of codes is heterogeneous in the sense that each stimulates a particular kind of resources the most. To reach the solution, both Xeon Phi and K40 devices are forced to use those resources.

C. Selected Input Sizes

To have a proper reliability evaluation, it is essential to fully utilize the device resources. An underused device can give different error criticalities due to smaller resource usage and fewer threads created. Input sizes were tailored to achieve high resource utilization (e.g., over 97.5% multiprocessor activity on the K40). This includes register files, cache memories, buses, ALUs, FPUs, control resources, and others. Table II resumes the input size and number of threads generated for each kernel and the selected configuration to achieve high resource usage. *DGEMM* input sizes (cell per matrix side) were varied between $2^{10} \times 2^{10}$ and $2^{13} \times 2^{13}$ in powers of two. *LavaMD*'s number of cubes in each dimension of a 3D grid was set to 13, 15, 19, and 23 (each cube contains 100 particles on Xeon Phi and 192 particles on K40. The number of particles was selected to best fit the hardware).

As tested input sizes are sufficient to saturate most of the resources on both devices, a bigger input size does not

[1]ratio between floating point operations and bytes brought from memory [45]

TABLE II: Parallel kernels' details.

	Domain	Input size	#Threads
DGEMM	Linear algebra	square matrix side ($2^{10} - 2^{13}$)	side$^2/16$
LavaMD	Molecular dynamics	grid size $(13, 15, 29, 23)$	grid size$^3\times$ #particles (100 on Xeon Phi, 192 on K40)
HotSpot	Physics simulation	#cells (1024×1024)	#cells
CLAMR	Fluid dynamics	#cells (512×512)	#cells or more (AMR)

increase the amount of resources required for computation and should not affect FIT [7]. However, increasing the input size increases the number of instantiated parallel processes, and modifies the shared resources distributions. Moreover, for most HPC applications the throughput is strongly dependent on the input size. Evaluating how error criticality changes with input size provides novel insights on parallel processes management reliability.

HotSpot's 2D stencil includes 1024×1024 cells. The workload employed in *CLAMR* is the standard test problem of a circular dam break. The mesh starts with 512×512 cells and simulates $5,000$ timesteps.

D. Neutron Beam Test Experimental Setup

Fault injection simulations are one of the possible ways to evaluate applications reliability by measuring their Architectural Vulnerability Factor (AVF) (i.e., the probability for a failure in a resource to be observed at the output [26]) or Program Vulnerability Factor (PVF) [37]. Some recent works evaluate the AVF for various parallel algorithms executed on GPUs [17], [40], [43]. Fault injectors provide the user with access to only a limited set of GPU resources. Thus, not all the possible sources of errors can be considered. Hardware schedulers and dispatchers as well as the PCIe controller, for instance, are among the inaccessible resources. Due to the limitations of fault injection, we take advantage of the controlled neutron beam to perform the error criticality analysis.

Experiments were performed at the LANSCE facility, Los Alamos, NM, and at the ISIS facility, RAL, Didcot, UK. LANSCE and ISIS fluxes are suitable to mimic the terrestrial neutron flux effects on electronic devices [42]. This means that error rates measured at LANSCE or ISIS scaled down to the natural flux provide the predicted error rates on a realistic application expressed in Failure In Time (FIT). The neutron flux available at LANSCE or ISIS was between $1 \times 10^5 n/(cm^2 \times s)$ or $2.5 \times 10^6 n/(cm^2 \times s)$, about 6 to 8 orders of magnitude higher than the atmospheric neutron flux at sea level ($13n/(cm^2 \times h)$ [23]). Tests were conducted for more than 400 hours of beam time. As we test multiple boards in parallel (two Xeon Phis and two K40), we report here results obtained in 800 hours of effective test per architecture. If scaled to the natural environment, our results cover at least 8×10^8 hours of normal operations, which are about 91,000 years.

In a realistic environment, it is highly likely to have no more than one corruption during a single execution. To maintain this behavior, experiments were tuned to guarantee observed output error rates lower than 10^{-3} errors/execution, ensuring that the probability of more than one neutron generating a failure in a single code execution remains negligible.

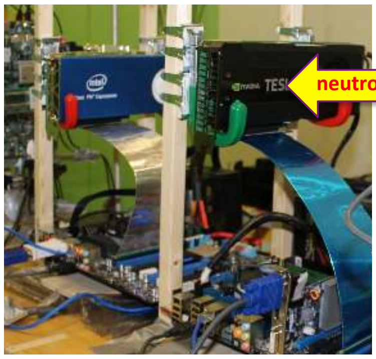


Fig. 1: Part of the experimental setup at LANSCE. Neutrons direction is indicated by the arrow.

The beam was restricted to a spot with a diameter of 2 inches, which was enough to fully irradiate the K40 and Xeon Phi chips without directly affecting nearby board power control circuitry and DRAM chips. This implies that data stored in the main memory is not to be corrupted, allowing an analysis focused on the devices' core reliability.

Figure 1 shows part of the experimental setup mounted at LANSCE. We irradiate a total of 2 Xeon Phis and 2 K40s, placed at different distances from the neutron source. A derating factor was applied to consider distance attenuation. After the de-rating the device radiation sensitivity seemed independent on the position, suggesting that the neutron attenuation caused by other boards between the source and the device under test is negligible.

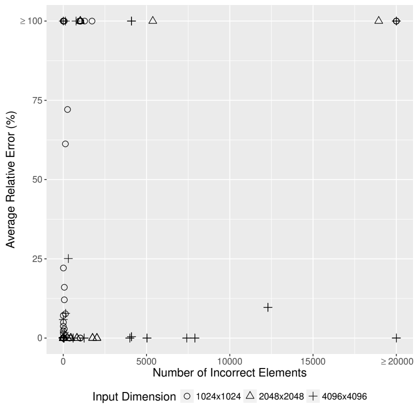
A host computer initializes the test sending pre-selected input to the accelerator and gathers results, comparing them with a pre-computed golden output. When a mismatch is detected, the execution is marked as affected by an error. To avoid precision and round-off issues, golden outputs were calculated on the very same device used for experiments. Input values were ensured to be small enough to avoid overflow but still big enough to be considered representative. Additionally, to avoid biases on input values, small input sizes are a subset of big input sizes and input has been generated balancing the number of 0s and 1s.

V. Reliability and Criticality Evaluation

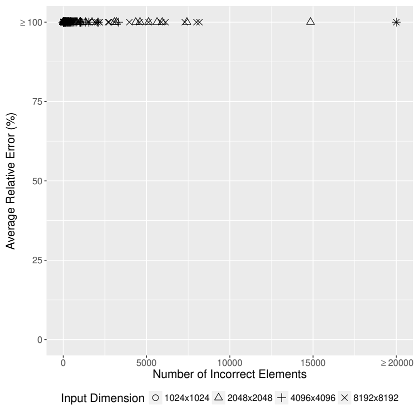
This section evaluates error criticality of HPC application classes. The analysis is based on the metrics proposed in Section III using the codes and methodology presented in Section IV. While this paper focuses only on SDCs, with our methodology it is also possible to measure radiation-induced crash and hang rates. As a reference, we measured that SDCs are between 1.1 to tens of times more likely than crashes and hangs for both the K40 and Xeon Phi. For *DGEMM*, K40 experienced between $1.1\times$ to $4\times$ more SDCs than crashes and hangs (the larger the input, the higher the crashes and hangs rate), while for Xeon Phi SDCs are about $4\times$ more likely than crashes and hangs (independently on the input). For *LavaMD*, K40 has about $3\times$ and Xeon Phi from $3\times$ to $12\times$ (increasing with input size) more SDCs than crashes and hangs. For *HotSpot*, K40 has $7\times$ and Xeon Phi has $3\times$ more SDCs. Observed differences may be dependent on algorithm control-flow characteristics, control logic sensitivity, instruction cache proprieties, or architecture peculiarities. We consider crashes and hangs less critical than SDCs as, for their nature, they are detectable. A detailed analysis of crashes and hangs causes and effects is then out of the scope of this paper. In the following, we consider only SDCs obtained during our radiation experiments.

Results are presented as relative FIT, expressed in arbitrary units ($a.u.$). Absolute FIT are considered business-sensitive data and are not included in this paper to protect our industrial partners. Nevertheless, as we use the same normalization for each device and code, relative FIT data still allows cross comparisons between codes and devices. As stated in Section IV-A, Xeon Phi and K40 have extremely different architectures built with different transistor layouts. The scope of this paper is not to exhaustively compare the error rate of the two devices, but rather to evaluate and compare the corrupted output criticality for different classes of algorithms with different input sizes executed in different HPC accelerators.

A. *DGEMM*



(a) K40.



(b) Xeon Phi.

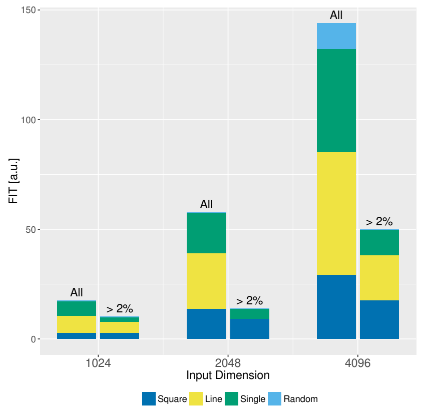
Fig. 2: *DGEMM* Mean relative error and Incorrect Elements.

Figures 2a and 2b show the mean relative error correlated with the number of corrupted elements for the faulty executions of *DGEMM* executed with 3 and 4 input sizes on K40 and Xeon Phi, respectively. It is worth noting that, to improve figure quality, for *DGEMM* we assign a 100% relative error to all those errors with a relative error higher or equal to 100%.

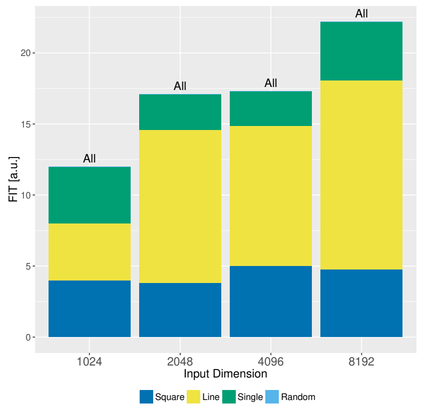
Most executions had a small number of incorrect elements in both architectures (at most 0.4% of output elements corrupted). The number of incorrect elements grows together with input size. We recall that, as described in Section IV-D, observed (multiple) corrupted elements are caused by a single impinging particle. When multiple corrupted elements affect the output it means that the initial corruption propagates disturbing the calculation of more than one element. An increase of *DGEMM* input size requires a higher number of parallel processes and a higher amount of shared resources (like caches). A corruption in either one is likely to cause multiple corrupted elements.

As shown in Fig. 2b, the mean relative error is extremely high on the Xeon Phi. Almost all the corrupted elements are extremely different from the expected value, independent of the number of corrupted elements or input dimension.

For the K40, about 75% of radiation-induced output errors have a lower than 10% mean relative error. The K40 has overall fewer corrupted elements and those elements' values are less different to the expected ones than on the Xeon Phi, indicating that *DGEMM* errors are then to be considered less critical on the K40 than on the Xeon Phi.



(a) K40.



(b) Xeon Phi.

Fig. 3: *DGEMM* spatial locality and magnitude.

Figures 3a and 3b present the spatial locality and relative errors for *DGEMM* executed on K40 and Xeon Phi. For each input size, we show the relative FIT break down into the different error patterns detected with our spatial locality analysis. For each dimension we report two FIT break downs, one considering all the corrupted executions and one applying the 2% relative error filter (All and $> 2\%$, respectively, in Figure 3a). For the $> 2\%$ break down, we do not consider as corrupted those output elements with a relative error lower than 2%. As for the Xeon Phi no relative error was lower than 2%, we present only the FIT break down for all errors. For the K40, on the contrary, 50% to 75% corrupted executions had all the elements with a relative error lower than 2%. Therefore, if we tolerate 2% of discrepancy from the correct value, K40's reliability is at least 60% better than when considering all mismatches.

It is worth noting that for the K40 errors distribution changes when results are filtered with the 2% tolerance. Random distributed errors almost disappear while single and line errors are significantly lowered. The 2% filter does not clear those incorrect elements with a magnitude higher than 2%. One execution classified as square may change to line or single when some elements are filtered. Unfortunately, the spatial distribution after the filter depends on the magnitude of each incorrect element and cannot be easily predicted.

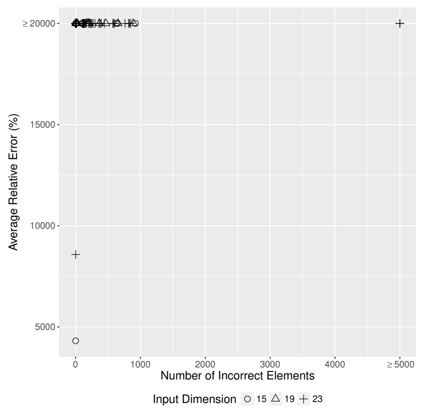
Spatial locality has a strong impact on the effectiveness of hardening strategies like ABFT [20]. Single and line are easily corrected in linear time on parallel devices [33], [47] while square and random errors are more difficult to detect and correct. Therefore, applying ABFT, *DGEMM* would be affected by only 20% to 40% of all errors on K40, and 60% to 80% on Xeon Phi.

Even if an exhaustive comparison between K40 and Xeon Phi is out of the scope of this paper, comparing Figures 3a and 3b it is clear that even considering a 2% tolerance in the output, the K40 has still a higher error rate than the Xeon Phi. If ABFT is applied to both devices the error rates become comparable.

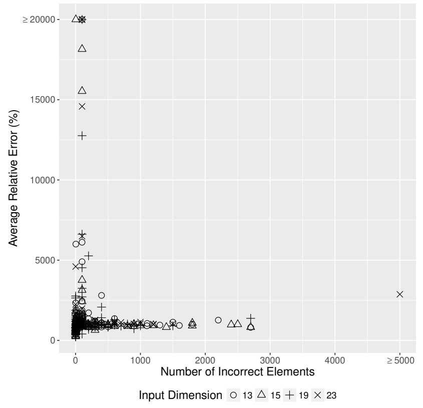
It is interesting to notice that the input size has a strong impact on K40 FIT but not on Xeon Phi FIT. From $2^9 \times 2^9$ to $2^{11} \times 2^{11}$ K40 FIT increases of $7\times$ for ALL and $5\times$ for $> 2\%$ while Xeon Phi FIT increases of only $1.8\times$. As discussed in Section IV-C, the different behavior between NVIDIA and Intel devices when input size is increased depends mainly on two reasons that derive from the different parallel threads management philosophies:

(1) Increasing the number of parallel threads increases the scheduler strain required to manage and dispatch threads. The scheduler on NVIDIA devices is implemented in hardware and has already been demonstrated to contribute to the device radiation sensitivity [34]. Intel Xeon Phi relies on the operating system to manage execution [22] which may be less susceptible to radiation-induced failures. It is worth noting that while the K40 thread management seems to increase its sensitivity, it may be more efficient. The K40 may then produce more correct data before experiencing a failure [34].

(2) NVIDIA and Intel adopt opposite solutions to manage those threads that are active but waiting to be dispatched. On the K40, active threads' data is kept in registers while other threads are being executed. A larger number of threads increases, then, the time data stays exposed in registers waiting to be used. The available ECC on K40 registers mitigates this effect, but data may still sit in internal queues or flip-flops that are not protected. On the contrary Xeon Phi waits for current threads (up to four per core) to finish before launching other ones. Subsequent threads' data sit in the DRAM, so there is no expected FIT increase caused by additional threads.



(a) K40.



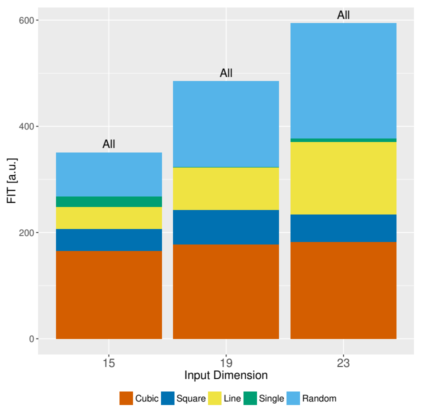
(b) Xeon Phi.

Fig. 4: *LavaMD* Mean relative error and Incorrect Elements.

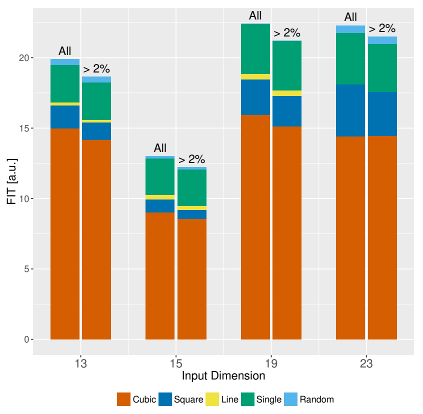
B. *LavaMD*

Mean relative error and number of incorrect elements for *LavaMD* are reported in Figures 4a and 4b. As the mean relative error is extremely high for *LavaMD* we represent errors with a mean average error up to 20,000%. Executions with a mean average error higher or equal to 20,000% are shown as 20,000% to improve figure quality). We hypothesize that the higher relative error of *LavaMD* compared to *DGEMM* is related to the exponentiation operation used when computing particle interactions, which can turn small value variations into large differences. The number of incorrect elements, on the contrary, is low and concentrated for the K40. Xeon Phi shows a higher number of corrupted elements than the K40 but a much lower average error. Although K40 simulates more particles than Xeon Phi (192 and 100 per box, respectively), Xeon Phi is affected by a larger number of incorrect elements. However, those corrupted elements for the Xeon Phi have an overall lower difference with the expected values.

Spatial locality and relative error for *LavaMD* is presented in Figures 5a and 5b. K40 has no errors with a relative error lower than 2% while Xeon Phi has only about one tenth of errors lower than the 2% threshold. Spatial locality highlights that most of the errors for Xeon Phi are cubic and square. K40

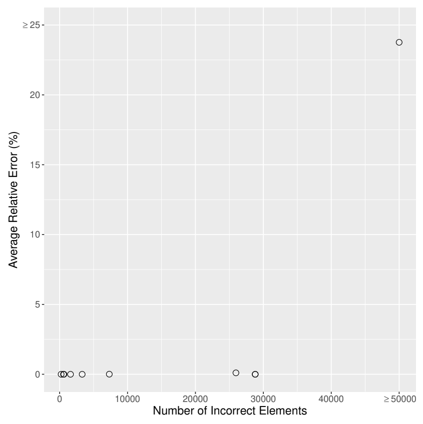


(a) K40.

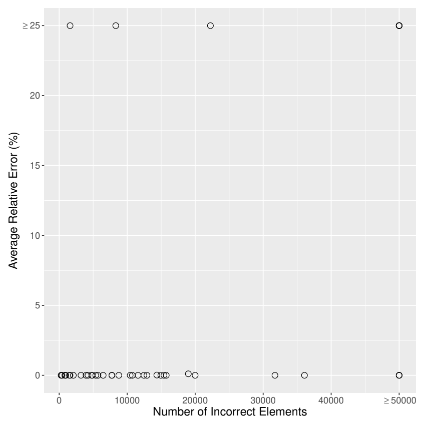


(b) Xeon Phi.

Fig. 5: *LavaMD* spatial locality and magnitude.



(a) K40.



(b) Xeon Phi.

Fig. 6: *HotSpot* Mean relative error and Incorrect Elements.

corrupted output affected by cubic and square error patterns are 40% to 60% of the total. The spatial locality for Xeon Phi is related to the larger number of incorrect elements which corresponds to an increased spatial area of corrupted data. As Xeon Phis have larger shared cache memories, it is easier for a single impinging particle to affect data used by multiple cores when running *LavaMD*.

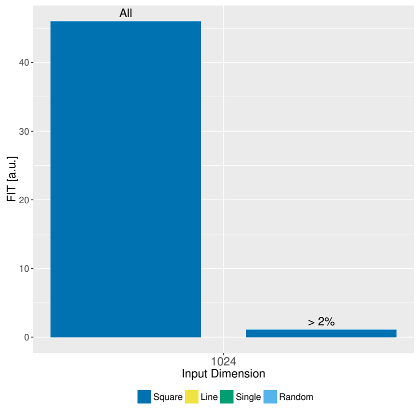
The percentage of K40 corrupted outputs with cubic and square error patterns are decreasing as the input dimension grows (55% of all corrupted output for 15 cubes, 50% for 19, and 42% for 23). With a larger input, more threads have to be scheduled and more data has to be read and written. The increased pressure in the GPU reduces the sharing of resources like caches, increasing the isolation between blocks of threads. This isolation, in turn, reduces the probability of corrupted data to be shared among many blocks, causing less cubic and square errors.

For the K40, *LavaMD*'s FIT rate increase with input size is only about 30% from one input size to the next one, definitely less than for *DGEMM*. This is only in apparent contrast with (1) and (2). In fact, *LavaMD* makes heavy usage of local memory ($\approx$14 KB per block of threads), which limits the number of active threads at any given time on the K40. Thus, the increase in number of active threads is limited for *LavaMD*, reducing the impact of the scheduler strain.

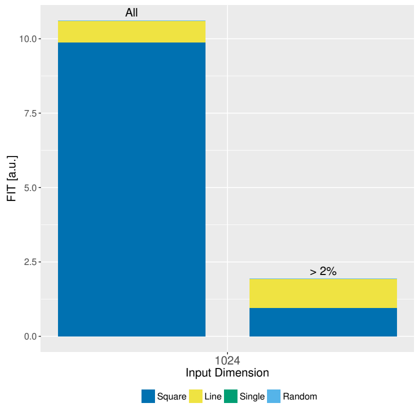
C. *HotSpot*

HotSpot values for mean relative error and incorrect elements are shown in Figures 6a and 6b. *HotSpot* shows an extremely low mean relative error (lower than 25% in all cases) independent of the number of incorrect elements for both architectures, which is due to intrinsic algorithm characteristics. *HotSpot* simulates energy dissipation taking into consideration the power input and the temperature of nearby cells. Therefore, errors will eventually dissipate as the result tend to reach an equilibrium. Analyzing the number of incorrect elements in Figures 6a and 6b, it is clear that Xeon Phi shows a greater tendency to have multiple errors than K40. K40, in fact, has at most about 50,000 incorrect elements in the output while Xeon Phi experienced up to 130,000 incorrect elements in the output (executions with a number of incorrect elements higher or equal to 50,000 are shown as 50,000 to improve figure quality).

Figures 7a and 7b depict the spatial locality and relative errors for *HotSpot*. Both architectures presented only square and line errors. The computation of each cell takes as direct input the values of the neighbor cells. Therefore, one single error will affect neighbor cells in the next iteration, always increasing spatial locality criticality. Considering only errors above 2%, *HotSpot* shows the most expressive results as we could consider as correct about 80% to 95% of faulty



(a) K40.



(b) Xeon Phi.

Fig. 7: *HotSpot* spatial locality and magnitude.

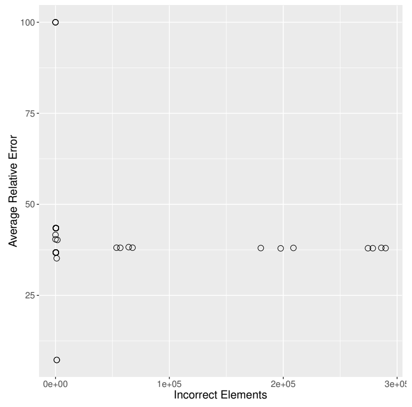


Fig. 8: *CLAMR* Mean relative error and Incorrect Elements for Xeon Phi.

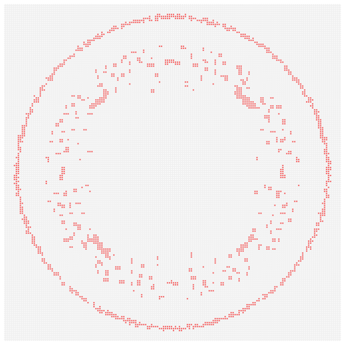


Fig. 9: *CLAMR* Error Locality Map. The output result is represented as a 2D matrix. Red dots are incorrect elements.

executions for Xeon Phi and K40, respectively.

HotSpot can greatly recover from errors naturally due to algorithm characteristics. Most of the faulty executions presented errors smaller than 2%. *HotSpot* is intrinsically robust and considering all mismatches as an error would erroneously decrease its resilience. Therefore, one can imprecisely classify *HotSpot* with a radiation sensitivity up to 95% higher considering any mismatch with the expected value as the sole metric.

The evaluation of neighbors to detect disparities coming from errors in Stencil-like applications like *HotSpot* can be difficult. The criticality results show that an error could be dissipated to neighbors leading to small disparities but with significant accumulated error in the affected elements. Thus, to detect an error, the checking routine would need to be executed constantly, reducing performance. The system entropy could be evaluated to detect a widespread error in stencil-like applications, especially if the system is isolated and the entropy need to be constant. However, for non-isolated systems, if the growing or lowering of system entropy is well-behaved, the entropy could be checked at regular time intervals to detect disturbances caused by induced errors. The time interval could be adjusted to better detect errors without affecting performance too much.

D. CLAMR

Fig. 8 shows the mean relative error and number of incorrect elements for *CLAMR* on Xeon Phi. We do not have the results for K40 as *CLAMR* is a LANL's proprietary workload to be used in supercomputers like Trinity, which will be based on Xeon Phis.

CLAMR shows a mean relative error between 25% and 50% while incorrect elements are definitely high. When mapping the incorrect elements to the 2D grid, most of the forms were similar to the one presented in the Fig. 9. We can see that a wave of incorrect elements was propagating confirming the fault injection analysis performed in [19]. Incorrect elements are, then, not isolated, affecting first the neighborhood and propagating as a wave, increasing the number of incorrect elements as the executions continue.

All the faulty elements of *CLAMR* have relative errors greater than 2%. Square errors amount to 99% of spatial locality as the errors propagate to all directions as depicted in Fig. 9. *CLAMR* works with the shallow waters equations considering momentum as well as mass. *CLAMR* errors may change the total mass of the system and will not be recovered as the execution continue, on the contrary, because of the mass conservation principle, the error will keep affecting the solution. Therefore, among all the codes studied in this work, the error criticality of *CLAMR* was the most sensitive to radiation-induced errors.

The sparse spatial location of incorrect elements with a moderate to low relative error makes it hard to provide efficient techniques for detecting errors. Similar to *HotSpot*, one of the few ways that can be used to detect errors is to evaluate the whole system taking advantage of the mass conservation principle, where the summation of all the incorrect elements' errors can lead to a detectable mass difference. This mass check technique has already been implemented in *CLAMR* and fault injection showed a fault coverage of 82% [4]. Furthermore, the load imbalance of the algorithm can provide opportunities to include mass-consistency checking routines that introduce little overhead to the overall execution time.

E. Discussion

The tested codes were selected as they stimulate specific resources, and have peculiar control flow or arithmetic characteristics. Thus, it is reasonable to correlate the particular behaviors observed for Xeon Phi and K40 with the code proprieties. Additionally, in some cases we can extend the experimental criticality analysis to other codes that share the same principles and data structures of the tested codes.

DGEMM is part of several applications and similar codes are even used as the standard HPC performance benchmark. Our analysis shows that K40 provides a lower error criticality (i.e., smaller difference with the expected value and fewer corrupted elements) for applications using *DGEMM*. We believe this is an intrinsic characteristic of GPUs, which have shortened and faster pipelines compared to CPUs. As a result, purely arithmetic operations, that are not based on control-flow instructions, are likely to be executed in a faster and more reliable way on a GPU.

Solvers using FDM like *LavaMD* will have a lower relative error on Xeon Phi, in contrast to *DGEMM*. Transcendental functions play a key role in FDMs performance and reliability. In the case of *LavaMD*, the exponentiation operations can turn small value variations into large differences. This is especially critical for the K40, for which all the SDCs are significantly different from the expected value. A hypothesis is that the transcendental function unit in the K40 is more prone to corruption. Its reliability should be improved in the future. Also, *LavaMD* performs dot products between particles which prevent the attenuation of transient errors with other correctly calculated particles.

While the Xeon Phi seems more resilient than the K40 when executing *LavaMD*, it produced more incorrect elements, leading to a more widespread spatial locality. To choose the platform in which to execute a FDM algorithm, it is fundamental to evaluate the trade-off between having more incorrect elements with lower relative errors (so the Xeon Phi) or the contrary (so the K40). Such a trade-off strictly depends on how FDMs outputs are used.

Stencil applications have been proved to be the most resilient ones. *HotSpot* showed that most of the errors have less than 2% of relative error. K40 seems slightly more resilient than Xeon Phi as the former shows less incorrect elements than the latter. We believe this behavior to be common for stencil applications that iteratively update the solution based on the current state. For these applications a transient fault could modify the current simulation state but, in the following iterations, the error is smoothed and filtered.

Fluid dynamics like *CLAMR* are less reliable, especially simulations that involve invariants such as mass or energy conservation. The impact of errors in such algorithms only increases with execution time as the invariant is now altered, affecting neighbor elements in the following iterations.

Based on our experimental analysis we can compare the reliability of some peculiarities of Xeon Phi and K40 architectures. Xeon Phi shows a tendency to have more incorrect elements than K40. Xeon Phi has larger caches than K40, so its data is not evicted as often. Hence, corrupted data, once in the caches, will be used by more elements before eviction. As a result, the same error spreads affecting several output elements.

The comparison of results with different input sizes for *DGEMM* and *LavaMD* highlights that the hardware scheduler makes the FIT rate dependent on the number of instantiated threads. On the contrary, the Xeon Phi operating system seems less prone to be corrupted. It is worth noting that the hardware scheduler may be more efficient, reducing the execution time and, thus, the number of neutrons that hit the device during computation. Other architectural decisions like long pipelines or large caches that keep alive data for a long time during computation modify both the code reliability and execution time, enhancing performance but leaving data more exposed to radiation strikes. Thus, the architectural design must tune the performance gain obtained by such decisions with the reliability issues incurred [35].

Finally, the spatial locality and magnitude of the errors measured for the different applications and devices can help users understand incorrect results generated from radiation-induced errors, and guide the usage of detectors and replication mechanisms [8].

VI. Conclusion and Future Work

In this paper we have addressed the error criticality in HPC systems. As radiation effects are a huge concern for today's HPC machines, it is fundamental to deeply understand the impact of errors in the applications' output.

We have selected four metrics to qualify and quantify radiation-induced errors on two state-of-the-art accelerators (Xeon Phi and K40) executing three representative benchmarks and an application. The unprecedented amount of beam time and experimental data available permits a deep and representative study of modern HPC accelerators reliability.

The selected metrics allow a deep understanding of radiation-induced corruptions as well as a correlation between observed error patterns and device architecture. Additionally, as the selected codes are representative of broader algorithm classes we can extend and generalize (under certain premises) the performed analysis.

We show that K40 is clearly more resilient when doing DGEMM (considering relative error and error locality). LavaMD has smaller relative errors on Xeon Phi but errors are spread (cubic). HotSpot, being an iterative stencil, has a very small relative error but very spread errors (mostly square). Finally, CLAMR differs from HotSpot with spread errors but with large relative errors (wave of errors).

In the future we plan to perform fault injection on both the K40 and Xeon Phi to detect the sources for the most critical errors. This information is going to be used to apply selective hardening to only those procedures, variables, or resources whose corruption is likely to produce the observed critical errors.

Acknowledgments

This work was supported by the STIC-AmSud/CAPES scientific cooperation program under the EnergySFE research project grant 99999.007556/2015-02, EU H2020 Programme, and MCTI/RNP-Brazil under the HPC4E Project, grant agreement n° 689772. Tested K40 boards were donated thanks to Steve Keckler, Timothy Tsai, and Siva Hari from NVIDIA.

References

[1] "Log data," https://github.com/UFRGS-CAROL/HPCA2017-log-data, 2016.

[2] "Why a chip that's bad at math can help computers tackle harder problems," 2016. [Online]. Available: "https://www.technologyreview.com/s/601263/why-a-chip-thats-bad-at-math-can-help-computers-tackle-harder-problems/"

[3] K. Asanovic *et al.*, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[4] B. Atkinson, N. Debardeleben, Q. Guan, R. Robey, and W. M. Jones, "Fault injection experiments with the clamr hydrodynamics mini-app," in *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, Nov 2014, pp. 6–9.

[5] D. Bailey, Barsck, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, S. Fineberg, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Parallel Benchmarks," *NASA Ames Research Center, RNR Technical Report RNR-94-007*, 1994. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/5/3/63

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

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

[8] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, *Exploring Partial Replication to Improve Lightweight Silent Data Corruption Detection for HPC Applications*. Cham: Springer International Publishing, 2016, pp. 419–430. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-43659-3_31

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[10] M. A. Breuer, "Multi-media applications and imprecise computation," in *Digital System Design, 2005. Proceedings. 8th Euromicro Conference on*. IEEE, 2005, pp. 2–7.

[11] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger, "Comparison of error rates in combinational and sequential logic," *Nuclear Science, IEEE Transactions on*, vol. 44, no. 6, pp. 2209–2216, 1997.

[12] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Oct. 2009, pp. 44–54. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2009.5306797

[13] W.-F. Chiang, G. Gopalakrishnan, Z. Rakamaric, D. H. Ahn, and G. L. Lee, "Determinism and reproducibility in large-scale HPC systems," in *Workshop on Determinism and Correctness in Parallel Programming (WoDet)*, 2013.

[14] J. de la Puente, M. Ferrer, M. Hanzich, J. E. Castillo, and J. M. Cela, "Mimetic seismic wave modeling including topography on deformed staggered grids," *GEOPHYSICS*, vol. 79, no. 3, pp. T125–T141, 2014.

[15] J. Dongarra, H. Meuer, and E. Strohmaier, "TOP500 Supercomputer Sites: June 2016," 2016. [Online]. Available: http://www.top500.org

[16] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov 2004.

[17] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, March 2014, pp. 221–230.

[18] L. A. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, S. Keckler, K. Pattabiraman, R. Rech, and M. S. Reorda, "GPGPUs: How to Combine High Computational Power with High Reliability," in *2014 Design Automation and Test in Europe Conference and Exhibition*, Dresden, Germany, 2014.

[19] Q. Guan, N. DeBardeleben, B. Artkinson, R. Robey, and W. Jones, "Towards Building Resilient Scientific Applications: Resilience Analysis on the Impact of Soft Error and Transient Error Tolerance with the CLAMR Hydrodynamics Mini-App," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, Sept 2015, pp. 176–179.

[20] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *Computers, IEEE Transactions on*, vol. C-33, no. 6, pp. 518–528, June 1984.

[21] "An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors," Intel. [Online]. Available: http://download.intel.com/newsroom/kits/xeon/phi/pdfs/overview-programming-intel-xeon-intel-xeon-phi-coprocessors.pdf

[22] "Intel Xeon Phi Coprocessor System Software Developers Guide," Intel. [Online]. Available: https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system-software-developers-guide.pdf

[23] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," JEDEC Standard, Tech. Rep. JESD89A, 2006.

[24] R. Lucas, "Top ten exascale research challenges," in *DOE ASCAC Subcommittee Report*, 2014.

[25] N. Mahatme, T. Jagannathan, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, "Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 6, pp. 2719–2725, 2011.

[26] S. S. Mukherjee *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*. IEEE Computer Society, 2003.

[27] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5dd blocking optimization for stencil computations on modern cpus and gpus," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–13. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.2

[28] J. Noh, V. Correas, S. Lee, J. Jeon, I. Nofal, J. Cerba, H. Belhaddad, D. Alexandrescu, Y. Lee, and S. Kwon, "Study of neutron soft error rate (ser) sensitivity: Investigation of upset mechanisms by comparative simulation of finfet and planar mosfet srams," *Nuclear Science, IEEE Transactions on*, vol. 62, no. 4, pp. 1642–1649, Aug 2015.

[29] "NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110," NVIDIA. [Online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[30] "CUDA C Programming Guide," http://docs.nvidia.com/cuda/cuda-c-programming-guide/, NVIDIA, 2015.

[31] "Kepler Tuning Guide :: CUDA Toolkit Documentation," http://docs.nvidia.com/cuda/kepler-tuning-guide/, NVIDIA, 2015.

[32] H. Quinn, W. H. Robinson, P. Rech, M. Aguirre, A. Barnard, M. Desogus, L. Entrena, M. Garcia-Valderas, S. M. Guertin, D. Kaeli, F. L. Kastensmidt, B. T. Kiddie, A. Sanchez-Clemente, M. S. Reorda, L. Sterpone, and M. Wirthlin, "Using benchmarks for radiation testing of microprocessors and fpgas," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2547–2554, Dec 2015.

[33] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 4, pp. 2797–2804, 2013.

[34] P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro, "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, 2014.

[35] G. Reis, J. Chang, N. Vachharajani, and S. Mukherjee, "Design and evaluation of hybrid fault-detection systems," in *Proceedings of the 2005 International Symposium on Computer Architecture, ISCA'05*. IEEE Press, 2005, pp. 148–159.

[36] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson *et al.*, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, pp. 1–45, 2014.

[37] V. Sridharan and D. R. Kaeli, "The effect of input data on program vulnerability," in *Proceedings of the 2009 Workshop on Silicon Errors in Logic and System Effects*, ser. SELSE '09, 2009.

[38] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. Gunnels, "100+ tflop solidification simulations on bluegene/l," in *Proceedings of IEEE/ACM Supercomputing*, vol. 5, 2005.

[39] L. G. Szafaryn, T. Gamblin, B. R. De Supinski, and K. Skadron, "Experiences with achieving portability across heterogeneous architectures," *Proceedings of WOLFHPC, in Conjunction with ICS, Tucson*, 2011.

[40] J. Tan, N. Goswami, T. Li, and X. Fu, "Analyzing soft-error vulnerability on GPGPU microarchitecture," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, Nov 2011, pp. 226–235.

[41] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. Debardeleben, P. Navaux, L. Carro, and A. B. Bland, "Understanding GPU Errors on Large-scale HPC Systems and the Implications for System Design and Operation," in *Proceedings of 21st IEEE Symp. on High Performance Computer Architecture (HPCA)*. ACM, 2015.

[42] M. Violante, L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, C. Andreani, G. Gorini, A. Pietropaolo, G. Cardarilli, S. Pontarelli, and C. Frost, "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 1184–1189, 2007.

[43] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 293–305.

[44] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Dover Publications, Incorporated, 1994.

[45] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[46] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2. ACM, 1995, pp. 24–36.

[47] H.-J. Wunderlich, C. Braun, and S. Halder, "Efficacy and efficiency of algorithm-based fault-tolerance on gpus," in *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, July 2013, pp. 240–243.