Efficacy and Efficiency of Algorithm-Based Fault Tolerance on GPUs

Wunderlich, Hans-Joachim; Braun, Claus; Halder, Sebastian

Proceedings of the IEEE International On-Line Testing Symposium (IOLTS'13) Crete, Greece, 8-10 July 2013

doi: http://dx.doi.org/10.1109/IOLTS.2013.6604090

Abstract: Computer simulations drive innovations in science and industry, and they are gaining more and more importance. However, their high computational demand generates extraordinary challenges for computing systems. Typical highperformance computing systems, which provide sufficient performance and high reliability, are extremly expensive. Modern GPUs offer high performance at very low costs, and they enable simulation applications on the desktop. However, they are increasingly prone to transient effects and other reliability threats. To fulfill the strict reliability requirements in scientific computing and simulation technology, appropriate fault tolerance measures have to be integrated into simulation applications for GPUs. Algorithm-Based Fault Tolerance on GPUs has the potential to meet these requirements. In this work we investigate the efficiency and the efficacy of ABFT for matrix operations on GPUs. We compare ABFT against fault tolerance schemes that are based on redundant computations and we evaluate its error detection capabilities

Preprint

Efficacy and Efficiency of Algorithm-Based Fault-Tolerance on GPUs

Hans-Joachim Wunderlich, Claus Braun, Sebastian Halder
Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569, Germany, Email: {wu,braun}@informatik.uni-stuttgart.de

***Abstract*—Computer simulations drive innovations in science and industry, and they are gaining more and more importance. However, their high computational demand generates extraordinary challenges for computing systems. Typical high-performance computing systems, which provide sufficient performance and high reliability, are extremly expensive.**

Modern GPUs offer high performance at very low costs, and they enable simulation applications on the desktop. However, they are increasingly prone to transient effects and other reliability threats. To fulfill the strict reliability requirements in scientific computing and simulation technology, appropriate fault tolerance measures have to be integrated into simulation applications for GPUs. *Algorithm-Based Fault Tolerance on GPUs* has the potential to meet these requirements.

In this work we investigate the efficiency and the efficacy of ABFT for matrix operations on GPUs. We compare ABFT against fault tolerance schemes that are based on redundant computations and we evaluate its error detection capabilities

***Keywords*-Scientific Computing, GPGPU, Soft Errors, Fault Simulation, Algorithm-based Fault Tolerance**

I. Introduction

Simulation technology and scientific computing are two of the key forces that drive innovation in science and industry. International research efforts, like the *Stuttgart Research Center for Simulation Technology*[1] (SimTech) or *Caltech's Center for Advanced Computing Research*[2] (CACR), reflect the growing importance of these research areas. Sophisticated simulation applications, which are often dominated by compute-intensive mathematical tasks, like linear algebra matrix operations, generate extraordinary challenges for contemporary computing systems. Typical high-performance computing systems, which deliver sufficient performance and high hardware reliability, are extremely expensive. However, the extensive architectural development of graphics processing units (GPUs) within the last decade brought far-reaching changes and the opportunity to solve many of these challenges on the desktop. Modern GPUs left the niche of visual data processing and emerged to fully programmable many-core processor architectures. They exploit massive on-chip parallelism and throughput optimization to deliver tremendous floating-point computing performance at very low costs. This enables simulations from very different domains like EDA [1, 2], biology [3], or thermodynamics [4], with significant speedups in the order of several magnitudes.

[1]http://www.simtech.uni-stuttgart.de/
[2]http://http://www.cacr.caltech.edu

The acronym *GPGPU* (General-Purpose Computations on Graphics Processing Units) [5] has been coined for the use of GPUs in such compute-intensive, non-graphical applications.

However, being manufactured in latest generation semiconductor technology nodes of 28 nm and below, GPUs are increasingly prone to transient effects, latent defects, and different aging mechanisms. Although the impacts of these reliability threats might be tolerable to a certain extent in graphical applications, they may present a serious problem for applications in scientific computing and simulation technology. Here, the reliability requirements are very high and must be strictly adhered to.

- Extensive simulation runtimes and the associated costs prohibit partial or even complete recomputations in case of errors.
- Scientific applications mainly rely on floating-point arithmetic and they demand stable and highly accurate results.
- Scientific simulations are typically tuned for high efficiency and maximum performance. Hence, there is only small room for the integration of fault tolerance schemes.
- Potentially long chains of operations that are performed on the data sets favor spreading of errors. Such spreading errors and also silent data corruptions are absolutely intolerable and demand early error detection and correction.

Obviously, these requirements have strong implications on potential fault tolerance measures.

Algorithm-Based Fault Tolerance (ABFT) is a software fault tolerance technique that has the potential to meet the high reliability requirements in scientific computing and simulation technology. The basic idea of ABFT is to utilize information redundancy by modifying algorithms to work on encoded input data and to produce encoded results. In [6] the ABFT approach is introduced for matrix operations like multiplications, additions and LU decompositions. Here, the rows and columns of input matrices are summed up to form row- and column-checksums that are attached to the matrices as additional columns or rows, respectively. After the target operation has been performed, the result is a so-called full-checksum matrix, whose checksum rows and columns are used to detect errors. The moderate performance overhead and the good error detection capabilities make ABFT schemes particularly interesting for integration into

simulation applications on GPU many-core architectures.

In this work we investigate the efficacy and efficiency of *Algorithm-Based Fault Tolerance for GPU architectures* and floating-point arithmetic. We compare the performance of ABFT for matrix multiplications against established approaches that are based on redundant computations. Furthermore, we evaluate the error detection efficacy and the inherent fault tolerance potential of floating-point arithmetic for optimal error threshold determination.

II. Related Work

Over the years, several approaches for ABFT schemes on GPUs have been introduced, including [7–10]. Other works propose the integration of *duplication with comparison* (DWC) or *triple modular redundancy* (TMR). Both schemes can be realized at different levels of granularity, from single instructions that are executed multiple times up to threads or complete kernel calls that are duplicated or triplicated. Although DWC is able to detect errors, it halves the payload throughput and it requires a full repetition of the performed operation in case of an error. TMR has the advantage of providing a correct result in case of a successful majority vote. However, it effectively reduces the payload throughput by 66%, and hence causes a significant performance penalty, especially for large data sets. In [11] the authors proposed the execution of redundant instructions by utilizing unused instruction level parallelism on GPUs. Inspired by *Simultaneous Redundant Multithreading*, the authors in [12] proposed the execution of redundant threads to detect errors. Both approaches work well for older generations of GPU architectures, where only one kernel could be executed at a time and the GPU was allocated by a single user. For modern GPUs, which allow the parallel execution of multiple different kernels and which comprise much more advanced hardware thread schedulers, the number of "free slots" for redundant instructions or threads is significantly reduced. Hence, the benefit of theses schemes diminishes. *Checkpointing and restart* is also a well established technique to improve the reliability in scientific computing, typically targeting large-scale clusters or workstation grids. In [13] checkpointing protocols for GPU/CPU systems are introduced, which use GPU virtualization and overlapping memory copy and kernel execution (Nvidia Streams). Compared to DWC, TMR and ABFT, such checkpointing approaches are rather coarse-grained and involve significant data transfer between the GPU and the CPU.

III. GPU Programming Challenges

Although modern GPUs provide a very impressive computational performance, their architectures often demand careful and insightful programming. Some of the points that have to be kept in mind represent major challenges, especially with respect to the integration of software-based fault tolerance.

- Whenever possible, parallel threads on GPUs should follow a common control flow, since diverging threads typically result in a serialized execution.
- The register files of GPUs comprise several thousands of registers and all of these registers have to be shared among all active threads. Hence, excessive per-thread register usage reduces the number of concurrently executed threads significantly and should be avoided.
- Memory bandwidth on GPUs is a scarce ressource and typical access times to the device memories are in the range of several hundreds of clock cycles. This forces bundled and well coalesced memory access patterns.
- Wherever possible, the small shared memories of GPUs have to be utilized for caching and data re-use among the threads.
- Inter-thread communication across the GPU and global barrier synchronization are very expensive and should be avoided.

Even more than for standard GPU application development, these aspects have to be kept in mind when software- or algorithm-based fault tolerance mechanisms have to be integrated with minimized performance impact.

IV. Efficiency of ABFT on GPUs

To achieve maximum performance for ABFT-protected matrix multiplications on GPUs, the multiplication itself has to be very efficient, since it will dominate the runtime. Moreover, the required steps for encoding and checking of the produced results have to be designed in a way, that they utilize the hardware characteristics of modern GPUs.

The realized ABFT matrix multiplication uses a partitioned encoding similar to [14], and a flexible matrix data structure, which allows the handling of matrices in different encoded and non-encoded states. The partitioned encoding matches the block-based computation approach of GPUs perfectly and it improves the error detection capabilities, since the checksums are computed over smaller submatrices and not over the possibly very large input matrices.

The required computational steps are separated into three different kernels:

- Kernel #1 handles the encoding of the input matrices and the computation of the reference checksums for the check of the final result. Given a $n \times n$ matrix and a block size of b, the kernel executes $n \cdot b$ parallel threads to compute the row checksums and $n \cdot b$ parallel threads to compute the column checksums. Since every matrix element is read only once during this step, additional caching in the shared memories gives no advantage.
- Kernel #2 performs a block-based matrix multiplication and utilizes the shared memories to achieve high performance. The block size b is chosen to match the thread block size. Before the actual computation starts, the elements of the input matrices are loaded into

the shared memories to reduce the number of global memory accesses.
- Kernel #3 handles the check of the final result and marks erroneous results for subsequent correction. The computation of the required reference checksums for the comparison is performed analogous to Kernel #1. In case of detected errors, the corresponding blocks (sub-matrices) are marked for subsequent error correction.

The separation of the computational steps into different GPU kernels has several advantages. First, each kernel can be optimally parametrized for the task it performs. This means that we can utilize a maximum number of threads for the checksum encoding, the multiplication and the checking. Second, the encoding and checking kernels can be called outside of the actual multiplication to verify the matrix data in memory. This can be very useful on GPUs without ECC-protected memory. Third, the encoding and checking kernels can be used for other ABFT-protected matrix operations on the GPU, like matrix additions or LU decompositions.

The realized ABFT matrix multiplication has been evaluated over different matrix dimensions and the runtimes have been compared to the unprotected version of the matrix multiplication and to a TMR approach. The TMR version computes the matrix multiplication three times and compares the results. As baseline, the same computations have been performed on a single core of the host CPU.

Matrix Dim.	CPU time in s	GPU unprot. time in s	GPU TMR time in s	GPU ABFT time in s
32^2	8.5	0.04	0.06	0.09
64^2	29.5	0.04	0.06	0.09
128^2	102.5	0.04	0.09	0.18
256^2	131.5	0.15	0.44	0.45
512^2	1232	1.1	4.3	1.6
1024^2	9731.5	8.4	25.2	11.3
2048^2	75287.5	70.9	200.9	78.9
3072^2	328839	229.9	684.5	258.6
4096^2	2184289.5	554.6	1633.8	619.9

Table I
PERFORMANCE COMPARISON BETWEEN THE UNPROTECTED GPU MATRIX MULTIPLICATION, THE TMR VARIANT, AND ABFT.

The results in Table I show that the runtime overhead for small matrices is comparable between the TMR approach and the ABFT matrix multiplication. This is mainly due to the kernel invocation overhead. However, for more interesting matrix dimensions beginning with 512×512, the low performance overhead of the ABFT kernels becomes visible. Besides the computational overhead of the TMR scheme, its memory requirements are much worse, compared to ABFT, since all matrices have to be kept in memory three times.

V. EFFICACY OF ABFT ON GPUS

The efficacy of ABFT strongly depends on the successful distinction between critical soft errors that affect the final result, and those soft errors, whose erroneous values reside within the roundoff margin. The first group has to be detected and corrected in any case to provide reliable results. The second group does not harm the result, but it affects the performance negatively. Since we assume that the error detection thresholds are defined by the target application's accuracy level, ABFT is able to deliver full error coverage, with respect to these requirements. However, to provide effective and efficient ABFT schemes, the fraction of false positives has to be reduced.

The assessment and optimization of ABFT schemes requires detailed information on the soft error vulnerability of the underlying hardware. One option for obtaining such data are radiation experiments like those that have been successfully performed for GPUs in [10]. However, such statistical experiments and functional evaluations are not able to exhaustively characterize the impact of single event effects with respect to the structure of the underlying hardware.

To solve this problem, we developed a general structural hardware model and an integrated multi-level simulation environment. The hardware model is independent of vendor-specific hardware information, it provides transferable soft error data, and it offers large flexibility for the adaption to new hardware structures. The simulation environment utilizes gate-level logic simulation and fault injections to gain detailed soft error data, and it combines this data with the native execution of ABFT schemes. The hardware basis for the model is a synthesized netlist of an industry-grade floating-point unit (FPU), which fully complies with the IEEE-754 standard. The fault injections are performed exhaustively, which means that we inject line flips into *each gate of the FPU* and not only into the state elements. Given that the structural model of the FPU consists of ca. 22.000 logic gates, 22.000 fault injections are performed for the characterization of a single operation. Since the simulation environment is highly optimized and parallelized, such a full characterization takes only a few seconds. The injection time and the duration can be arbitrarily chosen. The simulation framework allows the tracking of single clock cycles during the execution of a floating-point operation, and it delivers detailed data on the quantitative and qualitative outcomes of the injected faults.

We distinguish between the following errors:

- *Masked errors* are injected faults that do not cause any change in the circuits state or the result value.
- *Visible errors* are injected faults that cause either the state of the circuit, the computed result or both to be faulty.
- *Value errors* are those visible errors that cause the final result to be erroneous. These are the errors that are targeted by the ABFT scheme.

Table II shows the results of a soft error characterization of the basic floating-point operations FADD, FSUB, FMUL, and FDIV. The fraction of masked errors and visible errors are given, as well as the fraction of total value errors.

FP Op.	Masked Errors (%)	Visible Errors (%)	Total Value Errors (%)
FADD	67.8	32.2	**14.5**
FSUB	67.8	32.2	**14.5**
FMUL	63.2	36.8	**24.3**
FDIV	72.5	27.5	**24.5**

Table II
SOFT ERROR CHARACTERIZATION OF FLOATING-POINT OPERATIONS.

DWC and TMR have the advantage of fast result checking at bit-level. Since the same computations are performed with the same input data on the same hardware, the results have to match exactly in the error free case. However, in a further experiment we were able to show, that there exist non-negligible fractions of visible soft errors, which do not affect the final result of a computation, but which would cause DWC and TMR to trigger a recomputation. Here, the ABFT scheme is able to classify these soft errors correctly and hence to ignore them. To determine the exact fractions, we performed 78126 evaluations of the ABFT matrix multiplication, where a floating-point operation was randomly chosen for fault injection. Table III shows the results of this campaign. The error detection thresholds have been determined beforehand.

Matrix Dim.	Error Det. Threshold	Visible Soft Errors below Err. Det. Threshold(%)
4x4	1.78e-15	**6.64**
6x6	3.60e-15	**7.88**
8x8	7.50e-15	**8.97**
16x16	4.30e-14	**10.56**
24x24	1.15e-13	**11.40**
32x32	2.30e-13	**11.86**

Table III
SOFT ERRORS BELOW THE ERROR DETECTION THRESHOLD THAT WOULD CAUSE DWC/TMR TO TRIGGER RECOMPUTATION.

Given the results of the performance evaluation, fractions of 6.64% to 11.86% soft errors that would cause DWC or TMR to replay, render these techniques highly unattractive and demonstrate the efficiency of ABFT.

VI. CONCLUSION

In this work we investigated the efficiency and the efficacy of *Algorithm-Based Fault Tolerance on modern GPUs*. We evaluated an ABFT matrix multiplication with respect to the performance overhead and we compared it to TMR. For the evaluation of the ABFT error detection efficacy, we presented an integrated simulation environment and a structural hardware model, which allows the exhaustive soft error characterization of floating-point arithmetic operations. The obtained soft error data has been used to evaluate the ABFT matrix multiplication.

The results show, that ABFT can be integrated into scientific computations on GPUs with low performance overhead, and that it provides very good error detection capabilities. In contrast to schemes like DWC and TMR, ABFT is also able to utilize roundoff effects, to avoid unnecessary error corrections.

VII. ACKNOWLEDGMENT

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart

REFERENCES

[1] M. A. Kochte *et al.*, "Efficient Fault Simulation on Many-core Processors", in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC'10)*. ACM, 2010, pp. 380–385.

[2] S. Holst, E. Schneider, and H.-J. Wunderlich, "Scan Test Power Simulation on GPGPUs", in *Proceedings of the 21st IEEE Asian Test Symposium (ATS12)*. IEEE Computer Society, 2012, pp. 155–160.

[3] C. Braun *et al.*, "Parallel Simulation of Apoptotic Receptor-Clustering on GPGPU Many-Core Architectures", in *IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2012*, oct. 2012, pp. 1–6.

[4] C. Braun *et al.*, "Acceleration of Monte-Carlo Molecular Simulations on Hybrid Computing Architectures", in *IEEE 30th International Conference on Computer Design (ICCD), 2012*, 30 2012-oct. 3 2012, pp. 207–212.

[5] J. Nickolls and W. Dally, "The GPU Computing Era", *IEEE Micro*, vol. 30, no. 2, pp. 56–69, march-april 2010.

[6] K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations", *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518 –528, June 1984.

[7] C. Braun and H.-J. Wunderlich, "Algorithm-Based Fault Tolerance for Many-Core Architectures", in *Proceedings of the 15th IEEE European Test Symposium (ETS'10)*. IEEE Computer Society, 2010, pp. 253–253.

[8] C. Braun and H.-J. Wunderlich, "Algorithmen-basierte fehlertoleranz für many-core-architekturen; algorithm-based fault-tolerance on many-core architectures", *it - Information Technology*, vol. 52, no. 4, pp. 209–215, 2010. [Online]. Available: http://www.it-information-technology.de/

[9] C. Ding *et al.*, "Matrix multiplication on gpus with on-line fault tolerance", in *9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'11)*, 2011, pp. 311–317.

[10] P. Rech *et al.*, "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs", *IEEE Transactions on Nuclear Science*, vol. XX, no. 99, p. Accepted for publication, 2013.

[11] M. Dimitrov, M. Mantor, and H. Zhou, "Understanding Software Approaches for GPGPU Reliability", in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, ser. GPGPU-2. New York, NY, USA: ACM, 2009, pp. 94–104. [Online]. Available: http://doi.acm.org/10.1145/1513895.1513907

[12] J. Backer and R. Karri, "Balancing Performance and Fault Detection for GPGPU Workloads", in *30th IEEE International Conference on Computer Design (ICCD'12)*, 2012, pp. 518–519.

[13] S. Laosooksathit, N. Naksinehaboon, and C. Leangsuksan, "Two-level checkpoint/restart modeling for gpgpu", in *Computer Systems and Applications (AICCSA), 2011 9th IEEE/ACS International Conference on*, 2011, pp. 276–283.

[14] J. Rexford and N. Jha, "Algorithm-Based Fault Tolerance for Floating-Point Operations in Massively Parallel Systems", in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS '92)*, vol. 2, 1992, pp. 649–652.