Low-Overhead Fault-Tolerance for the Preconditioned Conjugate Gradient Solver

Schöll, Alexander; Braun, Claus; Kochte, Michael A.; Wunderlich, Hans-Joachim

Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'15) Amherst, Massachusetts, USA, 12-14 October 2015

doi: http://dx.doi.org/10.1109/DFT.2015.7315136

Abstract: Linear system solvers are an integral part for many different compute-intensive applications and they benefit from the compute power of heterogeneous computer architectures. However, the growing spectrum of reliability threats for such nano-scaled CMOS devices makes the integration of fault tolerance mandatory. The preconditioned conjugate gradient (PCG) method is one widely used solver as it finds solutions typically faster compared to direct methods. Although this iterative approach is able to tolerate certain errors, latest research shows that the PCG solver is still vulnerable to transient effects. Even single errors, for instance, caused by marginal hardware, harsh environments, or particle radiation, can considerably affect execution times, or lead to silent data corruption. In this work, a novel fault-tolerant PCG solver with extremely low runtime overhead is proposed. Since the error detection method does not involve expensive operations, it scales very well with increasing problem sizes. In case of errors, the method selects between three different correction methods according to the identified error. Experimental results show a runtime overhead for error detection ranging only from 0.04% to 1.70%.

Preprint

Low-Overhead Fault-Tolerance for the Preconditioned Conjugate Gradient Solver

Alexander Schöll, Claus Braun, Michael A. Kochte and Hans-Joachim Wunderlich

Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569, Germany, Email: {schoell,braun,kochte,wu}@informatik.uni-stuttgart.de

***Abstract*—Linear system solvers are an integral part for many different compute-intensive applications and they benefit from the compute power of heterogeneous computer architectures. However, the growing spectrum of reliability threats for such nano-scaled CMOS devices makes the integration of fault tolerance mandatory. The preconditioned conjugate gradient (PCG) method is one widely used solver as it finds solutions typically faster compared to direct methods. Although this iterative approach is able to tolerate certain errors, latest research shows that the PCG solver is still vulnerable to transient effects. Even single errors, for instance, caused by marginal hardware, harsh environments, or particle radiation, can considerably affect execution times, or lead to silent data corruption. In this work, a novel fault-tolerant PCG solver with extremely low runtime overhead is proposed. Since the error detection method does not involve expensive operations, it scales very well with increasing problem sizes. In case of errors, the method selects between three different correction methods according to the identified error. Experimental results show a runtime overhead for error detection ranging only from 0.04% to 1.70%.**

***Keywords*-Fault Tolerance, Sparse Linear System Solving, Preconditioned Conjugate Gradient, ABFT**

I. Introduction

Linear systems play an essential role in many large-scale applications in science and engineering [1], including structural mechanics [2], computational fluid dynamics [3], or power grid analysis [4]. For this reason, the efficient solution of linear systems is an integral computational task in high-performance computing. The *preconditioned conjugate gradient* (PCG) solver [5] is one of the most important methods for solving large linear systems of the form $Ax = b$. Compared to direct methods like e.g. the Gaussian-Elimination, PCG finds a solution typically faster. At the same time, the underlying operations are parallelizable, which makes the PCG solver well suited for heterogeneous computing systems comprising multi-core CPUs and many-core GPUs. Recent works in this area gain significant speedups [6, 7] by exploiting different characteristics of the underlying linear algebra operations.

However, such modern nano-scaled CMOS devices become increasingly vulnerable to a growing spectrum of reliability threats such as transient events, latent defects, stress and aging mechanisms as well as marginal hardware due to process variations [8, 9]. Future manufacturing processes will allow even smaller chip feature sizes resulting in an increased vulnerability, which demands the application of fault tolerance measures [10]. Compared to direct methods, iterative solvers such as PCG exhibit an inherent robustness against transient effects causing soft errors [5]. However, iterative methods are still considerably vulnerable to soft errors as recent works [11, 12] shows. Single errors may lead to silent data corruption and are able to degrade the performance of PCG by factors of up to 200x. In case of silent data corruption (SDC), the derived solution may not satisfy the original problem $Ax = b$, despite apparent convergence.

An approach to detect errors is to evaluate the computed solution x using the original problem $Ax = b$ after the completion of PCG. However, the execution of PCG has to be repeated in case of errors, until a correct result is obtained. Without a fault-tolerance technique, which already detects errors during the execution, possible errors remain undetected until the end of the execution. Different fault tolerance approaches were proposed that tackle the vulnerability of PCG to different extents. These are discussed in Section III. Some of them limit the fault detection only to certain subsets of the PCG operations. A majority of them induce significant runtime overheads since they utilize additional expensive matrix-vector multiplications to detect errors. Besides, some approaches use traditional checkpointing techniques, which induce high cost in recomputing erroneous results compared to immediate correction approaches [13].

In [14], we proposed a fault-tolerance technique with reduced computational complexity compared to the existing methods for error detection (cf. Section IV). Here, we propose both an improved error detection criterion and a new error recovery scheme, which allow a significant reduction in runtime overhead while the error coverage is not only maintained, but often improved. In case of errors, the proposed error recovery scheme selects between three different correction methods adaptively. The proposed fault-tolerance technique is evaluated on many-core GPUs as GPU-accelerated scientific computing is gaining popularity while it is increasingly vulnerable to reliability threats [15].

II. Preconditioned Conjugate Gradient Method

The PCG method [5] is an iterative approach, which is suitable for solving large linear systems $Ax = b$, when A is a symmetric positive-definite matrix. The fundamental

operations of the PCG solver are shown in Algorithm 1. The inputs include a coefficient matrix A, a right-hand side vector b, an initial guess vector x_0, a preconditioner M, a tolerance ϵ to accept a sufficient approximation, and an upper limit for the number of iterations k_{max}.

Each successive iteration of the *PCG loop* provides an improved approximation x_k with respect to the exact solution. PCG composes the solution x as a linear combination of *search directions* $p_0, p_1, p_2, ..., p_N$ and $x = x_0 + \sum_{k \leq N} \alpha_k p_k$. In every subsequent iteration, a new search direction p_k is computed from the residual r_k such that $p_i \perp Ap_k, k \neq i$. Therefore, each residual r_k is orthogonal to both each preceding search direction p_i as well as each residual r_i with $i < k$. The time complexity of PCG depends on both the size and the condition number of the matrix A [5]. A suitable *preconditioner* M is able to diminish the condition number of the matrix A, which improves the rate of convergence. As will be further discussed below, the proposed fault-tolerant PCG solver is independent of the utilized preconditioning operation.

Algorithm 1: Preconditioned Conjugate Gradient

Input: A,M,b, x_0, ϵ, k_{max}
$r_0 \leftarrow b - Ax_0$; // Initial residual
$s_0 \leftarrow M^{-1} r_0$; // Preconditioning
$p_0 \leftarrow s_0$; // Initial search direction
$\delta_0 \leftarrow r_0^T r_0$; // Approximation error
$k \leftarrow 0$; // Iteration count
/* PCG loop */
while $\delta_k > \epsilon \wedge k < k_{max}$ **do**
  $w_k \leftarrow Ap_k$;
  $\gamma \leftarrow r_k^T s_k$;
  $\alpha \leftarrow \frac{\gamma}{p_k^T w_k}$;
  $x_{k+1} \leftarrow x_k + \alpha p_k$; // Improve approximation
  $r_{k+1} \leftarrow r_k - \alpha w_k$; // Update residual
  $s_{k+1} \leftarrow M^{-1} r_{k+1}$; // Preconditioning
  $\delta_{k+1} \leftarrow r_{k+1}^T r_{k+1}$;
  $\beta \leftarrow \frac{r_{k+1}^T s_{k+1}}{\gamma}$;
  $p_{k+1} \leftarrow s_{k+1} + \beta p_k$; // New search direction
  $k \leftarrow k + 1$;
end

III. State of the Art

The investigation of fault tolerance for linear algebra algorithms is an active research area. *Algorithm-Based Fault Tolerance (ABFT)* [16] allows the protection of important linear algebra operations such as matrix multiplication and LU decomposition. ABFT evaluates checksum mismatches between encoded input data and encoded results to detect errors. More sophisticated ABFT methods address the influence of rounding errors in the encoding of checksums [17]. Manifestations of faults that impact, for instance, the control flow or corrupt data in the memory can be treated by low-overhead techniques such as error-detecting/correcting codes, signature monitoring, or assertions [18, 19].

Error detection codes, however, are not able to protect all operations in PCG efficiently such as inner products. Therefore, fault tolerance for PCG demands different methods to achieve complete protection. The vulnerability of PCG was assessed over the last decade: The insufficient ability of PCG to detect errors resulting in silent data corruptions is presented in [11]. The influence of soft errors on the performance of linear solvers is discussed in [12] while a performance degradation of PCG by factors of up to 200x is demonstrated. Besides, different fault tolerance techniques were proposed: The evaluation of checksum invariants during matrix-vector multiplications is proposed in [20] to detect errors. During error-free executions, the reported overhead of this technique is on average 11.3%. In [21], a partial recomputation scheme is presented that localizes and corrects errors during sparse matrix-vector multiplications. In [22], inherent relations between internal values in PCG are exploited for error detection. If an incorrect solution is detected after the completion of PCG, then PCG is repeated on the obtained residual $Ad = r = (b - Ax)$. The method aims to avoid repetitions of PCG on the original problem. However, the technique awaits the result after complete convergence of PCG, before it applies error detection. A self-stabilizing approach which requires system modes that are inherently fault-tolerant is proposed in [23]. A stabilization scheme is presented that exploits the convergence conditions of PCG to transform arbitrary states to valid states. In [24], a technique is proposed that periodically checks the residual vector and the orthogonality of consecutive search direction vectors. In our previous work [14], we presented a fault-tolerance technique for PCG with an reported error detection overhead ranging from 0.1% to 5.4%. That technique outperformed the existing approaches in terms of both runtime overhead as well as error coverage. In this work, we propose an improved fault-tolerance technique for PCG with extremely low runtime overhead. At the same time, the error coverage is not only maintained, but often improved.

In summary, the discussed approaches address fault tolerance for PCG to different degrees. The approaches in [20, 21] limit the error detection to certain subroutines of the PCG solver. The approaches in [23, 24] on the other hand cover complete PCG iterations. However, they exhibit significant overheads to detect errors, since they require expensive sparse matrix-vector multiplications with complexity $\mathcal{O}(NNZ)$ with $NNZ \gg N$. NNZ is the number of non-zero elements in the matrix A with size $N \times N$.

IV. Fault-Tolerant Preconditioned Conjugate Gradient Solver

In this work, an improved *fault-tolerant preconditioned conjugate gradient solver* with extremely low runtime overhead is proposed. Since errors are detected during the execution of the solver, this technique prevents the repetition of complete executions. The proposed error detection scheme is very efficient because it does not require

expensive sparse matrix-vector multiplications. Instead, it is based on the periodic evaluation of additional inner products with complexity $\mathcal{O}(N)$. Therefore, the runtime overhead for error detection is vanishing with increasing matrix sizes. A new *error detection criterion* is presented below, which is referred to as *σ-criterion*. In combination with our previously proposed λ-criterion [14], the complete set of vectors used in PCG is covered for error detection. Therefore, the error detection latency is reduced, which allows to increase the *error checking interval*, i.e. the number of PCG iterations after which error detection is performed, without loss in error coverage (cf. Section VII). Thus, the runtime overhead is reduced. Besides, our error correction technique is complemented by a new *corrective roll-back recovery* technique. Now, our proposed *adaptive error correction method* identifies the degree of corruption and trades off three different correction methods against each other. Therefore, the proposed fault-tolerance method provides efficient error detection and reduces the number of expensive recomputations to correct corrupted results. Figure 1 shows the steps of our *fault-tolerant preconditioned conjugate gradient solver*. The first two steps comprise the *preparation of PCG* and the computation of a *PCG iteration* which together form the operations in the original PCG solver as shown in Algorithm 1. Steps 3 to 6 are added to establish fault tolerance. Our proposed error detection scheme is performed in the third step (cf. Section V). If no error is detected, then a checkpoint is periodically generated in a *reliable storage* (e.g. ECC-protected memory [25]). Both error detection criteria, λ and σ, are periodically computed in the fifth step. In case of errors, our adaptive error recovery scheme selects the most promising technique in the sixth step, namely *online correction*, *complete roll-back* and *corrective roll-back* (cf. Section VI).

V. Error Detection

Our proposed error detection scheme exploits specific relations between successive iterations during the execution of PCG. The convergence of PCG and the correctness of the final result is ensured, if those relations are maintained throughout the whole execution [5]. Soft errors, resulting from e.g. transient effects corrupt these relations and become therefore detectable. Thus, the periodic evaluation of those relations ensures a reliable error detection for the PCG solver. In particular, to detect errors between iterations i and k with $k > i$, the following criteria are evaluated.

$$\lambda \approx x_k^T w_i \quad \wedge \quad \frac{r_k^T p_i}{\delta_k \cdot \sigma} \approx 0 \quad \text{if } k > i. \tag{1}$$

The former is referred to as *λ-criterion*, while the latter is referred to as *σ-criterion*. In our previous work [14], we derived the *λ-criterion* and presented its error detection ability for the PCG solver. Now, we complement our error detection technique by the *σ-criterion*, which extends the scope of error detection to the complete set of vectors involved in each iteration of PCG.

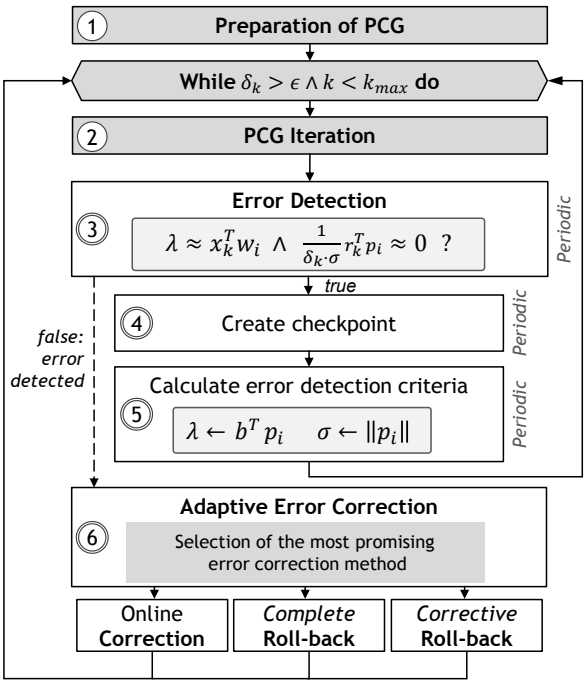

Figure 1. Overview of the Fault-Tolerant Preconditioned Conjugate Gradient Algorithm.

Derivation of the λ and σ-criteria

The *λ-criterion* is based on the combined evaluation of specific *orthogonality* and *residual* relations, which ensure that iteration k is related to each preceding iteration i with $k > i$. At each iteration k, the residual r_k is orthogonal to every preceding search direction p_i [5].

$$r_k \perp p_i \iff \frac{r_k^T p_i}{\|r_k\| \|p_i\|} \approx 0 \quad \text{if } k > i. \tag{2}$$

The *residual* r_k at each iteration k is calculated as

$$r_k = b - Ax_k \tag{3}$$

The combination of Equations 2 and 3 allows the derivation of the following equation [14]:

$$b^T p_i \approx x_k^T w_i \tag{4}$$

The inner product $b^T p_i$ on the left-hand side does not involve vectors from the current iteration k. Thus, this inner product can be calculated during the preceding iteration i. The result is stored in the scalar λ:

$$\lambda = b^T p_i \tag{5}$$

The σ-criterion detects errors in the residual r_k based on the *orthogonality* relation

$$\frac{r_k^T p_i}{\|r_k\| \|p_i\|} \approx 0 \quad \text{if } k > i. \tag{6}$$

The norm of the residual r_k is already computed as δ_k during iteration k, which corresponds to Line 13 of Algorithm 1. The norm of the preceding search direction p_i is independent of any successive computation after iteration i. Thus, its

result is stored in the scalar σ. Finally

$$\frac{r_k^T p_i}{\delta_k \cdot \sigma} \approx 0 \quad \text{if } k > i \text{ with } \delta_k = \|r_k\| \wedge \sigma = \|p_i\|. \qquad (7)$$

VI. Error Correction

To correct errors during the execution of the PCG solver, we presented an error correction scheme in our previous work [14], which utilized two correction techniques, namely online correction and complete roll-back. These error correction techniques are now complemented by a new *corrective roll-back recovery* technique. Our new adaptive error correction method identifies the degree of corruption of the intermediate solution to trade off these three different correction methods against each other. An *online correction* is attempted to correct erroneous intermediate results without any roll-back. If such a correction is not *promising*, the error recovery scheme performs a *complete roll-back* to avoid complete recomputations of PCG. If the utilized checkpoint appears to be corrupted, a *corrective roll-back* is performed to avoid *endless loops*.

A. Identifying Degrees of Corruption for the PCG Solver

Occurring errors are able to corrupt the different relationships between the iterations of the PCG solver. However, errors can take an apparently corrupted approximation x_k closer to the actual solution. Therefore, if the residual r_k in the approximation x_k is closer to zero compared to the residual in the approximation r_i of a checkpoint, then it is *promising* for PCG to continue using x_k. In that case, PCG is likely to require fewer additional iterations to converge compared to a roll-back recovery. Otherwise, if the residual r_k of the approximation x_k is larger than r_i, then a roll-back to iteration i is more reasonable. The details of on-line correction and roll-back recovery are presented below.

A prerequisite to compute the residuals is the absence of floating-point exceptions such as *NaN* and *Inf*. Such elements are replaced by random values if the underlying values are not recoverable.

B. Online Correction, Complete and Corrective Roll-back

An erroneous iteration k is *corrected* if the continuation using x_k is promising to converge in fewer iterations compared to a roll-back to the last checkpoint. *Online correction* re-establishes the *residual* and *orthogonality* relations for successive iterations after iteration k. The following steps are performed for correction: First, the residual r_k is recomputed in the approximation x_k. Second, the search direction p_k is computed using the preconditioned residual $p_k = M^{-1}r_k$.

During *complete roll-back recovery*, the stored vectors are copied to the vectors of the current iteration. *Corrective roll-back recovery* is performed if a checkpoint is used more than once for error recovery. In this case, only the stored approximation x_i is restored and the remaining vectors are corrected according to x_i. The residual r_k is recomputed in the approximation x_k and the search direction p_k is set to the preconditioned residual $p_k = M^{-1}r_k$. Afterwards, the execution of PCG is continued in both cases.

VII. Experimental Results

The proposed fault-tolerance approach for PCG was evaluated with respect to the *runtime overhead for error detection*, achievable *error coverage*, and *runtime overhead for error correction*. Our method is compared with three recent fault tolerance methods for the PCG solver. The first method is the approach from our previous work [14]. The second method is the *periodic correction of the residual* which is proposed in different degrees in [23] as well as in [22]. The third method is the *periodic evaluation of orthogonality and residual relationships* which is proposed in [24].

A. Experimental Setup

For the experiments, the PCG algorithm was tailored to a heterogeneous computing system comprising of multi-core CPUs and many-core GPUs. All parallelizable linear algebra operations were mapped to GPU architectures and GPU-accelerated linear algebra libraries were utilized. All experiments have been performed in double precision on a Nvidia Titan Black GPU. As benchmarks, 25 matrices from the Florida Sparse Matrix Collection [26] were evaluated which are shown shown in Table I. Besides the names and sizes of the matrices ($N \times N$), the number of nonzero elements (*NNZ*) are presented. As a side information, the portion of 0s within the matrices and the condition number are presented (cf. Section II). The evaluated matrices were stored in the compressed sparse row storage format [27].

Name	N	NNZ	Portion of 0s	Condition
nos3	960	15844	98.28%	3.77e+04
bcsstk10	1086	22070	98.13%	5.24e+05
msc01050	1050	26198	97.62%	4.58e+15
bcsstk21	3600	26600	99.79%	1.76e+07
bcsstk11	1473	34241	98.42%	2.21e+08
ex3	1821	52685	98.41%	1.68e+10
ex10hs	2548	57308	99.12%	5.48e+11
nasa2146	2146	72250	98.43%	1.72e+03
sts4098	4098	72356	99.57%	2.17e+08
bcsstk13	2003	83883	97.91%	1.10e+10
msc04515	4515	97707	99.52%	2.27e+06
ex9	3363	99471	99.12%	1.17e+13
aft01	8205	125567	99.81%	4.39e+18
bodyy6	19366	134208	99.96%	7.69e+04
muu	7102	170134	99.66%	7.65e+01
s3rmt3m3	5357	207123	99.28%	2.40e+10
s3rmt3m1	5489	217669	99.28%	2.48e+10
bcsstk28	4410	219024	98.87%	9.45e+08
s3rmq4m1	5489	262943	99.13%	1.77e+10
bcsstk16	4884	290378	98.78%	4.94e+09
bcsstk38	8032	355460	99.45%	5.52e+16
msc23052	23052	1142686	98.95%	9.97e+09
msc10848	10848	1229776	98.95%	9.97e+09
nd3k	9000	3279690	95.95%	1.56e+07
ship_001	34920	3896496	99.68%	3.16e+12

Table I
Overview of evaluated matrices [26].

The chosen input parameters are described below. For the initial guess x_0, a random vector was generated. If the right-hand side b was not available for a matrix, then a random solution x was generated. Using x, the right-hand side b was computed with $Ax = b$. We set all thresholds and intervals according to related work for fair comparison. The error tolerance ϵ was set to 10^{-6} as proposed in [21]. The error detection threshold used in the comparison of floating-point values was set to 10^{-10} as proposed in [24]. The checkpoint generation interval was set to 20 iterations and both sampling and error detection interval were set to 10 iterations as proposed in [23, 24]. The approach from our previous work [14] performs error detection after each iteration. Besides, the *Jacobi-Preconditioner* was applied for preconditioning [5]. In addition to these results, we conducted experiments with other preconditioners such as *SSOR* and *Incomplete Cholesky*. However, the obtained results do not reveal significant differences.

B. Error Model

Different implementations of FPUs exist that may have different error propagation patterns for transient events. In accordance to related works [11, 20, 21, 23, 24], the errors are injected in results of intermediate arithmetic operations, comprising both vectors and scalars. 1500 error injection experiments were evaluated for each matrix. The results of the underlying floating point instructions were modified by randomized bursts of bit flips (1 to 64 bits per randomly chosen vector element or scalar). Each injection comprised the random selection of both an iteration and one of the operations in PCG to generate an erroneous result. Bit flips were also injected into operations that perform error detection.

C. Error Detection Overhead in Error-Free Execution

To investigate the runtime overhead for error detection, we applied the different fault-tolerance techniques to the PCG algorithm and compared the runtime in error-free executions. Figure 2 shows the results of this investigation. The evaluated matrices are ordered by the number of non-zero elements. The error detection overhead which is introduced by our method ranges from 0.04% to 1.7%. The overhead of our method becomes smaller with increasing numbers of non-zero elements, since it does not require matrix-vector multiplications. Therefore, our method scales very well with increasing problem sizes. The compared methods show an almost uniform overhead ranging from 7.8% to 9.9% [22, 23] as well as from 6.7% to 9.3% [24] respectively. For the three largest matrices, *msc10848*, *nd3k* and *ship_001*, the overhead of the proposed method is only between 0.4% and 1.5% compared to the overhead of the methods from related works [22–24]. Compared to our previous work, the overhead is only 33.7% on average. This reduction can be explained by the increased error detection interval.

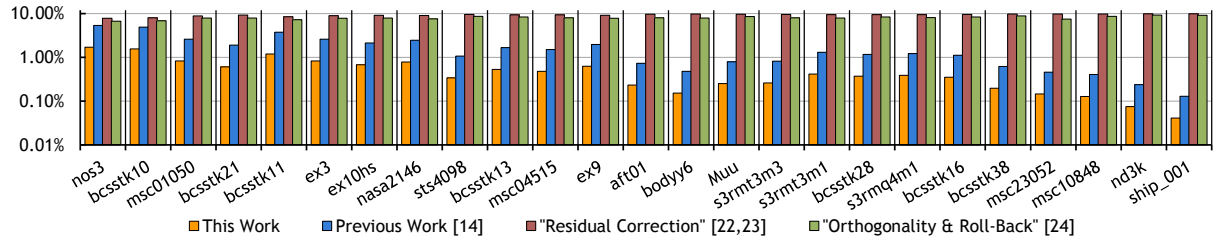

Figure 2. Average error detection overhead in error-free execution.

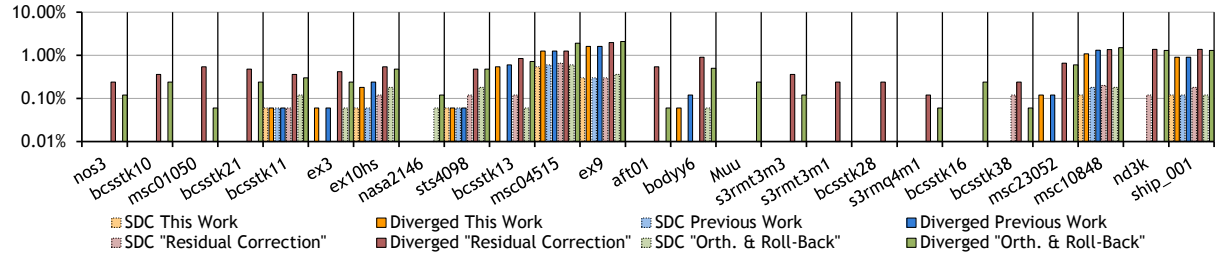

Figure 3. Portion of execution failures separated into SDCs and executions that exceeded the specified limit of iterations.

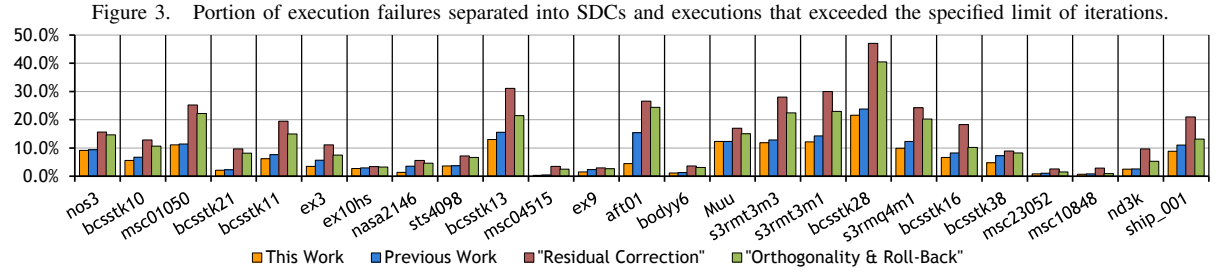

Figure 4. Average error correction overhead in case of errors.

D. Error Coverage and Error Correction Results

The error coverage corresponds to the portion of erroneous executions which converged to a correct result after error detection and correction and did not exceed a certain limit of iterations. We defined limits for both the number of iterations and the maximum deviation to the correct solution to evaluate silent data corruptions (i.e. convergence to a wrong result). Each experiment that exceeded at least one of those limits was considered a failure. The limit for the number of iterations was set to 200.000 iterations. The maximum acceptable deviation was set to 10^{-10}. Figure 3 shows the portion of execution failures separated into SDCs and diverged executions that exceeded the limit of iterations. The error coverage of our method is at least as high as the error coverage of the compared methods. At the same time, the error coverage is significantly increased for the majority of the evaluated matrices. Compared to our previous work, the error coverage is not only maintained, but often improved although the error detection interval was increased from 1 to 10 iterations. The number of executions which resulted in SDC is at most as high as the related work. Our method significantly reduces the overall number of SDCs. Figure 4 shows the *runtime overhead for error correction* of the different fault tolerance methods. The methods are compared by the average overhead required for successful convergence providing a correct result in erroneous cases. Our method reduces the error correction overhead on average by 17.1% compared to our previous work. Compared to related work, our method reduces the error correction overhead on average by 59.3% [22, 23] and 47.2% [24] respectively.

VIII. Conclusion

Efficient linear system solvers are an essential prerequisite for many compute-intensive applications. However, strong reliability requirements are imposed on such applications. The growing spectrum of reliability threats for CMOS devices makes the application of fault tolerance measures mandatory. In this work, we presented a new *fault-tolerant preconditioned conjugate gradient solver* with extremely low runtime overhead. The method achieves a significant reduction of both, the error detection overhead as well as the number of expensive recomputations. Our method scales very well with increasing problem sizes, since it does not involve matrix-vector multiplications. Experimental results show a runtime overhead for error detection ranging from 0.04% to 1.7%. Compared to our previous work, this method reduces the error detection overhead on average by 66.3%. For the largest matrices, the runtime overhead of the proposed method is only between 0.4% and 1.5% compared to related works. At the same time, the error coverage is at least as high as and often higher than the coverage of the related work. For erroneous executions, our proposed method significantly reduces the number of additional iterations to achieve correct results.

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 310/2) at the University of Stuttgart.

Bibliography

[1] S. Khaitan and A. Gupta, *High Performance Computing in Power and Energy Systems*, ser. Power Systems. Springer Berlin Heidelberg, 2012.

[2] I. Smith, D. Griffiths, and L. Margetts, *Programming the Finite Element Method*. Wiley, Oct 2013.

[3] D. Yuen *et al.*, *GPU Solutions to Multi-scale Problems in Science and Engineering*, ser. Lecture Notes in Earth System Sciences. Springer, 2013.

[4] K. Daloukas *et al.*, "A 3-D Fast Transform-based Preconditioner for Large-Scale Power Grid Analysis on Massively Parallel Architectures", in *Intl. Symposium on Quality Electronic Design (ISQED)*, Santa Clara, USA, Mar. 2014, pp. 723–730.

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Siam, 2003.

[6] M. Ament *et al.*, "A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform", in *18th Euromicro Intl. Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Pisa, Italy, Feb. 2010, pp. 583–592.

[7] E. Müller *et al.*, "Matrix-free GPU Implementation of a Preconditioned Conjugate Gradient Solver for Anisotropic Elliptic PDEs", *Computing and Visualization in Science*, vol. 16, no. 2, pp. 41–58, 2013.

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

[9] "The International Technology Roadmap for Semiconductors 2013 Edition". [Online]. Available: http://www.itrs.net/Links/2013ITRS/Home2013.htm

[10] F. Cappello *et al.*, "Toward Exascale Resilience: 2014 Update", in *Supercomputing Frontiers and Innovations*, vol. 1, no. 1, 2014.

[11] G. Bronevetsky and B. de Supinski, "Soft Error Vulnerability of Iterative Linear Algebra Methods", in *Proc. of the Intl. Conference on Supercomputing*, Island of Kos, Greece, Nov. 2008, pp. 155–164.

[12] M. Shantharam, S. Srinivasmurthy, and P. Raghavan, "Characterizing the Impact of Soft Errors on Iterative Methods in Scientific Computing", in *Proc. of the Intl. Conference on Supercomputing*, Seattle, USA, Nov. 2011, pp. 152–161.

[13] A. Moody *et al.*, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System", in *Proc. of the ACM/IEEE Intl. Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, USA, Nov. 2010, pp. 1–11.

[14] A. Schöll *et al.*, "Efficient On-Line Fault-Tolerance for the Preconditioned Conjugate Gradient Method", in *Proc. of the IEEE Intl. On-Line Testing Symposium (IOLTS)*, Elia, Greece, Jul. 2015, pp. 95–100.

[15] D. A. Oliveira *et al.*, "Modern GPUs Radiation Sensitivity Evaluation and Mitigation Through Duplication With Comparison", *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3115–3122, Dec. 2014.

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

[17] C. Braun, S. Halder, and H.-J. Wunderlich, "A-ABFT: Autonomous Algorithm-Based Fault Tolerance for Matrix Multiplications on Graphics Processing Units", in *Proc. of The 44th IEEE/IFIP Intl. Conference on Dependable Systems and Networks (DSN)*, Atlanta, USA, Jun. 2014, pp. 443–454.

[18] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Control-Flow Checking by Software Signatures", *IEEE Trans. on Reliability*, vol. 51, no. 1, pp. 111–122, Aug. 2002.

[19] K. Wilken and J. P. Shen, "Continuous Signature Monitoring: Low-cost Concurrent Detection of Processor Control Errors", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 6, pp. 629–641, Jun. 1990.

[20] M. Shantharam, S. Srinivasmurthy, and P. Raghavan, "Fault Tolerant Preconditioned Conjugate Gradient for Sparse Linear System Solution", in *Proc. of the ACM Intl. Conference on Supercomputing*, Venice, Italy, Jun. 2012, pp. 69–78.

[21] J. Sloan, R. Kumar, and G. Bronevetsky, "An Algorithmic Approach to Error Localization and Partial Recomputation for Low-Overhead Fault Tolerance", in *Proc. of the 43rd IEEE/IFIP Intl. Conference on Dependable Systems and Networks (DSN)*, Budapest, Hungary, Jun. 2013, pp. 1–12.

[22] F. Oboril *et al.*, "Numerical Defect Correction as an Algorithm-Based Fault Tolerance Technique for Iterative Solvers", in *IEEE Pacific Rim Intl. Symp. on Dependable Computing (PRDC)*, Pasadena, USA, Dec. 2011, pp. 144–153.

[23] P. Sao and R. Vuduc, "Self-stabilizing Iterative Solvers", in *Proc. of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, Nov. 2013, pp. 4:1–4:8.

[24] Z. Chen, "Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods", in *Proc. of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Shenzhen, China, Feb. 2013, pp. 167–176.

[25] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*, M. Kaufmann, Ed. Elsevier, 2010.

[26] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection", *ACM Trans. on Mathematical Software*, vol. 38, no. 1, pp. 1:1–1:25, Nov. 2011.

[27] E. F. D'Azevedo, M. R. Fahey, and R. T. Mills, "Vectorized Sparse Matrix Multiply for Compressed Row Storage Format", in *Computational Science*, ser. Lecture Notes in Computer Science, 2005, vol. 3514, pp. 99–106.