Software-Based Hardening Strategies for Neutron Sensitive FFT Algorithms on GPUs

L. L. Pilla, P. Rech, F. Silvestri, C. Frost, P. O. A. Navaux, M. Sonza Reorda, and L. Carro

***Abstract*—In this paper we assess the neutron sensitivity of Graphics Processing Units (GPUs) when executing a Fast Fourier Transform (FFT) algorithm, and propose specific software-based hardening strategies to reduce its failure rate. Our research is motivated by experimental results with an unhardened FFT that demonstrate a majority of multiple errors in the output in the case of failures, which are caused by data dependencies. In addition, the use of the built-in error-correction code (ECC) showed a large overhead, and proved to be insufficient to provide high reliability. Experimental results with the hardened algorithm show a two orders of magnitude failure rate improvement over the original algorithm (one order of magnitude over ECC) and an overhead 64% smaller than ECC.**

***Index Terms*—GPU, FFT, neutron sensitivity, ECC, software-based hardening strategies**

I. Introduction

THE Fast Fourier Transform (FFT) is one of the most representative algorithms in high performance computing. FFT algorithms are used in several applications such as signal processing, vibration and spectrum analysis, speech processing, communication, linear algebra, statistics, 3D reconstruction, and stock options pricing determination [1], [2]. In addition to this pervasiveness, the FFT algorithm is also highly parallel, which makes it a suitable candidate for acceleration in Graphics Processing Units (GPUs).

Nowadays, every desktop computer, laptop, or portable device includes at least one GPU, mainly used as a support for the Central Processing Unit (CPU) to accelerate graphics rendering. Due to their highly parallel structure, GPUs are more effective than general-purpose CPUs when large blocks of data need to be processed in parallel, and have recently become popular for high performance computing. For instance, TITAN, the second most powerful of supercomputers [3] in November 2013, includes $18,000$ GPUs to enhance its performance.

As we have demonstrated in previous works, radiation-induced errors, including those generated by the terrestrial neutron radiation environment at ground level, are one of the major issues for the newest GPU cores reliability [4], [5]. Still, the state of the art lacks studies describing radiation test methods for extremely parallel systems and parallel algorithm behavior analysis in radiation environments. Motivated by these facts, we investigate the behavior of a parallel Fast Fourier Transform algorithm executed on a GPU irradiated with neutrons.

The results of extensive radiation test campaigns attest that the FFT algorithm experiences a very high error rate and, in the majority of the cases, its output is affected by multiple errors. As we demonstrate in this paper, based on algorithmic and architectural analysis, multiple errors occur mainly because the FFT computation requires sequential iterations which are dependent on data from previous iterations. If in a given iteration a thread is corrupted by radiation, the error is then likely to spread over the following iterations leading to multiple output errors. Additionally, we experimentally demonstrate that the error-correction code (ECC) available in the latest GPUs is not sufficient to ensure by itself a high reliability as it address only L1 and L2 cache memories and registers, leaving logic and scheduling resources unprotected.

In this scenario, we propose a dedicated software-based hardening strategy for the FFT algorithm executed on a GPU based on the Algorithm-Based Fault Tolerance (ABFT) philosophy [6]. The resulting algorithm, named ABFT-FFT, uses a hardening strategy designed to profit from the FFT properties demonstrated by Jou and Abraham [7], and applies them in the GPU algorithm to detect and correct faulty executions. The results of our experimental evaluation with ABFT-FFT under an accelerated neutron beam show a two orders of magnitude failure rate reduction over the original FFT algorithm with an increase in execution time of only 18%. We also extend the proposed ABFT approach to achieve prompt error detection and prevent errors propagation, and demonstrate its applicability with fault-injection experiments.

The remaining sections of this paper are organized as follows: the experimental methodology, including the tested devices and algorithms, is detailed in Sec. II. An evaluation of the original FFT algorithm and the use of ECC is discussed in Sec. III. Our hardening strategies and their evaluations are presented in Sec. IV. Concluding remarks and future work are drawn in Sec. V.

II. Experimental Methodology

A. Neutron Beam

Radiation experiments were performed at the ISIS facility in the Rutherford Appleton Laboratories (RAL) in Didcot, UK [8]. The available neutron flux was of about $5.5 \times 10^4\ n/cm^2s$. The beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra, which is enough to fully and homogeneously irradiate the GPU chip without directly

L. L. Pilla, P. Rech, P. O. A. Navaux, and L. Carro are with the Instituto de Informatica, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil (email: llpilla, prech, navaux, carro @inf.ufrgs.br).

F. Silvestri is with the Dipartimento di Ingegneria dell'Informazione, University of Padova, Italy (email: silvest1@dei.unipd.it).

C. Frost is with STFC, Rutherford Appleton Laboratories, Didcot, UK (email: christopher.frost@stfc.ac.uk).

M. Sonza Reorda is with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy (email: matteo.sonzareorda@polito.it).

affecting nearby power control circuitry and DDR memories on the board. Nevertheless, even if the beam is collimated, scattering neutrons may still wander from the beam spot. Thus, to ensure that the DDR content was consistent during our experiments, we periodically checked it during experiments, but no error has been observed.

It is worth noting that input and output data were stored in the GPU's DDR memory, and no L1 cache memory was employed, so the errors reported in the following sections were only caused by the corruption of the GPU core logic resources or internal flip-flops. Irradiation was performed at room temperature with normal incidence, nominal voltages and frequency of operation.

B. Tested Devices

We tested commercial-off-the-shelf Tesla C2050 GPUs designed by NVIDIA and manufactured in a 40 nm technology node. The C2050 model includes 14 Streaming Multiprocessors (SMs), each of which is divided into 32 Compute Unified Device Architecture (CUDA) cores [9]. In the C2050 GPU, 14 blocks of threads can be executed in parallel with a maximum of 32 threads in each block for a total of 448 threads. If more threads or blocks are instantiated, their execution will be delayed until they can be scheduled.

NVIDIA provides the newest GPUs, such as the C2050 family, with an internal ECC mechanism that can be activated by the user. This mechanism was first disabled for the evaluation of the FFT sensitivity to neutrons, and later enabled. A discussion on the efficiency and drawbacks of the NVIDIA ECC mechanism takes place in Sec. III-B.

It is important to emphasize that the input vectors, as well as the results of computation, are stored in the GPU board DDR, which was not irradiated. On a realistic application, a higher number of blocks to be processed may extend the exposure time of input or output data, increasing the probability of having them corrupted by neutron-induced Single Event Upsets or Single Event Functional Interruptions. However, DDR sensitivity has been proved to decrease with the shrinking of technology nodes [10], and modern DDR memories are provided with efficient ECCs that increase in several orders of magnitudes the device reliability [11]. It is then reasonable to consider negligible, even on a real application, the probability for GPU input or output vectors to be corrupted.

C. Tested FFT Algorithm

We tested under radiation a benchmark that implements 512×512 1D-FFTs of 64-point each. The FFT input is composed of a $64 \times 512 \times 512$ double precision floating-point matrix for the real part and a $64 \times 512 \times 512$ matrix for the imaginary part. We choose to test relatively small FFTs (64-point) to limit the number of iterations and ease the study of error propagation, while having 512×512 1D-FFTs eases the gathering of a statistically significant amount of errors.

The implemented algorithm is based on the Fast Fourier Transform kernel of the NAS Parallel Benchmarks [12], named FT, implemented in C and ported to the GPU architecture using the NVIDIA CUDA programming model. Each 64-point 1D-FFT kernel is composed of 6 sequential iterations ($\log_2 64 = 6$) of a variant of the Stockham FFT algorithm [13].

Threads update the values of complex floating-point elements of the matrix in pairs using their previous values as input. This behavior mimics a butterfly module [7], which is illustrated in Fig. 1. This process involves reading four floating-point values to registers (two real and two imaginary values), using them once to compute their updated values, and writing them in memory. Due to the amount of data being processed in parallel, the FFT is not able to benefit from a caching mechanism.

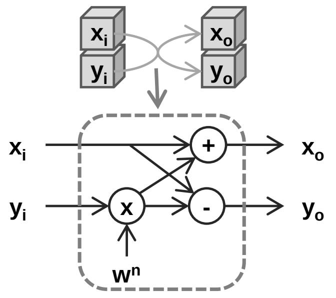

Fig. 1. A basic butterfly module used to update two-by-two all the 64 elements composing the FFT.

The execution of the FFT algorithm in the GPU requires the instantiation of 512×512 parallel threads. They are grouped in blocks of 512 threads each. In this scenario, a thread is in charge of evaluating the FFT values on its assigned complex vector of size 64.

III. Experimental Results and Discussion

This section presents the experimental results obtained with the plain FFT algorithm (i.e. the algorithm without any hardening solution applied, as described in the previous section) as well as with the use of ECC.

A. Sensitivity of the original FFT

Table I reports the experimentally measured cross section and failures in time (FIT, or failures in 10^9 hours) for the tested FFT code under the neutron beam discussed in Sec. II-A. The cross section is obtained dividing the number of faulty FFT executions per unit time by the flux. As the ISIS neutron spectrum of energies is suitable to mimic the terrestrial one, FITs can be calculated multiplying the experimentally obtained cross section by the expected natural neutron flux at sea level (about 13 n/cm^2h [14]). The values reported in Table I confirm that GPUs are extremely prone to be corrupted by neutrons.

TABLE I
512×512 64-point FFTs cross section and FIT at NYC.

	Cross section (cm^2)	FIT
512×512	$3.69 \times 10^{-6} \pm 4\%$	4.80×10^4

The tested FFT algorithm produces complex double-precision floating-point data as output, which can be split into real and imaginary parts. An execution is considered as faulty if at least one difference with respect to the expected value is detected in the real, imaginary, or both parts of the output.

Table II shows the percentage of faulty executions of the FFT algorithm in which the real part or the imaginary part was detected as corrupted, as well as the cross section and FIT of each of these parts. Even if the implemented algorithm is symmetric for the real and imaginary parts, as illustrated in Fig. 1, some executions experienced errors in just one of the parts. These errors are caused by the corruption of internal registers used by threads for storing the intermediate values of the complex number. As stated in the second column of Table II, more than 90% of the executions considered as faulty experience errors on both the real and imaginary part of the output. In all other situations, the radiation induced error corrupted a resource in a thread and its effect was confined in the real or imaginary part of the result, without propagating to the other part.

TABLE II
REAL AND IMAGINARY PART ERRORS ON 512×512 64-POINT FFTS.

	Percentage	Cross section (cm^2)	FIT
FFT Real	94.96%	$3.50 \times 10^{-6} \pm 4\%$	4.55×10^4
FFT Imaginary	96.17%	$3.55 \times 10^{-6} \pm 4\%$	4.62×10^4

The computation steps in the FFT algorithm are not independent, since an iteration uses the output of previously executed ones to update the real and imaginary parts of two complex elements (see Fig. 2). It is then very likely that a radiation-induced event affecting a thread in the early stages of the FFT execution will spread, affecting various bits of the output. This fault behavior is illustrated in Fig. 2, where an error in the first iteration of the FFT corrupts the whole output vector.

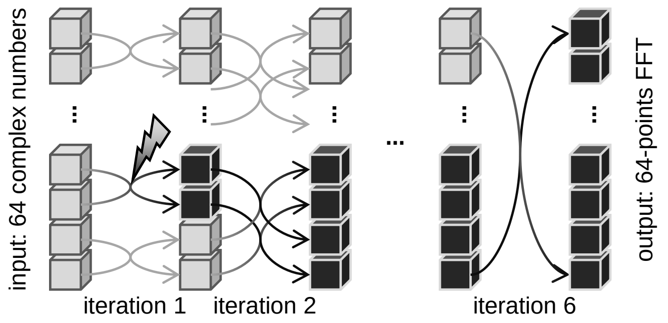

Fig. 2. Graphical representation of the FFT algorithm under radiation. At each iteration, a thread updates two-by-two all the 64 values of the FFT using the basic butterfly module. Six iterations are necessary to complete the execution. If an operation in one iteration is corrupted by radiation, two (or more) values will be wrongly updated, and the number of errors will double in the following iteration.

As a thread is in charge of updating two complex values, a radiation-induced error that prevents the thread from completing its execution or corrupts the thread input data produces at least two output errors. Nevertheless, a single error in a thread can be generated by the corruption of the internal register that stores the value of just one of the two elements to update, or disturbing just one of the operations needed to compute the FFT. A thread can then complete its execution, allowing the correct calculation of the second complex number. Single output errors occur in the FFT only if such single thread error occurs in the last FFT iteration. To illustrate that, Table III reports the proportion of single and multiple errors in each part of the complex values. Single output errors occurred in just 1.63% and 4% of the faulty executions for the real and imaginary parts, respectively. These errors should become even less likely with an increase in the number of points computed by the FFT algorithm, as the number of iterations increases by the logarithm of the number of points, and so the portion of execution time spent on iterations other than the last one also increases.

TABLE III
SINGLE AND MULTIPLE ERRORS ON 512×512 64-POINT FFTS.

	Number of errors	Percentage	FIT
FFT Real	Single	1.61%	7.33×10^2
	Multiple	98.39%	4.48×10^4
FFT Imaginary	Single	4.00%	1.85×10^3
	Multiple	96.00%	4.43×10^4

The importance of the occurrence of multiple errors in realistic applications is highlighted in the last column of Table III, in which the FIT values of FFT executions affected by single or multiple errors in the real and imaginary parts are reported.

The experimentally observed multiple error distributions are shown in Fig. 3. It is worth noting that, in most cases, 64 or less output values were found corrupted, and those values belong to the same 64-point FFT. Such error patterns are caused by the propagation of errors from one iteration to the following ones in the same 64-point FFT, as previously represented in Fig. 2. As said, the amount of errors is likely to double at each iteration. Thus, it is very unlikely to have an odd number of errors in the output. This assumption is in agreement with experimental data reported in Fig. 3.

The worst case for a 64-point FFT occurs when radiation affects a thread in its first iteration. In such scenario, the corruption of the thread input or a thread functional interruption produces 64 output errors in the FFT. Meanwhile, if a single error is produced in a thread in its first iteration, 32 errors are expected in its output. This amount of errors is a result of the following 5 iterations doubling the amount of corrupted values in the FFT, as it executes for 6 iterations in total, and $2^5 = 32$. In this sense, having between 32 and 64 errors in the output vector is very improbable. In fact, as it is very unlikely to have two neutrons corrupting the GPU in the same FFT execution, the only way of having more than 32 errors is to have a thread in the first iteration to generate two or more errors that spread to 64 errors in the output.

Finally, only few executions experienced more than 64 errors in the output. This situation occurred when radiation led

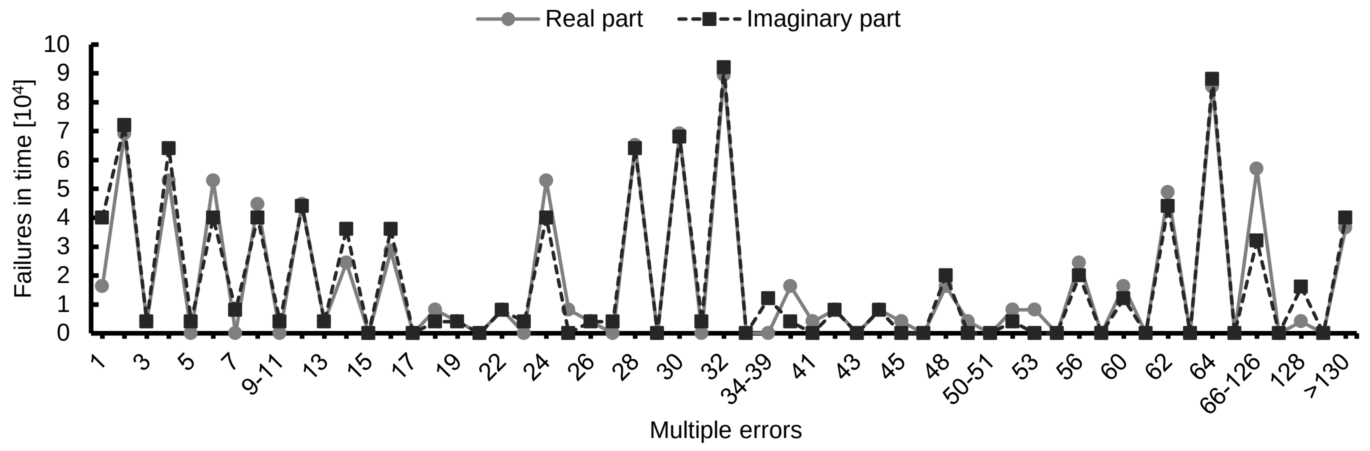

Fig. 3. FFT real and imaginary parts multiple output errors FIT. Consecutive distributions that were never experimentally observed are grouped in the picture (for instance, 9 to 11 errors, and 20 to 21). Statistical error is limited to 5%.

an SM to experience a functional interruption preventing a whole warp of 32 threads or even a whole block of 512 threads from completing their execution. This interruption ends up affecting the outputs of more than one 64-point FFT. Those errors will then spread and a huge amount of errors are expected at the output.

B. *Sensitivity with error-correction code*

NVIDIA latest GPUs, including the irradiated C2050s, are provided with an ECC mechanism that can be activated or deactivated by the user. The ECC is applied to the cache and to the internal memory of the SM, and it is able to correct single error and detect double errors [15].

When ECC is turned ON, 12.5% of the device memory becomes unavailable to the user. In addition, depending on the algorithm and device, ECC typically reduces the GPU performance in the range of $20-30\%$ [15]. However, in the case of the tested FFT algorithm, a performance overhead of 50% was measured in our experiments, as reported in Table IV. In this sense, the computational and area overhead introduced by ECC are far from being negligible, and may compromise the GPU efficiency. Unfortunately, no detailed information about the implementation of the ECC mechanism is currently available. The analysis of the ECC efficiency and drawbacks is then limited to what was experimentally observed.

Table IV shows the average execution time of 512×512 64-point FFTs with and without the use of ECC. It also includes the measured cross sections under the neutron beam discussed in Sec. II-A. Data for the FFT without ECC row comes from the results presented in Sec. III-A.

When ECC was enabled on the irradiated GPU, the observed number of output errors was reduced by about one order of magnitude. In particular, no single output error occurred, while multiple errors patterns formed of 64 or more corrupted locations were still observed. Those errors are generated by SM functional interruptions and scheduler failures that prevent threads from completing execution, which are not detectable by the ECC mechanism. The proposed software-based hardening strategies presented in the next section are actually capable of dealing even with this kind of failures.

TABLE IV
FFT EXECUTION TIMES, CROSS SECTIONS, AND FITS MEASURED WITH AND WITHOUT THE USE OF ECC.

	Execution time	Cross section (cm^2)	FIT
FFT w/o ECC	106 ms	$3.69 \times 10^{-6} \pm 4\%$	4.80×10^4
FFT w/ ECC	159 ms	$5.33 \times 10^{-7} \pm 6\%$	6.92×10^3

IV. Hardening Strategies for N-point FFTs

This section discusses the proposed strategies to improve the error detection and correction capabilities of N-point FFTs. We start by discussing the use of Algorithm-Based Fault Tolerance (ABFT) [6], followed by its experimental validation using a neutron beam. An extension of the algorithm to reduce the recomputation time of the FFTs is discussed in the last subsection.

A. *Algorithm-Based Fault Tolerance*

In order to reduce the number of radiation-induced output errors, we devised ABFT-FFT, an ABFT technique for FFT algorithms. The proposed hardening strategy derives from the fault-free N-point FFT network of N processors presented by Jou and Abraham [7], which is based on the superposition principle of linear systems and the circular shift property of the FFT. The basic idea is to detect errors arising in any processor or connection with the use of input coding and a checksum comparison at the output.

We implement the fault-free network in software in the GPU viewing each thread in the GPU as a network. Each CUDA core in a GPU can be considered as an isolated unit such that a radiation-induced event in one CUDA core only corrupts the thread currently assigned to it. Threads that follow the corrupted one or threads assigned to computing units near the faulty one will not be affected. This assumption maintains the same set of premises presented by Jou and Abraham [7],

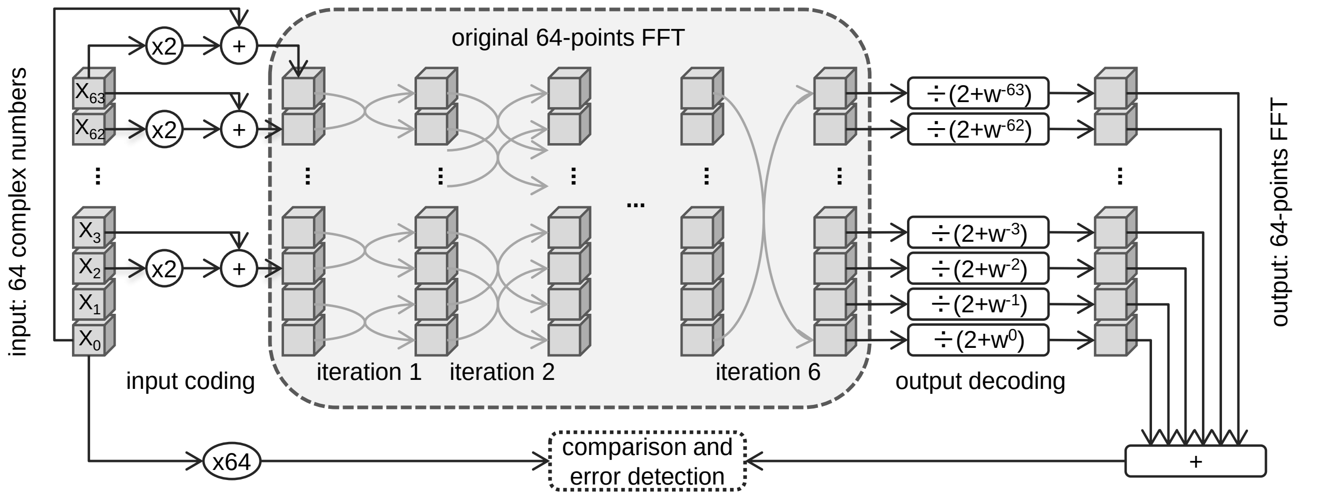

Fig. 4. FFT hardening scheme. The 64 complex elements of the input are coded, then the 64-point FFT is performed with the original algorithm, and its output is decoded. Errors are detected comparing the checksum generated summing the output values with $X_0 \times N$.

and hence the same mathematical demonstration ensuring the correctness and efficacy of the approach can be applied.

A few code modifications are required to implement the ABFT mechanism. They focus on modifying the input and output data, and do not require any changes to the core of the FFT algorithm. Considering the FFT algorithm as a network of butterfly modules (as seen in Fig. 2), its ABFT version can be seen as a network with some pre- and post-processing, as illustrated in Fig. 4. Algorithm 1 provides a more comprehensive view of the GPU implementation of ABFT-FFT.

Algorithm 1: ABFT-FFT: parallel algorithm-based fault-tolerant 1D-FFT.

```
Input: complex matrix $X$, thread id $tid$, FFT size $N$
Output: complex matrix $Y$, vector $v1$, vector $v2$
1  $i \leftarrow 0$
2  $v1[tid] \leftarrow$ ERR_VAL; $v2[tid] \leftarrow$ ERR_VAL
3  while $i <$ ITER_LIM $\wedge v1[tid] \neq COR\_VAL \wedge v2[tid] \neq COR\_VAL$ do
4      $X' \leftarrow$ input_coding$(X, tid)$
5      for $j \leftarrow 1$ to $\log N$ do
6          $X' \leftarrow$ fft_iteration$(X', j, tid)$
7      $Y \leftarrow$ output_decoding$(X', tid)$
8      if error_detection$(X, Y, tid)$ = false then
           /* No errors were detected */
9          $v1[tid] \leftarrow COR\_VAL$
10         $v2[tid] \leftarrow COR\_VAL$
11     $i \leftarrow i + 1$
12 return $Y, v1, v2$
```

In this parallel algorithm, each thread on the GPU computes one N-point FFT. For instance, in the case of the tested version, 512×512 threads are instantiated, and each computes one 64-point FFT. A thread takes its input sequence of N complex elements X and encodes it, resulting in the sequence X'. This process is represented in line 4 and based on Eq. 1. The original vector X is kept unmodified for the situation where the FFT has to be recomputed due to any internal error detected by our mechanism.

$$X'(i) = \begin{cases} 2X(i) + X(i+1) & 0 \leq i < N-1 \\ 2X(i) + X(0) & i = N-1 \end{cases} \tag{1}$$

The FFT is then evaluated using the coded values X' as input, as illustrated in lines 5 and 6 of Algorithm 1. When calculation is complete, the output X' is decoded through Eq. 2,

$$Y(k) = \left\{ \frac{1}{2+w_N^{-k}} X'(k) \quad 0 \leq k < N \right. , \tag{2}$$

where w_N^{-k} is the N^{th} root of the unity. The N decoded results are then summed, generating a checksum. As formally demonstrated by Jou and Abraham [7], this encoding and decoding scheme gives each output a nontrivial weighted contribution to the checksum such that any error will cause a detectable error syndrome. After computation, the checksum is compared to $N \times X(0)$. Any mismatch will identify the FFT as faulty, and will signalize the necessity of a recomputation.

In addition to coding and decoding data, our parallel algorithm includes two vectors $v1$ and $v2$. Their sizes are equal to the number of FFTs being computed (or the number of threads), which is much smaller than the size of matrix X. The additional vectors are employed to signalize any FFT recomputations needed, and to detect any failures from SM functional interruptions and scheduler failures that could prevent threads from completing their execution. We employ two vectors instead of one for redundancy. Both vectors are initialized with an error value ERR_VAL, and set to a correct value COR_VAL when no errors are detected in the FFT. These vectors are later evaluated in the CPU to check if any cell differs from the correct value. If that is the case, it means that a thread was not able to successfully finish its computations. We never observed in any of our experiments a difference in the value of the two vectors.

Lastly, an iteration limit $ITER\_LIM$ is used to choose the maximum number of FFT recomputations allowed. This mechanism is applied to guarantee that the kernel will finish even if a permanent failure happens in the GPU, or even if the input data gets corrupted. Still, no FFT was recomputed more than once in our experiments.

Our Algorithm-Based Fault Tolerance mechanism increases the GPU memory usage with the addition of the two vectors $v1$ and $v2$, and the encoded matrix X'. When compared to the use of ECC, our algorithm provides a smaller memory overhead if the input matrix is smaller than 12.5% of the memory available in the GPU. Even if this is not the case, our mechanism does not degrade memory performance as ECC does (see Sec. III-B).

Although performance degradation is expected when comparing our ABFT-FFT to the original FFT due to the insertion of more processing, ABFT-FFT keeps the computational complexity of the original algorithm, which is $O(n \log n)$ for a n-point FFT. This complexity is maintained because the functions for input coding, output decoding, and error detection are $O(n)$.

B. ABFT-FFT Neutron Sensitivity

Table V reports the experimentally measured cross section and FIT for the ABFT-FFT algorithm under the neutron beam discussed in Sec. II-A. It also presents the cross section and FIT of the original FFT algorithm with and without the use of NVIDIA ECC for comparison. As it can be seen in Table V, the ABFT-FFT algorithm provides a two orders of magnitude improvement over the original FFT without ECC, and a one order of magnitude improvement over the use of ECC.

TABLE V
512×512 64-POINT FFT AND ABFT-FFT CROSS SECTION AND FIT AT NYC.

	Cross section (cm^2)	FIT
FFT w/o ECC	$3.69 \times 10^{-6} \pm 4\%$	4.80×10^4
FFT w/ ECC	$5.33 \times 10^{-7} \pm 6\%$	6.92×10^3
ABFT-FFT	$6.56 \times 10^{-8} \pm 4\%$	8.52×10^2

Besides detecting and correcting failures in the GPU when possible, ABFT-FFT also detects scheduler failures that prevented threads from completing their execution. When scheduler failures happen, the output vector does not contain the output values from the FFT. By using two verification vectors, as described in Sec. IV-A, ABFT-FFT is able to diagnose any interrupted FFTs, which leads to a better failure coverage.

Table VI shows the execution time of ABFT-FFT running on the GPU, and compares it to the execution time of the original FFT with and without ECC. As it can be noticed, the additional coding, decoding, and verification steps incur in a 18% overhead in execution time. This overhead represents an increase in execution time almost two-thirds smaller than the use of ECC.

TABLE VI
512×512 64-POINT FFT AND ABFT-FFT EXECUTION TIMES.

	Execution time	Overhead
FFT w/o ECC	106 ms	–
FFT w/ ECC	159 ms	50%
ABFT-FFT	125 ms	18%
ABFT-FFT w/ errors	125 ms	18%

Besides comparing the performance of the ABFT-FFT algorithm without radiation failures, we also evaluated its performance when detecting and correcting failures. In order to inject errors during the execution of the ABFT-FFT, additional code was inserted in the GPU which modifies data stored in the output register of other threads using probabilities of injection that derive directly from the experimental results reported in the previous section. As reported in Table VI, the recomputation of an FFT due to errors does not increase the total execution time of ABFT-FFT. As 512×512 64-point 1D-FFTs are being computed in the GPU at the same time, the overhead of recomputing just one 1D-FFT is hidden in the execution time of the concurrent threads. The same happens when forcing errors in all threads in a given block. Still, this recomputation time could become a problem with larger N-point FFTs, as the execution time of a single FFT increases.

C. Extended ABFT-FFT

As stated previously, when an error occurs in one iteration, it is likely to spread to the following ones. Therefore, a prompt detection of the error would help to prevent error propagation. The ABFT-FFT hardening strategy has the ability of detecting all the experimentally observed error patterns, but requires recomputation in order to provide the correct output, introducing a not negligible overhead. It is worth noting that FFTs larger than the tested ones are typically executed on GPUs.

To better illustrate the recomputation overhead of larger FFTs, Table VII reports the execution time of a single FFT (instead of 512×512 FFTs as before) of various dimensions, both for the unhardened version and for the proposed ABFT-FFT algorithm. The performance overhead of the proposed technique with respect to the unhardened version ranges from 60% to 18%, and decreases with FFT size. As expected, the recomputation of the whole FFT due to the injected errors nearly doubles the execution time of the ABFT-FFT. The increase in total execution time when compared to the original FFT varies from 4.5 times for a 64-point FFT, to 2.6 times for a 4096-point FFT.

To reduce the recomputation overhead, we devised an extended ABFT-FFT strategy, named Ext ABFT-FFT. Its main idea is that the ABFT algorithm proposed in Sec. IV-A can be repeatedly applied to small portions of the computation, leading to prompt error detection and reduced recomputation costs. Smaller FFTs can be computed by dividing the original problem into small sub-problems. This solution is based on a well-known property of FFTs: a N-point FFT can be decomposed into $N1$ FFTs on $N2$-point and $N2$ FFTs on

TABLE VII
UNHARDENED 1D-FFT AND ABFT-FFT EXECUTION TIMES (IN MS) AS A FUNCTION OF FFT DIMENSION.

	64	256	1024	2048	4096
FFT	0.15	0.67	3.14	6.76	14.53
ABFT-FFT	0.24	0.99	4.27	8.85	19.41
ABFT-FFT w/ injected errors	0.67	1.99	8.64	17.95	37.38

$N1$-point, where $N1 \times N2 = N$. In this sense, we can still use the ABFT-FFT strategy but calculating X' (see Eq. 1) on the smaller FFTs that compose the sub-problems. Each sub-problem can be solved either recursively applying the Ext ABFT-FFT algorithm or directly using ABFT-FFT.

The main advantage of such division is that errors can be detected on a sub-problem, and only the corrupted sub-problem needs to be recomputed in the event of an error. When all sub-problems have been correctly computed, the extended ABFT-FFT strategy is completed by decoding the FFT using Eq. 2 and performing a final error verification.

The Ext ABFT-FFT algorithm is then modular, as its formal correctness is independent on the size of the sub-problems. The smaller the sub-problems are, the more promptly errors will be detected, and the lower the recomputation overhead will be. However, checksums have to be calculated for each sub-problem, so the higher the number of sub-problems, the higher the overhead for checksums will be. A user could select the best trade-off between the overhead required for recomputation in a coarse-grained version and the overhead introduced in the checksum evaluation and error detection on a fine-grained version.

It is worth noting that Ext ABFT-FFT recomputes the whole FFT only if an error occurs while preparing the sub-problem inputs or while computing Eq. 2, which is very unlikely to happen. In fact, there are only $O(N)$ critical operations that can induce the whole recomputation. In contrast, in the standard ABFT-FFT all the $O(N \log N)$ operations can induce the whole recomputation.

V. CONCLUSIONS

The Fast Fourier Transform is an algorithm suited to GPUs that performs important computations for many different applications. Unfortunately, its execution in GPUs is very prone to experience neutron-induced errors. As shown by our experimental evaluations, the characteristic data dependency of the algorithm lets errors spread, which generates a significant amount of multiple output errors.

Although the available ECC provides a way to reduce failure rates, it reduces both the memory available for the application and its performance. Additionally, it does not guarantee a high reliability, as scheduler failures and functional interruptions remain undetected.

We propose an alternative to ECC using the Algorithm-Based Fault Tolerance technique based on a known fault-free FFT network. Our technique proved to be able to improve failure rates by one order of magnitude over the use of ECC, and two orders of magnitude over the original FFT algorithm. These improvements were achieved with an 18% execution time overhead, which is significantly smaller than the 50% overhead from ECC.

We also discuss an extension of the ABFT-FFT algorithm using modular error detection and verification, which could reduce the recomputation time of larger N-point FFTs, but would require the user to decide the best trade-off between checksum calculation and recomputation overhead. As future work, we plan to evaluate Ext ABFT-FFT under radiation, and to provide automatic ways to find the best trade-off.

REFERENCES

[1] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[2] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 908–916. [Online]. Available: http://doi.acm.org/10.1145/1201775.882363

[3] J. Dongarra, H. Meuer, and E. Strohmaier, "TOP500 Supercomputer Sites: November 2013," 2013. [Online]. Available: http://www.top500.org

[4] P. Rech, C. Aguiar, R. Ferreira, M. Silvestri, A. Griffoni, C. Frost, and L. Carro, "Neutron-Induced Soft Errors in Graphic Processing Units," in *Radiation Effects Data Workshop (REDW), 2012 IEEE*, 2012, pp. 1–6.

[5] 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.

[6] M. D. Lerner, "Algorithm Based Fault Tolerance in Massively Parallel Systems," Department of Computer Science, Columbia University, Tech. Rep., 1988.

[7] J.-Y. Jou and J. Abraham, "Fault-tolerant FFT networks," *Computers, IEEE Transactions on*, vol. 37, no. 5, pp. 548–561, 1988.

[8] 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.

[9] "NVIDIA Tesla C2050/C2075 GPU Datasheet," 2010.

[10] I. Haque and V. Pande, "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rates in GPGPU," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 691–696.

[11] J. W. Sheaffer, D. P. Luebke, and K. Skadron, "A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors," in *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, ser. GH '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 55–64. [Online]. Available: http://dl.acm.org/citation.cfm?id=1280094.1280104

[12] 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

[13] T. G. Stockham, Jr., "High-speed convolution and correlation," in *Proceedings of the April 26-28, 1966, Spring joint computer conference*, ser. AFIPS '66 (Spring). New York, NY, USA: ACM, 1966, pp. 229–233. [Online]. Available: http://doi.acm.org/10.1145/1464182.1464209

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

[15] NVIDIA, "Tesla C2050 Performance Benchmarks."