Efficient On-Line Fault-Tolerance for the Preconditioned Conjugate Gradient Method

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

Proceedings of the 21st IEEE International On-Line Testing Symposium (IOLTS'15) Elia, Halkidiki, Greece, 6-8 July 2015

doi: http://dx.doi.org/10.1109/IOLTS.2015.7229839

Abstract: Linear system solvers are key components of many scientific applications and they can benefit significantly from modern heterogeneous computer architectures. However, such nano-scaled CMOS devices face an increasing number of reliability threats, which make the integration of fault tolerance mandatory. The preconditioned conjugate gradient method (PCG) is a very popular solver since it typically finds solutions faster than direct methods, and it is less vulnerable to transient effects. However, as latest research shows, the vulnerability is still considerable. Even single errors caused, for instance, by marginal hardware, harsh operating conditions or particle radiation can increase execution times considerably or corrupt solutions without indication. In this work, a novel and highly efficient fault-tolerant PCG method is presented. The method applies only two inner products to reliably detect errors. In case of errors, the method automatically selects between roll-back and efficient on-line correction. This significantly reduces the error detection overhead and expensive re-computations.

[1] **IEEE COPYRIGHT NOTICE**
©2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Efficient On-Line Fault-Tolerance for the Preconditioned Conjugate Gradient Method

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 key components of many scientific applications and they can benefit significantly from modern heterogeneous computer architectures. However, such nano-scaled CMOS devices face an increasing number of reliability threats, which make the integration of fault tolerance mandatory.**

The preconditioned conjugate gradient method (PCG) is a very popular solver since it typically finds solutions faster than direct methods, and it is less vulnerable to transient effects. However, as latest research shows, the vulnerability is still considerable. Even single errors caused, for instance, by marginal hardware, harsh operating conditions or particle radiation can increase execution times considerably or corrupt solutions without indication.

In this work, a novel and highly efficient fault-tolerant PCG method is presented. The method applies only two inner products to reliably detect errors. In case of errors, the method automatically selects between roll-back and efficient on-line correction. This significantly reduces the error detection overhead and expensive re-computations.

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

I. Introduction

The solution of linear systems is an essential computational task in many applications in science and engineering [1], such as structural mechanics [2], computational fluid dynamics [3], the study of electromagnetic fields [4] or power grid analysis [5]. The *preconditioned conjugate gradient method* (PCG) [6] is one of the most widely used methods for solving sparse linear systems of the form $Ax = b$. PCG typically finds a solution faster compared to direct methods like e.g. the Gaussian-Elimination. Furthermore, the PCG method is well suited for parallel computing, which makes it an ideal candidate for heterogeneous computing systems comprising of multi-core CPUs and many-core GPUs. Recent works in this area exploits different characteristics of the underlying linear algebra operations to gain significant speedups [7, 8].

However, such nano-scaled CMOS devices which are manufactured in 14nm process technology and below, become increasingly vulnerable to a growing spectrum of reliability threats like transient effects, process variations and latent defects, as well as stress and aging mechanisms [9, 10]. In the near future, the vulnerability will even grow with smaller chip feature sizes [11], demanding the application of fault tolerance measures.

Compared to direct methods, iterative methods such as PCG have an inherent robustness with respect to transient effects causing soft errors [6]. However, recent works [12, 13] show that iterative methods still exhibit a high vulnerability to soft errors. In particular, single soft errors may degrade the performance of PCG by factors of 200x or corrupt the computed solution without indication, resulting in silent data corruption. In these cases, the derived solution does not satisfy the original problem $Ax = b$, despite convergence.

A traditional approach to detect errors in the execution of PCG is to test the computed solution using the original problem $Ax = b$. However, this approach only detects errors without correcting them. In case of errors, the execution of PCG has to be repeated, until a correct result is obtained.

More sophisticated approaches, which are discussed in section III, were proposed in recent years. They address fault tolerance for PCG to different degrees. Some limit the fault detection to specific subroutines of PCG. Some approaches recover from errors by using traditional checkpoint-roll-back techniques, which induce high cost in recomputing erroneous results [14].

In this work, a new efficient *fault-tolerant preconditioned conjugate gradient method* is presented. The proposed error detection is very efficient since it only requires two periodically applied inner products. Since the error detection method does not involve matrix operations, it scales very well with increasing problem sizes. Besides, the proposed error recovery scheme adaptively selects a correction method. In particular, the recovery scheme attempts to correct corrupted intermediate results between iterations on-line to regain valid states. Only if a correction is not promising, a roll-back to the preceding checkpoint is performed to avoid complete recomputations.

As discussed below in Section VII, the proposed fault-tolerance method enables a significant reduction of both, the error detection overhead as well as the number of expensive recomputations. Despite the increased error detection efficiency, there is no loss in error coverage.

II. Preconditioned Conjugate Gradient Method

The PCG method [6] is an iterative approach, which solves large linear systems $Ax = b$, given that the underlying matrix A is symmetric and positive-definite. Algorithm 1 shows the fundamental operations of PCG in pseudo code.

The inputs include a coefficient matrix A, a preconditioner M, a vector b, the initial guess vector x_0, an error tolerance ϵ to accept an approximation as solution and a maximum number of iterations k_{max}.

With every iteration of the *PCG loop* an improved approximation x_k to the exact solution is provided.

Algorithm 1: Preconditioned conjugate gradient (PCG)

```
Input: $A$,$M$,$b$, $x_0$, $\epsilon$, $k_{max}$
1  $r_0 \leftarrow b - Ax_0$;
2  $s_0 \leftarrow M^{-1} r_0$;                       // Preconditioning
3  $p_0 \leftarrow s_0$;
4  $\delta_0 \leftarrow r_0^T r_0$;
5  $k \leftarrow 0$;
6  while $\delta_k > \epsilon \wedge k < k_{max}$ do
7      $w_k \leftarrow Ap_k$;
8      $\gamma \leftarrow r_k^T s_k$;
9      $\alpha \leftarrow \frac{\gamma}{p_k^T w_k}$;
10     $x_{k+1} \leftarrow x_k + \alpha p_k$;   // Improve approximation
11     $r_{k+1} \leftarrow r_k - \alpha w_k$;           // Update residual
12     $s_{k+1} \leftarrow M^{-1} r_{k+1}$;          // Preconditioning
13     $\delta_{k+1} \leftarrow r_{k+1}^T r_{k+1}$;
14     $\beta \leftarrow \frac{r_{k+1}^T s_{k+1}}{\gamma}$;
15     $p_{k+1} \leftarrow s_{k+1} + \beta p_k$;  // New search direction
16     $k \leftarrow k + 1$;
17 end
```

PCG represents the solution x as a linear combination of *search directions* $p_0, p_1, p_2, ..., p_N$ with $x = x_0 + \sum_{k \leq N} \alpha_k p_k$. In every subsequent iteration, a new search direction p_k is computed from the residual r_k such that $p_i^T A p_k = 0, k \neq i$. Therefore, each residual r_k is orthogonal to each preceding search direction p_i and each residual r_i, $i < k$. *Preconditioning* [6] is a useful prerequisite for the PCG method since it accelerates convergence significantly. The goal of preconditioning is to diminish the condition number of the coefficient matrix A, which directly affects the rate of convergence. As will be discussed below, the proposed fault-tolerant PCG method is independent of the utilized preconditioning operation.

III. State of the Art

The investigation of fault tolerance for linear algebra algorithms is an active research area. *Algorithm-Based Fault Tolerance (ABFT)* is aimed at linear algebra operations such as matrix operations [15]. The basic idea of ABFT is to evaluate checksum mismatches between encoded input data and encoded results to detect errors. More sophisticated ABFT methods address the influence of rounding errors within checksums [16].

Fault tolerance for PCG demands different methods, since the underlying operations such as inner products are not completely protectable with error detection codes. Over the last decade, the vulnerability of PCG was assessed and different fault tolerance techniques were proposed:

In [12], the poor inherent ability of PCG to detect errors resulting in silent data corruptions is demonstrated. The influence of soft errors on the performance of linear solvers is examined in [13]. Soft errors are able to degrade the performance of PCG by factors of 200x.

In [17], the evaluation of checksum invariants during matrix-vector operations is proposed to detect errors. During error-free executions, the overhead of this technique is on average 11.3%. Particular structures in matrices are exploited in [18] to detect errors in sparse matrix-vector multiplications. The experimental results indicate a minimal overhead of 17%. Based on partial recomputation, the proposed method is extended in [19] to localize and correct errors during sparse matrix-vector multiplications.

A different technique, which detects errors based on the inherent relations between internal values in PCG is presented in [20]. If silent data corruptions are detected after a complete execution of PCG, then PCG is repeated on the obtained residual $Ad = r = (b - Ax)$. Because of that, the method avoids repetitions of PCG on the original problem. However, the technique awaits the result of PCG, before it applies error detection. In [21], a self-stabilizing approach is proposed which requires reliable system modes. Based on the convergence conditions of PCG, a stabilization scheme is presented that transforms arbitrary states to valid states. A different error detection technique is proposed in [22]. The technique periodically checks the residual vector and the A-orthogonality of consecutive search direction vectors. A checkpoint-roll-back technique is applied to recover from corrupted states.

In summary, the discussed approaches address fault tolerance for PCG to different degrees. The approaches in [17–19] limit the error detection to specific subroutines of the PCG method. The approaches in [21, 22], which cover complete PCG iterations exhibit significant overheads to detect errors, since they require expensive sparse matrix-vector operations.

IV. Fault-Tolerant Preconditioned Conjugate Gradient Method

In this work, an efficient *fault-tolerant preconditioned conjugate gradient method* is presented. The proposed error detection method introduces low overhead since it only requires two periodically applied inner products. Hence, additional and expensive matrix-vector multiplications are avoided. Whenever possible, the proposed error recovery scheme avoids roll-backs and instead performs *on-line correction*. For these reasons, the proposed fault-tolerance method enables a significant reduction of both, the error detection overhead as well as the number of expensive re-computations. The steps of our *fault-tolerant preconditioned conjugate gradient method* are shown in Figure 1. In the first step, the PCG algorithm is prepared which corresponds to lines 1 to 5 in Algorithm 1. The *PCG loop* comprises the remainder of the steps. In the second step, a *PCG iteration* is computed which corresponds to lines 6 to 17 in Algorithm 1. With every *PCG iteration* an improved approximation x_k

to the exact solution is provided. Steps 3 to 6 are added to provide fault tolerance. In the third step, our proposed error detection method is applied as discussed below in Section V. In the fourth step, a checkpoint is periodically generated in a *reliable storage* (e.g. ECC-protected memory [23]). The error detection criterion λ is periodically computed in the fifth step. In the sixth step, the current approximation x_k is evaluated against the criterion λ to detect errors.

In case of errors, our proposed adaptive error correction method selects the most promising error correction method. It trades off on-line correction against roll-back recovery. This procedure is further discussed below in Section VI.

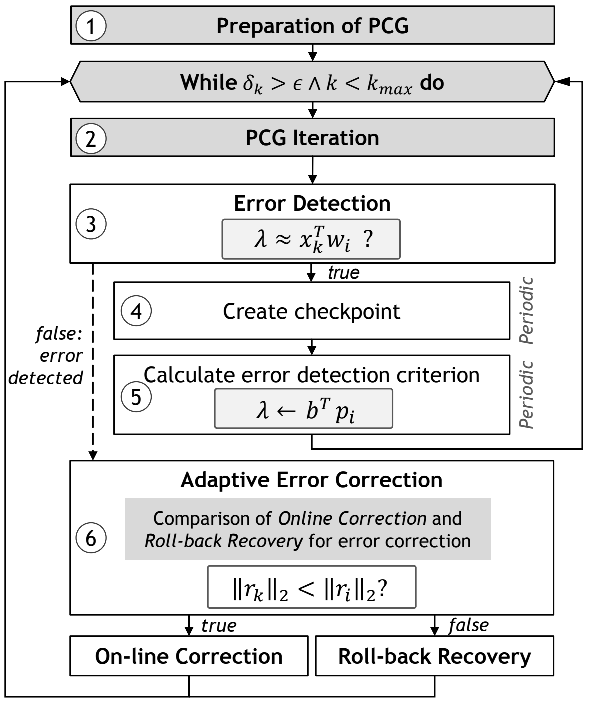
1 Preparation of PCG
While $\delta_k > \epsilon \wedge k < k_{max}$ do
2 PCG Iteration
3 Error Detection
$\lambda \approx x_k^T w_i$?
true
4 Create checkpoint
Periodic
false: error detected
5 Calculate error detection criterion
$\lambda \leftarrow b^T p_i$
Periodic
6 Adaptive Error Correction
Comparison of Online Correction and Roll-back Recovery for error correction
$\|r_k\|_2 < \|r_i\|_2$?
true
false
On-line Correction
Roll-back Recovery

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

V. Error Detection

Our proposed error detection method is based on the property that each successive approximation x_k is linked to its preceding approximations $x_{k-1}, x_{k-2}, ..., x_0$ via specific relations. These comprise *residual* and *orthogonality relations*, which are discussed further below. The relations maintain both the convergence of PCG and the correctness of the final result [6]. Soft errors, resulting from e.g. transient effects, are able to corrupt the computed solution of PCG without indication to the user. However, these relations will be affected in the event of errors. Thus, the evaluation of those relations enables a reliable error detection for the PCG method. In particular, to detect errors between iterations i and k with $i < k$, the following relation has to be evaluated.

$$b^T p_i \approx x_k^T w_i \quad \text{if } k > i. \tag{1}$$

The first inner product $b^T p_i$ is not dependent on results of the successive execution after iteration i and can therefore be stored in a scalar i.e. $\lambda = b^T p_i$. This step leads to the proposed error detection method, which is referred to as λ-*relationship* in the remainder of this work.

$$\lambda \approx x_k^T w_i \quad \text{if } k > i. \tag{2}$$

If equation 2 does not hold for iterations i and k, then an error occurred between them. The derivation of the λ-*relationship* from specific properties of PCG is presented below.

Derivation of the λ-relationship

In the fault-free execution of the PCG algorithm, at each iteration k, the residual r_k is orthogonal to every preceding search direction p_i [6].

$$r_k \perp p_i \iff \frac{r_k^T p_i}{\|r_k\|\|p_i\|} \approx 0 \quad \text{if } k > i. \tag{3}$$

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

$$r_k = b - Ax_k \tag{4}$$

Combining equations 3 and 4 leads to:

$$\frac{(b - Ax_k)^T p_i}{\|b - Ax_k\|\|p_i\|} \approx 0 \tag{5}$$

The numerator $(b - Ax_k)^T p_i$ can be simplified:

$$(b - Ax_k)^T p_i = b^T p_i - x_k^T (Ap_i) \tag{6}$$

The result of Ap_i is already computed, since it was evaluated as $w_i = Ap_i$ in the preceding iteration i. The respective computation is performed in Line 7 of Algorithm 1.

$$b^T p_i - x_k^T (Ap_i) = b^T p_i - x_k^T w_i \tag{7}$$

The derived fraction can be further simplified:

$$\frac{b^T p_i - x_k^T w_i}{\|b - Ax_k\|\|p_i\|} \approx 0 \tag{8}$$

$$\frac{b^T p_i}{\|b - Ax_k\|\|p_i\|} \approx \frac{x_k^T w_i}{\|b - Ax_k\|\|p_i\|} \tag{9}$$

By canceling the common denominator, finally:

$$b^T p_i \approx x_k^T w_i \tag{10}$$

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

$$\lambda = b^T p_i \tag{11}$$

VI. Error Correction

For error correction, we propose an *adaptive error correction method*, which significantly reduces the number of additional iterations to achieve convergence with a correct result. The error correction method selects a specific recovery method according to the identified degree of corruption adaptively. In particular, the method attempts an *on-line correction* to regain valid iteration states from corrupted intermediate results. Only if on-line correction is not *promising*, then a *roll-back recovery* to the preceding checkpoint is performed to avoid complete recomputations.

A. Adaptive Error Correction for the PCG Method

The idea behind the adaptive error correction method is based on the property of PCG, that the residual r_k tends to decrease with successive iterations ($i < k$). If the residual r_k of the approximation x_k is closer to zero than r_i, then x_k is likely to be closer to the exact solution than x_i

$$\|r_k\| \le \|r_i\|$$
$$\Rightarrow \|b - Ax_k\| \le \|b - Ax_i\|$$

In this case, it is *promising* for PCG 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 it is reasonable to perform a roll-back to iteration i. A requirement for computing the residuals is the absence of floating-point exceptions in x_k, which may occur due to errors. Therefore, the elements of x_k are checked for floating-point exceptions, such as *NaN* and *Inf*. Erroneous elements are replaced by random values if the underlying values are not recoverable.

The details of on-line correction and roll-back recovery are presented below.

B. On-line Correction and Roll-back Recovery

If the continuation based on x_k is promising to converge in fewer iterations compared to a roll-back recovery, then the erroneous iteration k is corrected. In particular, our proposed *on-line correction* method re-establishes the residual relationship as well as orthogonality relationships for successive iterations after iteration k.

The correction method performs the following steps. First, the algorithm's residual r_k is recomputed in the approximation x_k. Second, the search direction p_k is set to the preconditioned residual $p_k = M^{-1}r_k$.

During *roll-back recovery*, the stored vectors are copied to the vectors of the current iteration. Afterwards, the execution of PCG is continued.

VII. Experimental Results

The proposed fault tolerance approach has been evaluated in terms of *error detection overhead*, achievable *error coverage* and *error correction overhead*. We compared our proposed method with two recent methods which addressed fault tolerance for the PCG method. The first method is the *periodic correction of the residual* which is proposed in different degrees in [21] as well as in [20]. The second method is the *periodic evaluation of orthogonality and residual relationships* which is proposed in [22]. This method applies a checkpoint-roll-back technique to recover from errors.

A. Experimental Setup

For the experiments, PCG 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, using GPU-accelerated linear algebra libraries. The evaluated matrices were stored in the compressed sparse row storage format (CSR [24]). All experiments have been performed in double precision on a Nvidia Titan Black GPU.

As benchmarks, 25 matrices from the Florida Sparse Matrix Collection [25] were evaluated which are shown shown in Table I. Besides the name and size of the matrices (*N*), the number of nonzero elements (*NNZ*), the portion of 0s within the matrix and the condition number are presented.

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 [25].

The input parameters for the experiments 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 $Ax = b$.

In our experiments, we set all thresholds and intervals according to related work for fair comparison. As proposed in [19], the error tolerance ϵ was set to 10^{-6}. The error detection threshold used in the comparison of floating-point values was set to 10^{-10} as proposed in [22]. The checkpoint generation interval was set to 20 iterations and both sampling and checking interval were set to 10 iterations, as proposed in [21, 22].

Furthermore, the *Jacobi-Preconditioner* was utilized for preconditioning [6]. In addition to the results obtained with the Jacobi preconditioner, we conducted experiments with other preconditioners such as SSOR and incomplete Cholesky. However, the corresponding results do not show significant differences.

B. Error Model

In accordance to related work [12, 17–19, 21, 22], this evaluation focuses on erroneous outputs of numerical computations comprising erroneous result vectors as well as erroneous scalars. Other manifestations of faults, that impact for instance the control flow or corrupt other data in the memory can be treated by low overhead techniques such as error-correcting codes, signature monitoring, or assertions [26, 27]. For each matrix, 1500 error injection experiments were performed. In each experiment, one multi-bit error of different multiplicity was injected on-line. For the injection, one iteration and one of the operations in PCG were randomly chosen to generate an erroneous result. The results of the underlying floating point instructions were then modified by multiple bit flips. Errors were also injected into operations that perform error detection.

C. *Error Detection Overhead in Error-Free Execution*

We compared the runtime of the PCG algorithm in error-free executions under the application of the different fault tolerance methods. The error detection overhead in error-free executions is shown in Figure 2. The evaluated matrices are ordered by the number of non-zero elements.

The error detection overhead which is introduced by our method ranges between 0.1% and 5.4%. With increasing numbers of non-zero elements, the overhead of our method becomes smaller, since it does not require matrix-vector multiplications. The compared methods show an almost uniform overhead ranging between 7.8% and 9.9% [20, 21] as well as 6.7% and 9.3% [22] respectively. For the three largest matrices, *msc10848*, *nd3k* and *ship_001*, the overhead of the proposed method is only between 1.3% and 4.7% of the overhead of the compared methods [20–22].

D. *Error Coverage and Error Correction Results*

The most important property of a fault tolerant method is its error coverage. For the PCG method, the *error coverage* corresponds to the portion of erroneous executions which converged within a certain limit of iterations and computed a correct result.

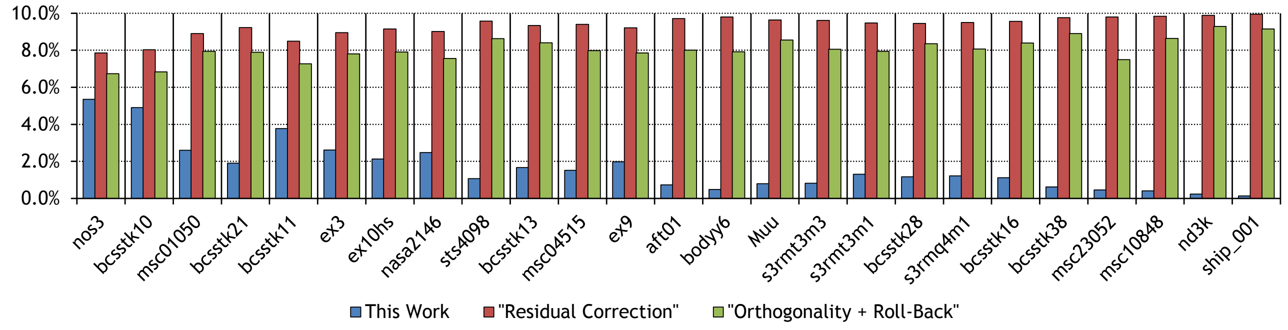

Figure 2. Error detection overhead in error-free execution.

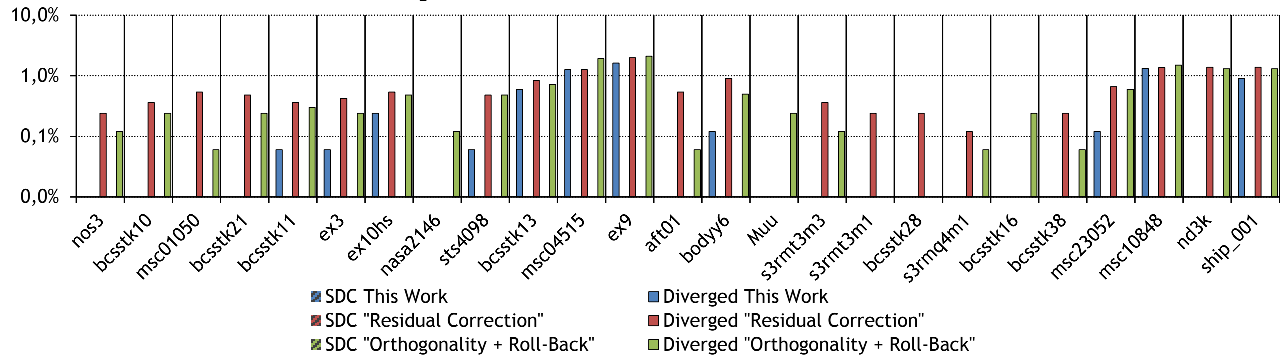

Figure 3. Portion of execution failures.

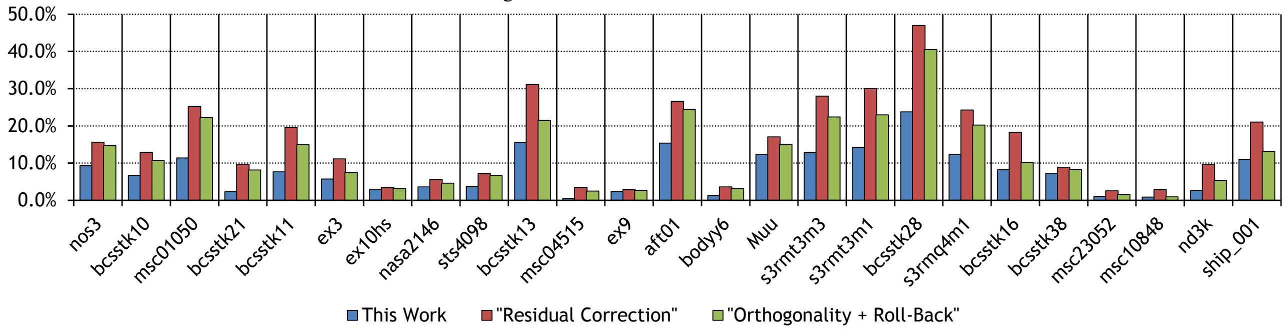

Figure 4. Error correction overhead in case of errors.

We defined limits for both the number of iterations and the maximum deviation to the correct solution to evaluate silent data corruptions (SDC). If one of those limits was reached, the experiment was considered a failure. The limit for the number of iterations was set to 200.000 iterations. The deviation was set to 10^{-10}. The portion of execution failures is presented in Figure 3 (Diverged: no convergence within the limit of iterations; SDC: convergence to a wrong result). The error coverage of our proposed fault tolerant method is at least as high as the error coverage of the compared methods. For the majority of the evaluated matrices, the error coverage is significantly increased. The number of executions causing SDC is also at most as high as the related work and significantly reduced for many matrices.

The *error correction overhead* of the different fault tolerance methods is shown in Figure 4. The methods are compared in terms of the average overhead required for successful convergence providing a correct result in erroneous cases. Compared to the related work methods, our method reduces the error correction overhead on average by 49.7% [20, 21] and 34.8% [22] respectively.

VIII. Conclusion

Efficient linear system solvers are an essential prerequisite for many applications in scientific computing and engineering. However, many of those applications exhibit strong reliability requirements to provide trustworthy results. The growing spectrum of reliability threats for CMOS devices makes the application of fault tolerance measures mandatory.

In this work, we presented a novel *fault-tolerant preconditioned conjugate gradient method*. The method enables a significant reduction of both, the error detection overhead as well as the number of expensive recomputations. The proposed error detection applies only two periodically inner products on-line and is therefore very efficient. Our proposed error detection method scales very well with increasing problem sizes, since it does not involve matrix operations. The experimental evaluation shows an error detection overhead ranging between 0.1% and 5.4%, which decreases with larger matrices. At the same time, the error coverage is at least as high as the error 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] F. Cappello, "Fault Tolerance in Petascale/ Exascale Systems: Current Knowledge, Challenges and Research Opportunities", *Intl. Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 212–226, 2009.

[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] A. Peixoto de Camargos *et al.*, "Efficient Parallel Preconditioned Conjugate Gradient Solver on GPU for FE Modeling of Electromagnetic Fields in Highly Dissipative Media", *IEEE Trans. on Magnetics*, vol. 50, no. 2, pp. 569–572, Feb 2014.

[5] K. Daloukas *et al.*, "A 3-D Fast Transform-based Preconditioner for Large-Scale Power Grid Analysis on Massively Parallel Architectures", in *15th Intl. Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA, Mar. 2014, pp. 723–730.

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

[7] 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.

[8] R. Helfenstein and J. Koko, "Parallel Preconditioned Conjugate Gradient Algorithm on GPU ", in *Proc. of the 15th Intl. Congress on Computational and Applied Mathematics (ICCAM)*, vol. 236, no. 15, Leuven, Belgium, July 2012, pp. 3584 – 3590.

[9] 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.

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

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

[12] 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.

[13] 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, WA, USA, Nov. 2011, pp. 152–161.

[14] 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, LA, USA, Nov. 2010, pp. 1–11.

[15] 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.

[16] 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, Georgia, USA, Jun. 2014, pp. 443–454.

[17] 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.

[18] J. Sloan, R. Kumar, and G. Bronevetsky, "Algorithmic Approaches to Low Overhead Fault Detection for Sparse Linear Algebra", in *Proc. of the 42nd IEEE/IFIP Intl. Conference on Dependable Systems and Networks (DSN)*, Boston, MA, USA, 2012, pp. 1–12.

[19] 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.

[20] 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, CA, USA, Dec. 2011, pp. 144–153.

[21] 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.

[22] 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.

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

[24] E. F. D'Azevedo, M. R. Fahey, and R. T. Mills, "Vectorized Sparse Matrix Multiply for Compressed Row Storage Format", in *Computational Science - ICCS 2005*, ser. Lecture Notes in Computer Science, V. S. Sunderam *et al.*, Eds. Springer Berlin Heidelberg, 2005, vol. 3514, pp. 99–106.

[25] 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.

[26] 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.

[27] 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.