Neutron Sensitivity and Hardening Strategies for Fast Fourier Transform on GPUs

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

***Abstract*—In this paper we analyze the neutron sensitivity of GPU devices when executing a Fast Fourier Transform algorithm. The provided experimental results demonstrate that in the majority of cases the output is affected by multiple errors, caused by thread and data dependencies. ECC is experimentally proved not to be sufficient to provide high reliability. Experimental data and analytical studies are employed to design specific software-based hardening strategies, which are validated through fault-injection.**

***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].

Nowadays, every desktop computer, laptop or portable device includes at least one Graphics Processing Unit (GPU), mainly used as a support for the 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, one of the most powerful of current supercomputers, is built using 18,000 GPUs.

We have already demonstrated in [3][4] that radiation-induced errors, including those generated from the terrestrial neutron radiation environment, are one of the major issues for the newest GPU cores reliability. However, only few papers describe possible radiation test methods for extreme parallel systems and fewer analyze the behavior of parallel algorithms in radiation environments.

In this paper we deeply investigate the behavior of a parallel version of the FFT 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 the FFT output is affected by multiple errors. As demonstrated with algorithm and architectural analyzes, multiple errors occur mainly because the FFT computation requires sequential iterations: hence, a thread output may depend on previously executed threads. If in a given iteration a thread is corrupted by radiation, the error is likely to spread over the following iterations leading to multiple output errors.

We experimentally prove that the Error Correction Code (ECC) available in the latest GPUs is not sufficient to ensure by itself a high reliability. In this work we propose two dedicated software-based hardening strategies for the FFT algorithm executed on a GPU, both based on the Algorithm Based Fault Tolerance (ABFT) philosophy [5]. The first hardening strategy we design takes advantage of the FFT properties demonstrated in [6], and applies them in the GPU algorithm to detect faulty executions. We then extend the proposed ABFT approach to achieve prompt error detection and prevent errors propagation. Finally, the computational overhead of the proposed hardening technique is evaluated and their efficiency is proved through fault-injection.

II. Experimental Methodology and Tested Devices

A. Neutron Beam

Radiation experiments were performed at ISIS facility in the Rutherford Appleton Laboratories (RAL) in Didcot, UK [7]. The available neutron flux was of about 5.5×10^4 n/(cm^2·s). The beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra, which was enough to fully and homogenously irradiate the GPU chip without directly affecting nearby board power control circuitry and DDR memories. Nevertheless, even if the beam is collimated, scattering neutron may still wander from the beam spot, thus we periodically check it during experiments to ensure that the DDR content was consistent during our experience, and no error has been observed. It is worth noting that input and output data were stored in the DDR, and no cache memory was employed, so the errors reported in the following sections were only caused by the corruption of the GPU core logic

P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro are with the Instituto de Informática, Federal University of Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500 - Campus do Vale - Bloco IV, Porto Alegre, RS, Brazil (phone : +55 (51) 3308-6806, fax : +55 (51) 3308-7308 email : {prech, llpilla, navaux, carro} @inf.ufrgs.br)

F. Silvestri is with the Dipartimento di Ingegneria dell'Informazione, Padova University, 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 Informatica e Automatica, Politecnico di Torino, Italy (email: matteo.sonzareorda@polito.it)

This work has been partially supported by CAPES, Brazil and the European Commission through the LoRelei project (PIRSES-GA-2011-295231).

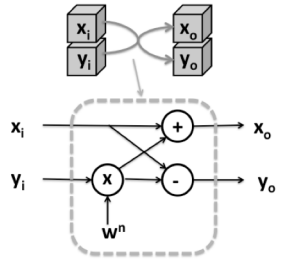

Fig. 1: A basic butterfly module used to update two-by-two all the 64 elements composing the FFT

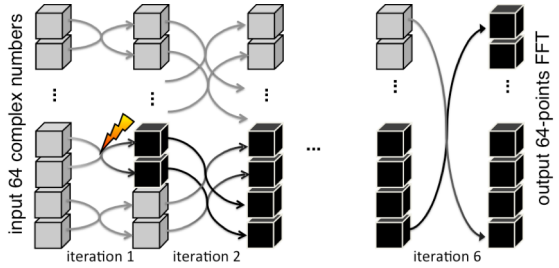

Fig. 2: In each iteration a thread updates two-by-two all the 64 values of the FFT using the basic butterfly module. 6 iterations are necessary to complete execution. If an operation in one iteration is corrupted by radiation, two (or more) values will be wrongly updated, and the number of errors will double in the following iteration.

resources or internal flip-flops. Irradiation was performed at room temperature with normal incidence and nominal voltages.

B. Tested Devices

We tested commercial-off-the-shelves Tesla C2050 GPUs designed by NVIDIA and manufactured in a 40nm technology node. The C2050 includes 14 Streaming Multiprocessors (SMs), each of which is divided in 32 CUDA cores [8]. 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, like the C2050 family, with an internal Error Correction Code (ECC) able to correct single errors and detect double errors, mechanism that can be activated by the user. The ECC was disabled to evaluate the FFT sensitivity to neutrons. A discussion on the efficiency and drawbacks of the NVIDIA ECC mechanism takes place in the following section.

It is worth noting that the delayed blocks input vectors as well as the results of computation are stored in the GPU board DDR, which were not irradiated. On a realistic application, the higher number of blocks may extend the exposure time of input or output data, increasing the probability of having them corrupted by neutrons. However, DDR sensitivity has been proved to decrease with the shrinking of technology nodes [9], and modern DDR chips are provided with efficient ECC that increase of several orders of magnitudes the device reliability [10]. 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 Fast Fourier Transform Code

We tested under radiation a benchmark that implements 512x512 1D-FFTs of 64-points each. The FFT input is composed of a 64x512x512 double precision floating-point matrix for the real part and a 64x512x512 matrix for the imaginary part. We choose to test relatively small FFTs (64-points) to limit the number of iterations and ease the study of errors propagation, while having 512x512 1D-FFTs eases the gathering of a statistically significant amount of errors.

A thread acts like a butterfly module [6] updating the values of two floating-point elements in the complex matrix using the values of two elements computed in the previous iteration as inputs (see Fig. 1). The implemented algorithm is based on the FT kernel of the NAS Parallel Benchmarks [11] implemented in C and ported to the GPU architecture using CUDA. As represented in Fig. 2, each 64-points 1D FFT kernel is composed of 6 sequential iterations ($\log_2 64=6$) of a variant of the Stockham FFT algorithm [12].

For all iterations, the GPU instantiates 512x512 parallel threads, grouped in blocks of 512 threads each. A thread is in charge of evaluating the intermediate FFT values on the assigned complex vector of size 64.

III. Experimental Results and Discussion

Table I reports the experimentally measured cross sections and the FIT for the tested FFT code. The FFT algorithm gives complex double precision floating point data as an output, which is then divided in 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, part of the output. The cross section is obtained dividing the number of faulty executions per unit time by the flux. Reported values confirm that GPUs are extremely prone to be corrupted by neutrons.

Table II shows the percentage of faulty executions of the FFT algorithm in which the real or imaginary part was detected as corrupted as well as the cross section and FIT of just the real and imaginary part. Some executions experienced errors just in the imaginary part or just in the real part even if the implemented algorithm is symmetric for the real and imaginary parts. These errors are caused by the corruption of internal registers used by the thread for storing the intermediate values of the complex number. As stated in the second column of Table II, in less than 5% of the executions considered as faulty no error was detected in the real part. This means that in those executions the FFT experienced an error just in the imaginary part. The same considerations apply to errors in the imaginary part only, which are less than 4% of the overall faulty executions.

The FFT algorithm is composed of threads that are not independent, since a thread uses the output of previously executed threads to update the real and imaginary part of two complex elements (see Fig. 1). 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 (see Fig. 2).

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 calculate the FFT. The 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 a single thread error occurs in the last iteration. As stated in Table III, this occurred in just 1.63% of the faulty executions for the real and in 4% of the faulty executions for the imaginary part.

The importance of the occurrences of multiple errors in realistic applications is highlighted in the last row 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 errors distributions are shown in Fig. 3. It is worth noting that in most of the cases 64 or less output values were found corrupted, and those locations belong to the same 64-point FFT. These errors patterns are caused by error propagation from one iteration to the following ones in the same 64-points FFT, as 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, and this is in agreement with experimental data (see Fig. 3).

The worst case for a 64-points FFT occurs when radiation affects a thread in its first iteration. If a single error is produced in one thread in the first iteration, at each of the following 5 iterations (there are 6 iterations in total) the number of errors is doubled, and 2^5=32 errors appear in the output. A double thread error is produced when radiation prevents the thread from completing its execution generating a functional interruption or corrupting the thread input. In this situation 64 output errors are to be expected in the FFT. It is improbable to have between 32 and 64 errors in the output vector. In fact, as it is very unlikely to have two neutrons corrupting the GPU in a FFT execution, the only way of having more than 32 errors is to have a thread in the first iteration to generate two errors that spread to 64 errors in the output.

Finally, only few executions experienced more than 64 errors in the output. This rare situation occurs when radiation leads a 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, possibly affecting more than one 64-points FFT outputs. Those errors will then spread and a huge amount of errors are expected at the output.

TABLE I
512X512 64-POINTS FFTS CROSS SECTION AND FIT AT NYC

	Cross section	FIT
FFT	$3.69 \cdot 10^{-6} cm^2$	$5.17 \cdot 10^5$

TABLE II
REAL AND IMAGINARY PART PERCENTAGE, CROSS SECTION, AND FIT

	Percentage	Cross section	FIT
FFT Real	94.96%	$3.50 \cdot 10^{-6} cm^2$	$4.90 \cdot 10^5$
FFT Imaginary	96.17%	$3.55 \cdot 10^{-6} cm^2$	$4.97 \cdot 10^5$

TABLE III
512X512 64-POINTS FFTS SINGLE AND MULTIPLE ERRORS

	FFT Real		FFT Imaginary	
	Single	Multiple	Single	Multiple
Percentage	1.61%	98.39%	4.00%	96.00%
FIT	$7.89 \cdot 10^3$	$4.82 \cdot 10^5$	$19.80 \cdot 10^3$	$4.67 \cdot 10^5$

IV. HARDENING STRATEGIES FOR N-POINT FFTS

A. NVDIA 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 [13].

When the ECC is turned ON the 12.5% of the device memory becomes unavailable to the user and, as reported in Tab. IV, the execution time of the 512x512 64-points FFTs is increased of about 50%. Depending on the algorithm and device, typically the ECC reduces the GPU performances in

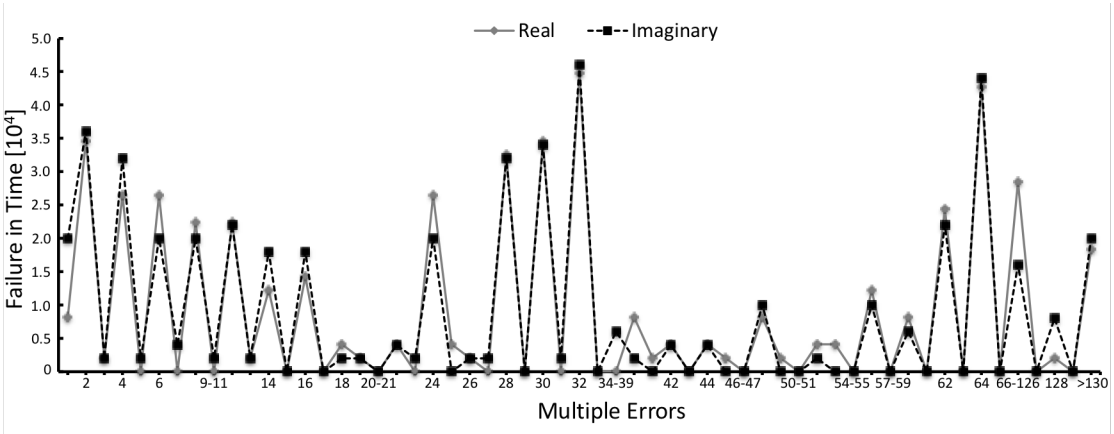

Fig. 3: FFT real and imaginary multiple output errors FIT. Consequent distributions that were never experimentally observed are grouped in the picture (it is the case of 9 to 11 errors, 20 and 21 etc.).

TABLE IV
FFT EXECUTION TIMES

	ECC OFF	ECC ON	Overhead
512x512 64-points FFT	106 ms	159 ms	50%

the range of 20-30% [13]. The computational and area overhead introduced by the NVIDIA ECC are then far from being negligible and may compromise the GPU efficiency.

Unfortunately, no detailed information about the implementation of the ECC is currently available. The analysis of the ECC efficiency and drawbacks is then limited to what was experimentally observed.

When the 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. This is mainly because SM functional interruptions and scheduler failures that prevent threads from completing execution are not detected by the ECC.

B. Algorithm-Based Fault Tolerance for FFT

In order to achieve higher error detection capabilities we design an ABFT technique for the FFT. The proposed hardening strategy derives from a clever fault-free N-points FFT network of N processors presented in [6], 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 induced in any processor or connection with the use of input coding and checksum comparison at the output.

We implement the fault-free network in software in the GPU viewing each thread in the GPU as a processor in the network (i.e., a butterfly module). 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 assigned to it. Threads that follow the corrupted one or threads assigned to computing units near the faulty one will not be affected. This maintains the same set of premises of [6], and hence the same mathematical demonstration ensuring the correctness and efficacy of the approach can be applied.

Only few code modifications are needed to implement the ABFT for FFT (see Fig. 4). The input sequence of N complex elements *x(i)* is encoded in the sequence, *x'(i)*, defined by Eq.1

$$x'(\mathrm{i}) = \begin{cases} 2 \cdot x(i) + x(\mathrm{i}+1) & 0 \le i < N-1 \\ 2 \cdot x(i) + x(0) & i = N-1 \end{cases} \quad (1)$$

The FFT is then evaluated using the coded values *x'* as inputs and, when calculation is completed, the output *X'* is decoded through Eq. 2

$$X(k) = \frac{1}{2+w_N^{-k}} \cdot X'(k) \qquad 0 \le k < N \quad (2)$$

where w_N^{-k} are the N^{th} roots of the unity. The N decoded results are then summed, generating a checksum. As formally

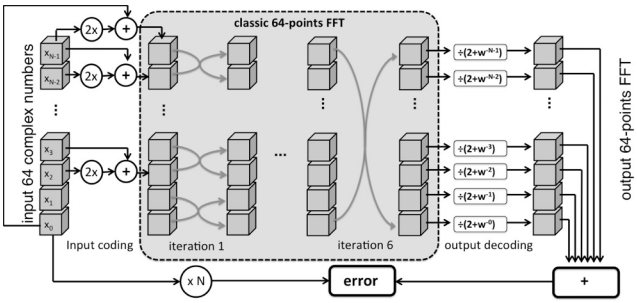

Fig. 4: FFT Hardening scheme. The 64 input complex elements are coded, then the 64-points FFT is performed with the classical algorithm, and output is decoded. Errors are detected comparing the checksum generated summing the output values with Nx_0.

demonstrated in [6] 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·x(0)* and any mismatch will identify the FFT as faulty, and will require re-computation.

Tab. V reports the execution time of the hardened version of the FFT, named ABFT-FFT, running on the GPU with ECC disabled. As it can be noticed, the overhead is comparable to the one introduced by the ECC. However, the memory overhead introduced in the ABFT-FFT is limited to checksums to compare: hence it is constant and definitely lower than the ECC one.

In order to inject errors during the execution of the ABFT-FFT we instantiate an additional thread in the code, independent from the ones composing the ABFT-FFT algorithm, which modifies the 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. All the injected errors, including SM and threads functional interruptions, were detected using the designed hardening strategy and the re-computation of the FFT marked as faulty was performed. As reported in Table V, when errors occur, the execution time of the ABFT-FFT is actually not increased. This happens because the overhead of re-computing just one 1D-FFT is hidden by the execution of concurrent threads executing the remaining (512x512)-1 FFTs. The same happens when forcing errors in all threads in a given block, re-computing some 1D-FFTs.

C. Extended Algorithm-Based Fault Tolerance for FFT

As stated in previous sections, when an error occurs in one iteration, it is likely to spread to the following ones: therefore, a prompt detection of the error is crucial to prevent error propagation. The ABFT-FFT hardening strategy has the ability of detecting all the experimentally observed error patterns, but requires re-computation in order to provide the correct output, introducing a not negligible overhead. It is worth noting that bigger FFTs than the tested ones are typically executed on GPUs. In Tab. V the execution time of a single FFT of various dimensions are reported, for the un-hardened version and for the proposed ABFT-FFT. As it can be noticed, in the event of errors the FFT re-computation drastically affects the ABFT-FFT performances. In the case of

TABLE V
512X512 64-POINTS FFT AND ABFT-FFT EXECUTION TIMES

	FFT	ABFT	overhead
no error	106 ms	161ms	55%
errors injected	106 ms	161ms	55%

TABLE VI
1-D FFT AND FFT-ABFT EXECUTION TIMES AS A FUNCTION OF FFT DIMENSION

	64	256	1024	2048	4096
FFT	0.15ms	0.67ms	3.14ms	6.76ms	14.53ms
ABFT-no errors	0.24ms	0.99ms	4.27ms	8.85ms	19.41ms
ABFT-errors	4.67ms	1.99ms	8.64ms	17.95ms	37.38ms

a single 4096-points FFT, if an error occurs the ABFT-FFT execution time is 2.5 times higher than the unhardened code one (last column of Tab. VI).

To reduce the re-computation overhead we devised an extended ABFT-FFT strategy, named Ext ABFT-FFT. The basic idea is that the ABFT proposed in the previous sub-section can be repeatedly applied to small portions of the computation, leading to prompt error detections and reduced re-computation costs. More in details, we can compute smaller FFTs diving the original problem into smaller sub-problems, using the well-known propriety that a N-point FFT can be decomposed into N1 FFTs on N2 points and N2 FFTs on N1 points, where $N1 \cdot N2 = N$. Then, 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 application or using directly the ABFT-FFT.

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

The Ext ABFT-FFT is then modular, as its formal correctness is independent on the size of the sub-problems. The smaller the sub-problems are, the more prompt errors detection will be and the lower the re-computation 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. The user can select the best trade-off between the overhead required for re-computation in a coarse-grained version and the overhead introduced in the checksum evaluations on a fine-grained version.

It is worth noting that the Ext ABFT-FFT re-computes 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 re-computation. In contrast, in the standard ABFT-FFT all the $O(NlogN)$ operations can induce the whole re-computation.

V. CONCLUSIONS

The FFT algorithm executed by GPUs is a powerful tool for many applications but, unfortunately, it is very prone to experience neutron-induced errors. The FFT characteristic thread and data dependency of the algorithm let the errors to spread, generating a huge amount of multiple output errors.

To increase the reliability of GPUs, NVIDIA introduced an ECC, which reduces errors but also reduces both the memory availability and the performances of the device. An alternative to the ECC is the Algorithm-Based Fault Tolerance technique we implemented on the GPU, which is based on a known fault-free FFT network, that was demonstrated through fault-simulation to be able to detect all the experimentally observed errors. We extended this strategy making it modular, so that the user can decide the best trade-off between checksum calculation and re-computation overheads.

REFERENCES

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

[2] J. Kruger and R. Westermann, "Linear Algebra operators for GPU implementation of numerical algorithms", ACM Trans. Graph. n. 22, vol. 3, 2003, pp. 908-916.

[3] P. Rech, C. Aguiar, R. Ferreira, M. Silvestri, A. Griffoni, C. Frost, and L. Carro, "Neutron-Induced Soft Error in Graphic Processing Units", in proc. IEEE REDW 2012, Miami, FL, USA.

[4] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs", IEEE Trans. Nucl. Sci, n. 4, vol 60, 2013, pp. 2797-2804.

[5] M.D. Lerner, "Algorithm Based Fault Tolerance in Massively Parallel Systems", Columbia University, 1988

[6] J.-Y. Jou, J.A. Abraham, "Fault-Tolerant FFT Networks", IEEE Transactions on Computers, Vol. 37, No. 5, May 1988, pp. 548-561

[7] M. Violante, et al., "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility", IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 1184-1189

[8] NVIDIA Tesla C2050/C2075 GPU Datasheet.

[9] I. S. Haque and V. S. Pande, "Hard Data on Soft Errors: A Large- Scale Assessment of Real-World Error Rates in GPGPU," In Proceedings of the IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 691-696, 2010

[10] 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 ACM SIGGRAPH Symposium on Graphics Hardware (GH), pp. 55-64, 2007

[11] D. Bailey, et al., "The NAS Parallel Benchmarks", RNR Technical Report RNR-94-007, March 1994.

[12] T. G. Stockham, "High-Speed Convolution and Correlation", in proc. Spring Joint Computer Conference, 1966, pp. 229-233.

[13] NVIDIA BENCH: Tesla C2050 Performance Benchmarks