# Applying Efficient Fault Tolerance to Enable the Preconditioned Conjugate Gradient Solver on Approximate Computing Hardware

Schöll, Alexander; Braun, Claus; Wunderlich, Hans-Joachim

Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'16) University of Connecticut, USA, 19-20 September 2016

doi: http://dx.doi.org/10.1109/DFT.2016.7684063

**Abstract:** A new technique is presented that allows to execute the preconditioned conjugate gradient (PCG) solver on approximate hardware while ensuring correct solver results. This technique expands the scope of approximate computing to scientific and engineering applications. The changing error resilience of PCG during the solving process is exploited by different levels of approximation which trade off numerical accuracy and hardware utilization. Such approximation levels are determined at runtime by periodically estimating the error resilience. An efficient fault tolerance technique allows reductions in hardware utilization by ensuring the continued exploitation of maximum allowed energy-accuracy trade-offs. Experimental results show that the hardware utilization is reduced on average by 14.5% and by up to 41.0% compared to executing PCG on accurate hardware.

Preprint

# Applying Efficient Fault Tolerance to Enable the Preconditioned Conjugate Gradient Solver on Approximate Computing Hardware

Alexander Schöll, Claus Braun and Hans-Joachim Wunderlich

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

***Abstract*—A new technique is presented that allows to execute the preconditioned conjugate gradient (PCG) solver on approximate hardware while ensuring correct solver results. This technique expands the scope of approximate computing to scientific and engineering applications. The changing error resilience of PCG during the solving process is exploited by different levels of approximation which trade off numerical precision and hardware utilization. Such approximation levels are determined at runtime by periodically estimating the error resilience. An efficient fault tolerance technique allows reductions in hardware utilization by ensuring the continued exploitation of suitable energy-precision trade-offs. Experimental results show that the hardware utilization is reduced on average by 14.5% and by up to 41.0% compared to executing PCG on precise hardware.**

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

## I. Introduction

Approximate computing constitutes an emerging paradigm that exploits the inherent error resilience of applications to trade-off computational precision in exchange for reduced runtime or energy demands [1, 2]. Different applications in, for instance, multimedia and signal processing exhibit error resilience to certain numerical errors and often do not need to compute *perfect* results. The resilience to such *approximation errors* allows the use of energy-efficient approximate hardware or software-based approximation techniques [3]. Different portions in applications typically exhibit different sensitivities to approximation errors [4]. At the same time, this error resilience may also change over time. Such a behavior is exhibited by different iterative methods like linear system solvers [5].

Many large-scale applications in science and engineering rely on linear systems including structural mechanics [6], computational fluid dynamics [7], or power grid analysis [8]. Therefore, the efficient solution of linear systems is an essential computational task in high-performance computing (HPC). The *preconditioned conjugate gradient* (PCG) solver [9] is an iterative technique and one of the most important methods to solve large linear systems of the form $Ax = b$. Since the energy demands of such compute-intensive tasks are already a constraining factor in future exascale HPC systems [10], the expansion of approximate computing techniques to applications in science and engineering is required to overcome future energy challenges.

Such applications often rely on tight accuracy constraints which render approximation techniques exploiting single degrees of precision (e.g. relying on single approximate hardware designs) unsuitable. The gained energy savings are often canceled out by additional PCG iterations that are required to achieve correct convergence. Instead, *approximation levels* $L_{\text{Approx}}$ that offer different degrees of precision (e.g. with certain numbers of precise bits) [11–13] have to be adaptively utilized according to the changing error resilience. To ensure correct solver results while minimizing energy demands, the error resilience must be dynamically evaluated during the iterative solving process. Such a periodic evaluation not only requires to estimate the error resilience *effectively*, but also highly efficiently. One idea [5] is to begin the execution using the lowest available degree of precision which is increased if the underlying *optimization function* is violated (i.e. for PCG: $\min_x E(x) := \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$ with $\langle \cdot, \cdot \rangle$ being the inner product [14]). While this optimization function can be directly evaluated for some iterative methods, the PCG method, however, does not inherently compute it. Additionally computing this function induces significant runtime overheads since it requires additional expensive matrix operations.

We present a new technique that allows the execution of the PCG solver on approximate hardware to gain hardware utilization reductions while still providing correct solver results. Thus, this technique extends the application scope of approximate computing to scientific and engineering applications. The presented technique is based on the insight that the error resilience can change over time. Exploiting this observation requires the execution with changing degrees of precision. To determine suitable *approximation levels* at runtime, the error resilience is estimated using inherent solver properties which induce only low hardware utilization. An efficient fault tolerance technique such as [15] ensures the continued exploitation of suitable energy-precision trade-offs. At the same time, reductions in hardware utilization are enabled by efficiently adapting approximation levels. The hardware utilization translates to energy demand which can be quantified by an in-depth analysis e.g. using standard cell libraries. While such a technology-specific energy discussion is not in the scope of this work, the applicability of approximate computing in the scientific and engineering domain is shown. Experimental results show that the number of iterations is only increased on average by 9.5% while

the complete hardware utilization is reduced on average by 14.5% compared to executing PCG on precise hardware.

## II. Preconditioned Conjugate Gradient Method

The PCG method [9] iteratively solves linear systems $Ax = b$, when $A$ is a symmetric positive-definite matrix. Compared to direct methods like the Gaussian-Elimination, PCG finds a solution typically faster. To make a clear distinction between the error induced by approximate hardware and the outcomes of PCG iterations, we use the term *intermediate result* for $x$, which is referred to as *approximation* $x$ in literature. 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 $M$, and an upper limit for the number of iterations $k_{max}$. The constraints for result accuracy are set by tolerances $\epsilon := (\epsilon_a, \epsilon_r)$ to accept a sufficiently good intermediate result $x_k$. Based on the initial guess vector $x_0$, each iteration of the *PCG loop* provides an improved intermediate result $x_k$ with respect to the exact solution.

**Algorithm 1:** Preconditioned Conjugate Gradient

**Input**: $A$,$M$,$b$, $x_0$, $\epsilon$, $k_{max}$

$r_0 \leftarrow b - Ax_0$ ; // Initial residual vector
$s_0 \leftarrow M^{-1}r_0$; // Preconditioning
$p_0 \leftarrow s_0$; // Initial search direction
$\delta_0 \leftarrow r_0^T r_0$; // Residual
$k \leftarrow 0$ ; // Iteration count
/* PCG loop */
**while** $(\delta_k > \epsilon_a) \wedge (\delta_k/\delta_0 > \epsilon_r) \wedge (k < k_{max})$ **do**
  $w_k \leftarrow Ap_k$;
  $\gamma \leftarrow r_k^T s_k$;
  $\alpha \leftarrow \frac{\gamma}{p_k^T w_k}$;
  $x_{k+1} \leftarrow x_k + \alpha p_k$; // Next interm. result
  $r_{k+1} \leftarrow r_k - \alpha w_k$; // Update residual vector
  $s_{k+1} \leftarrow M^{-1} r_{k+1}$; // Preconditioning
  $\delta_{k+1} \leftarrow r_{k+1}^T r_{k+1}$; // Update residual
  $\beta \leftarrow \frac{r_{k+1}^T s_{k+1}}{\gamma}$;
  $p_{k+1} \leftarrow s_{k+1} + \beta p_k$; // New search direction
  $k \leftarrow k + 1$;
**end**

PCG represents 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 each subsequent iteration $k$, a new search direction $p_k$ is computed from the residual $r_k$ such that $p_i \perp Ap_k, k \neq i$. The time complexity of PCG depends on both the size and the condition number of the matrix $A$ [9]. A suitable *preconditioner* $M$ is able to diminish the condition number of the matrix $A$, which improves the rate of convergence.

As discussed further below, the error resilience of PCG is closely related to the update vector length $||u_k||_2$ (with $u_k = x_k - x_{k-1}$). Figure 1 shows two examples (cf. Section V-C) for PCG executions in which the update vectors $u_k$ range within several orders of magnitude before converging to a correct result. The update vectors $u_k$ are often large for early PCG iterations and approach zero when PCG converges to the solution.

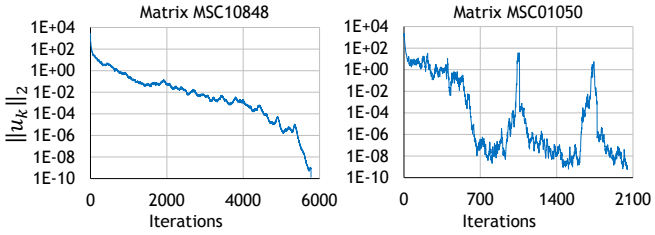


Figure 1. Comparison of update vectors $u_k$ at runtime for two input matrices $A$.

## III. State of the Art

The approximate computing paradigm [1, 2] has been applied to all layers of the computing stack including circuits, architectures and programming models. The spectrum of proposed approximate hardware designs ranges from approximate adder structures [16, 17] over approximate floating-point components [12, 18] to structures that allow to configure the underlying precision at runtime [11, 13, 19]. On the software level, the approximate nature of neural networks is exploited to mimic certain algorithms [3]. In [20], an approximate computing technique for a direct Cholesky decomposition based solver is presented that targets well-conditioned problems arising in video processing applications. For iterative methods on approximate computing hardware, the technique in [5] starts the execution using the lowest available degree of precision, which is increased if the underlying *optimization function* is violated. While some iterative methods rely on evaluating their underlying optimization function, *Krylov-subspace methods* including PCG typically do not rely on this evaluation to find solutions. For PCG, the optimization function is $\min_x E(x) := \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$ [14]. The periodic computation of this function induces significant runtime overheads since it requires an additional expensive matrix-vector multiplication. Such additional matrix-vector multiplications often cancel out potential energy savings as experimental results (cf. Section V-E) show.

Different fault tolerance approaches were proposed for the PCG method to detect and correct transient effects causing soft errors: In [21], PCG is repeated on an auxiliary problem, if an incorrect solution is detected after the completion of PCG. In such a case, PCG is repeated on the obtained residual $Ad = r = (b - Ax)$. While the method aims to avoid repetitions of PCG on the original problem, it awaits the result after complete convergence of PCG before it applies error detection. Inherently reliable system modes are periodically required in [22] to *stabilize* the solver execution. Such stabilizations exploit the convergence conditions of PCG to transform arbitrary iterations to valid iterations. The periodic check of both the residual invariant (i.e. $r_k \approx b - Ax_k$) and the orthogonality of consecutive search direction vectors is used in [23] for error detection. In [15], we presented a fault tolerance technique that exploits inherent orthogonality invariants of PCG to check intermediate results for errors.

Instead of relying on expensive matrix operations to detect errors as in [21–23] with complexity $\mathcal{O}(NNZ)$, this approach only relies on inner products with complexity $\mathcal{O}(N)$ with $NNZ \gg N$ which allows very low runtime overheads. $NNZ$ is the number of non-zero elements in the matrix $A$ with size $N \times N$.

## IV. Using Approximate Hardware for the Preconditioned Conjugate Gradient Solver

In this work, a new technique is presented that allows to execute the *preconditioned conjugate gradient* (PCG) solver on approximate hardware while ensuring correct solutions within the required accuracy. The inherent error resilience of PCG allows to trade off numerical precision and hardware utilization (e.g. energy savings) for certain PCG operations and iterations. Since the error resilience may change between PCG iterations, different *approximation levels* $L_{\text{Approx}}$ that provide certain degrees of precision are utilized and exchanged at runtime. Such approximation levels are provided by approximate hardware designs such as [11–13] that allow to configure the underyling precision. To minimize the hardware utilization while ensuring correct results, the inherent error resilience is periodically estimated and matching approximation levels are determined. Such approximation levels are periodically evaluated by exploiting the error detection capability of *fault tolerance techniques* [9, 15, 21–23]. These fault tolerance techniques check PCG's inherent convergence invariants to detect errors, which is exploited by our technique to improve the approximation level estimation. By applying our fault tolerance technique from previous work [15], correct solver results are ensured while the hardware utilization is effectively reduced.

Figure 2 shows the steps of our technique for the PCG solver. The first step comprises the *preparation of the PCG* algorithm which corresponds to lines 1 to 5 in Algorithm 1. In the second step, the approximation level $L_{\text{Approx}}$ is periodically determined based on estimating the current error resilience in iteration $k$ as explained below in Section IV-A. Using this approximation level $L_{\text{Approx}}$, a *PCG iteration* is computed in the third step which corresponds to lines 6 to 17 in the original Algorithm 1. Afterwards, the approximation level is evaluated periodically in the fourth step by a fault tolerance technique for PCG such as [15, 21–23]. If the approximation level is too aggressive (i.e. the underlying precision is unsuitable for solver convergence), PCG's inherent convergence properties are violated, which is detected by the fault tolerance technique as an *error*. In such a case, an offset is introduced in the fifth step which ensures that PCG iterations are continued using lower approximation levels with increased precision. Besides, a valid solver state is recovered by performing the *error correction* scheme of the fault tolerance technique. This step is explained further below in Section IV-C.

### A. Approximation Level Determination

The proposed estimation scheme is based on the insight, that the error resilience is closely related to the update

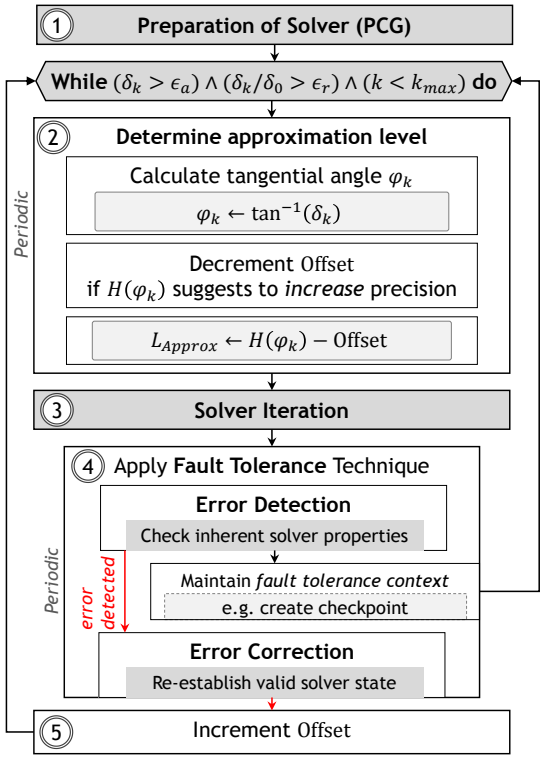


Figure 2. Overview of our technique for the Preconditioned Conjugate Gradient Solver (PCG).

vector length $||u_k||_2$ (i.e. $||x_k - x_{k-1}||_2$) for each iteration $k$ which may be highly variable during runtime. Such update vectors $||u_k||_2$ are typically large during the first PCG iterations and approach zero when PCG converges to the solution (cf. Section II). The *directions* of successive update vectors must be $A$-orthogonal (i.e. $u_k A u_{k+1} = 0$) to ensure correct convergence of PCG [9]. The PCG method is less sensitive to approximation errors for large update vectors $u_k$ as the *update direction* is often only marginally altered by approximation errors and therefore allows the utilization of approximation levels with less precision and reduced hardware utilization. At the same time, PCG is becoming increasingly sensitive for smaller update vectors since the update direction is now increasingly altered by approximation errors which can *violate* the $A$-orthogonality between successive update vectors.

Instead of directly evaluating the update vector length, which requires additional inner products, our method calculates the *tangential angle* $\varphi$ in the intermediate result $x_k$ which constitutes the update vector length $||u_k||_2$ with

$$\varphi := \tan^{-1}(\delta_k), \quad \varphi \in [0°, 90°[ \tag{1}$$

and $\delta_k$ being the norm of the residual vector $r_k$. This operation only requires a single additional scalar operation (i.e. $\tan^{-1}$). For large angles $\varphi$, the update vector length $||u_k||_2$ is large while it becomes smaller when PCG converges as $\varphi$ approaches $0°$. Our proposed technique calculates the angle $\varphi$ periodically to estimate the error resilience for the succeeding iterations and uses the function $H(\varphi)$ to determine a suitable approximation level $L_{\text{Approx}}$. The

function $H(\varphi) : \{[0°, 90°[ \rightarrow \mathbb{N}\}$ maps angles $\varphi$ to the set of approximation levels $L_{\text{Approx}}$. Here, we consider $H(\varphi)$ being a user-defined lookup table. The calculation of both $\varphi$ and $H(\varphi)$ induces very low performance and energy overheads since only scalar operations are required.

### B. Relation between angle $\varphi$ and update vectors $u_k$

As mentioned before, the error resilience of the PCG execution is constituted by the update vector length ($||u_k||_2$) which is closely related to tangential angle $\varphi$. This correlation is based on the insight that solving a system of linear equations $Ax = b$ (using e.g. PCG) is equivalent to finding the minimum of its quadratic form $E(x)$ [14]:

$$\min_x E(x) := \tfrac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle.$$

For each intermediate result $x_k$, a tangential angle $\varphi$ can be derived from the gradient $\nabla E|_{x_k}$ in $x_k$. Figure 3 presents an example for a two dimensional system of equations. The tangential angle $\varphi$ is calculated from the gradient $\nabla E|_{x_k}$ in $x_k$ which is in case of PCG $-r_k$ [14] with

$$\varphi := \tan^{-1}(\nabla E|_{x_k}) = \tan^{-1}(||-r_k||_2) = \tan^{-1}(\delta_k).$$

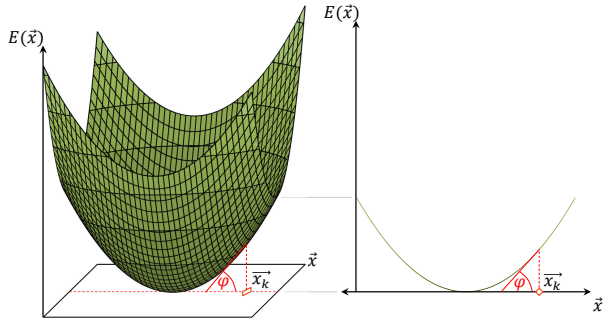


Figure 3. Quadratic function $E(x)$ with a tangential angle $\varphi$ in the intermediate result $x_k$.

### C. Using Fault Tolerance to Evaluate Estimated Error Resilience

Since the determined approximation level $L_{\text{Approx}}$ can be too *aggressive*, a fault tolerance technique is periodically evaluated which detects and corrects deviations (i.e. errors) being too harmful for the solver execution. Fault tolerance techniques which are tailored for the PCG method evaluate certain invariants [9] between the PCG variables that must be satisfied for correct convergence throughout the whole execution. As explained before, our fault tolerance technique [15] ensures correct solver results while effectively reducing the hardware utilization. Such reductions are ensured by the underyling efficient error detection scheme that avoids expensive operations. An offset for the current approximation level is introduced if the fault tolerance technique detects errors. This offset ensures the use of approximation levels with increased precision. At the same time, this offset is reduced if the result of the mapping function $H(\varphi)$ suggests an approximation level with increased precision.

## V. Experimental Results

The proposed technique for PCG is evaluated with respect to *required PCG iterations* for correct convergence and *reductions in hardware utilization* for three different fault tolerance techniques. Besides, our approach is compared with a *static technique* that does not change the precision at runtime.

The following fault tolerance techniques are evaluated and compared: The first method is the approach from previous work [15]. The second method is the *periodic correction of the residual* which is proposed in different degrees in [22] as well as in [21]. The third method is the *periodic evaluation of orthogonality and residual relationships* which is proposed in [23].

### A. Hardware Model

In the experiments, the behavior of approximate hardware was modeled in software. This model performs approximate arithmetic operations based on concatenating *precise* result bits with uniformly random bits, which model approximately computed bits. Different *approximation levels* (i.e. degrees of precision) were provided at runtime by changing the number of approximated bits. Higher approximation levels use increasing numbers of approximated bits (and vice versa).

The approximation was applied to the most energy-demanding arithmetic operation in the dominant operation in PCG, namely the floating-point multiplications [12] in sparse matrix-vector multiplications (SpMV). To approximate such floating-point multiplications, the least significant bits of the result mantissa were approximated. Assuming an $N$-bit multiplier using carry-save logic with $N(N-1)$ binary adders [24], $k(2N-1)-k^2$ adders can be deactivated (i.e. $N(N-1)-(N-k)((N-k)-1)$) by approximating the $k$ least significant bits in the result value.

To compare reductions in *hardware utilization*, the number of required active binary adders was determined for each PCG iteration according to the underlying approximation level and accumulated to obtain the hardware utilization for complete PCG executions. In this work, the term *precise hardware* refers to a IEEE 754-compliant floating-point unit.

### B. Experimental Setup

To accelerate the experiments, the PCG algorithm and the approximate hardware model were mapped to a heterogeneous computing system comprising multi-core CPUs and many-core GPUs. At runtime, the floating-point multiplications in the SpMV operation were replaced by their approximate counterparts. All parallelizable linear algebra operations were mapped to a GPU architecture and GPU-accelerated linear algebra libraries were utilized. All experiments have been performed in double precision.

The following experimental parameters were utilized: For all experiments, a random vector was generated for the initial guess $x_0$, 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 execution of PCG was continued until $\delta_k$ fell below the *absolute* error tolerance

$\epsilon_a$ or $\delta_k/\delta_0$ fell below the *relative* error tolerance $\epsilon_r$. The absolute error tolerance $\epsilon_a$ was set to $10^{-6}$ and the relative error tolerance $\epsilon_r$ was set to $10^{-15}$. An experiment was considered a failure, if the number of iterations exceeded $10 \cdot N$ iterations. To avoid false identification of convergence due to accumulation of rounding errors in $\delta_k$, the final result was checked against $r_k \approx (b - Ax_k)$. In case of a violation, the true residual $r_k$ (i.e. $:= b - Ax_k$) was reconstituted and PCG was continued as discussed in [25]. The error detection threshold used in the comparison of floating-point values was set to $10^{-7}$. Approximation levels were determined each 20 iterations and evaluated by the fault tolerance each 10 iterations. The context of each fault tolerance method (e.g. checkpoint generation) was updated each 20 iterations as proposed in [15, 22, 23]. The function $H(\varphi)$ was defined to map the angles $\varphi$ to five approximation levels as follows: For the highest approximation level, 35 operand bits were approximated (i.e. multiplication using 17 precise bits). Lower approximation levels provide increased precision with linearly increased number of precise operand bits. The lowest approximation level provides full machine precision (i.e. 52 bits). The highest approximation level was used for tangential angles $\varphi > 25°$. The hardware platform consists of two Intel Xeon E5-2623 and 128 GB RAM with three Nvidia Tesla K80 GPUs and 24 GB GDDR5 RAM per device. More than 200,000 experiments were performed to obtain the results presented below. Each experiment was executed using a combination of a single CPU core and a single GPU.

### C. Benchmarks

As benchmarks, 14 matrices from the Florida Sparse Matrix Collection [26] were used which are shown in Table I. Besides the names and sizes of the matrices ($N \times N$), the number of nonzero elements (NNZ) is presented. As a side information, the portion of zeros within the matrices is shown. The matrices have the following properties: square, symmetric, real and positive definite. The evaluated matrices were stored in the compressed sparse row format [27].

Table I
OVERVIEW OF EVALUATED MATRICES FROM THE FLORIDA SPARSE MATRIX COLLECTION [26].

| Name | N | NNZ | Portion of 0s |
|---|---|---|---|
| nos3 | 960 | 15844 | 98.28% |
| bcsstk10 | 1086 | 22070 | 98.13% |
| msc01050 | 1050 | 26198 | 97.62% |
| nasa2146 | 2146 | 72250 | 98.43% |
| sts4098 | 4098 | 72356 | 99.57% |
| bcsstk13 | 2003 | 83883 | 97.91% |
| ex9 | 3363 | 99471 | 99.12% |
| bodyy4 | 17546 | 121550 | 99.96% |
| bodyy5 | 18589 | 128853 | 99.96% |
| bodyy6 | 19366 | 134208 | 99.96% |
| Muu | 7102 | 170134 | 99.66% |
| bcsstk16 | 4884 | 290378 | 98.78% |
| msc10848 | 10848 | 1229776 | 98.95% |
| nd3k | 9000 | 3279690 | 95.95% |

### D. PCG Iteration Overhead

The use of approximate hardware may induce some additional PCG iterations required for convergence to a correct result. Figure 4 shows the additional iterations compared to PCG executions on precise hardware. All evaluated experiments converged to a correct result. The evaluated matrices are ordered by the number of non-zero elements. The iteration overhead is on average 9.5% and ranges from 0.0% to 28.5% when the fault tolerance technique from [15] is applied. In comparison, the iteration overhead is on average 83.6% lower compared to the approach in [23] and 93.3% lower compared to the approach in [21, 22]. The comparatively low overhead can be explained by the increased error coverage of that fault tolerance technique which ensures the use of suitable approximation levels.

### E. Reduction of Hardware Utilization

To evaluate potential energy savings gained by our approach, the hardware utilization (i.e. the number of active binary adders during the execution of PCG) was determined. As explained before, the active binary adders were counted for each PCG iteration and accumulated for complete PCG executions to determine the overall hardware utilization. In Figure 5, the hardware utilization of our approach is compared to the hardware utilization when precise hardware is used. Positive results represent reductions in total hardware utilization while negative results indicate overheads. Although the number of iterations is often increased with approximate hardware, reductions in hardware utilization can be observed for 11 out of 14 matrices when fault tolerance technique [15] is applied. In these 11 cases, the reduction of hardware utilization is on average 14.5% and ranges from 3.1% to 41.0%. Reductions in total hardware utilization can be observed for one matrix for the approaches in [21–23]. The reductions of hardware utilization for fault tolerance technique [15] can be explained by the efficiency of this technique, which avoids the periodic computation of expensive sparse matrix operations. Figure 6 shows the adaption of approximation levels along with the number of precise bits during two example PCG executions.

### F. Evaluation for Fixed Precision

We evaluated PCG executions using fixed precision. In this setup, five operand bits were approximated (i.e. multiplication using 47 precise bits) for each floating-point multiplication during the SpMV operation. For 9 out of 14 matrices, PCG never converged to a correct result. For the remaining five matrices (*nos3, bodyy4, bodyy5, Muu,* and *nd3k*), the hardware utilization was reduced on average by 11.2% (ranging from 8.7% to 17.3%) compared to execution on precise hardware.

## VI. CONCLUSION

Efficient linear system solvers are an important task for many scientific and engineering applications. However, their energy demands require suitable approximate computing techniques to overcome future energy challenges in exascale

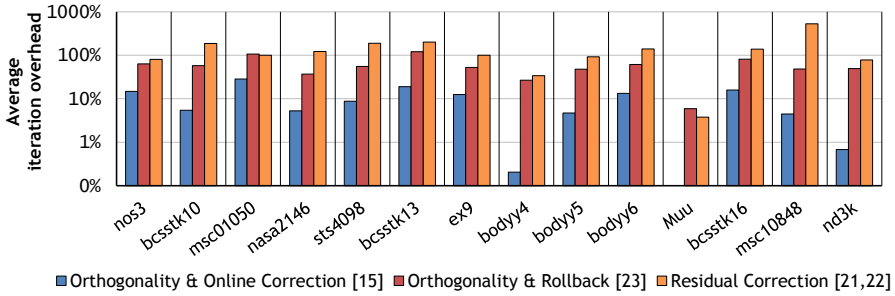


Figure 4. Average iteration overhead compared to PCG executions on precise hardware.

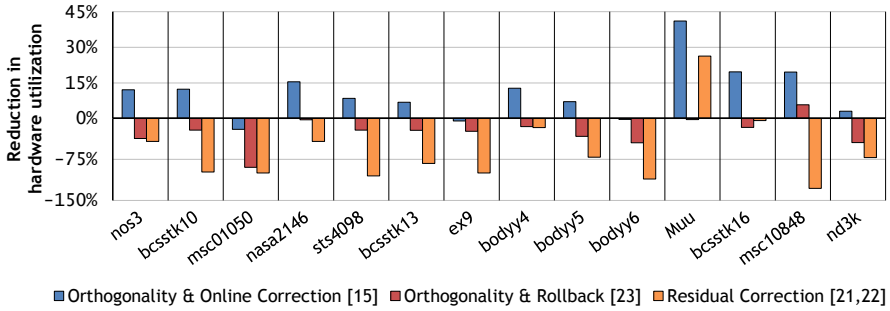


Figure 5. Reduction in hardware utilization compared to execution on precise hardware.

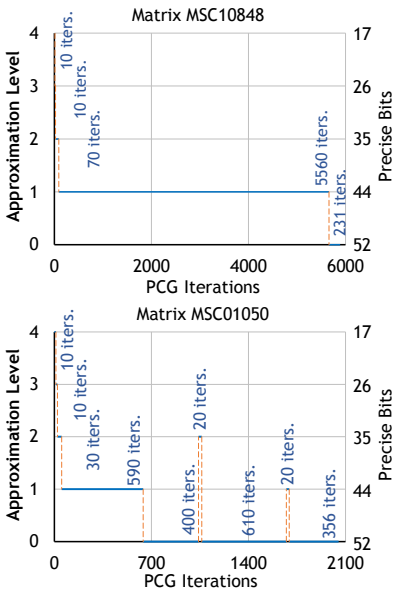


Figure 6. Adaption of approximation levels over time for two input matrices $A$.

computing. In this work, we presented a new technique that allows the execution of the PCG solver on approximate hardware to gain reductions in hardware utilization while still providing correct solver results. This technique periodically estimates the error resilience and uses different approximation levels to ensure suitable energy-precision trade-offs. By applying an efficient fault tolerance technique to evaluate the estimated error resilience, the hardware utilization is reduced while correct solver results are ensured. Experimental results show that, when compared to executing PCG on precise hardware, the number of iterations is only increased on average by 9.5% while the complete hardware utilization is on average reduced by 14.5%.

## Acknowledgment

The authors 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] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm for Energy-efficient Design", in *Proc. 18th IEEE European Test Symposium (ETS'13)*, Avignon, France, May 2013, pp. 1–6.

[2] S. Venkataramani *et al.*, "Approximate Computing and the Quest for Computing Efficiency", in *Proc. 51st ACM/EDAC/IEEE Design Automation Conference (DAC'15)*, San Francisco, CA, USA, Jun. 2015, pp. 1–6.

[3] T. Moreau *et al.*, "SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration", in *Proc. IEEE Intl. Symp. on High Performance Computer Architecture (HPCA)*, San Francisco, CA, Feb. 2015, pp. 603–614.

[4] V. Chippa *et al.*, "Analysis and Characterization of Inherent Application Resilience for Approximate Computing", in *Proc. 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13)*, Austin, Texas, May 2013, pp. 1–9.

[5] Q. Zhang *et al.*, "ApproxIt: An Approximate Computing Framework for Iterative Methods", in *Proc. 51st ACM/IEEE Design Automation Conference (DAC'14)*, 2014, pp. 1–6.

[6] I. Smith, D. Griffiths, and L. Margetts, *Programming the Finite Element Method*, 4th ed. Wiley, Oct 2013.

[7] D. Yuen *et al.*, *GPU Solutions to Multi-scale Problems in Science and Engineering*, 8th ed., ser. In Earth System Sciences. Springer, 2013.

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

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

[10] B. Dally, "Power, Programmability, and Granularity: The Challenges of Exascale Computing", in *Proc. IEEE International Test Conference (ITC)*, Anaheim, CA, 2011, pp. 1–12.

[11] A. B. Kahng and S. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", in *Proc. ACM/IEEE 49th Design Automation Conference (DAC'12)*, San Francisco, CA, Jun. 2012, pp. 820–825.

[12] H. Zhang, W. Zhang, and J. Lach, "A Low-Power Accuracy-Configurable Floating Point Multiplier", in *Proc. IEEE Intl. Conf. on Computer Design (ICCD)*, Seoul, South Korea, Oct. 2014, pp. 48–54.

[13] C. Liu, J. Han, and F. Lombardi, "A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery", in *Proc. Conference on Design, Automation & Test in Europe (DATE'14)*, Dresden, Germany, Mar. 2014, pp. 95:1–95:4.

[14] W. Gander, M. Gander, and F. Kwok, *Scientific Computing*, ser. Computational Science and Engineering. Springer International, 2014, vol. 11, no. 1.

[15] A. Schöll *et al.*, "Low-Overhead Fault-Tolerance for the Preconditioned Conjugate Gradient Solver", in *Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Amherst, MA, Oct. 2015, pp. 60–66.

[16] V. Gupta *et al.*, "IMPACT: Imprecise Adders for Low-Power Approximate Computing", in *Proc. 17th IEEE/ACM Intl. Symp. on Low-power Electronics and Design*, Fukuoka, Japan, Aug. 2011, pp. 409–414.

[17] H. Jiang, J. Han, and F. Lombardi, "A Comparative Review and Evaluation of Approximate Adders", in *Proc. Great Lakes Symposium on VLSI*, Pittsburgh, PA, USA, May 2015, pp. 343–348.

[18] W. Liu *et al.*, "Design and Analysis of Inexact Floating-Point Adders", *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 308–314, Jan. 2016.

[19] V. Camus, J. Schlachter, and C. Enz, "Energy-Efficient Inexact Speculative Adder with High Performance and Accuracy Control", in *Proc. IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015, pp. 45–48.

[20] M. Schaffner *et al.*, "An Approximate Computing Technique for the Complexity of a Direct-solver for Sparse Linear Systems in Real-time Video Processing", in *Proc. 51st ACM/EDAC/IEEE Design Automation Conference (DAC'15)*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

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

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

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

[24] M. Elrabaa, I. Abu-Khater, and M. Elmasry, *Advanced Low-Power Digital Circuit Techniques*. Springer, 2012, vol. 405, no. 1.

[25] H. A. Van Der Vorst and Q. Ye, "Residual Replacement Strategies for Krylov Subspace Iterative Methods for the Convergence of True Residuals", *SIAM Journal on Scientific Computing*, vol. 22, no. 3, pp. 835–852, 2000.

[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.