Journal of Computational Science xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Fine-grained bit-flip protection for relaxation methods

Hartwig Anzt[a,*], Jack Dongarra[a,b,c], Enrique S. Quintana-Ortí[d]

[a] Innovative Computing Lab, University of Tennessee, Knoxville, TN, USA
[b] Oak Ridge National Laboratory, USA
[c] School of Computer Science, University of Manchester, United Kingdom
[d] Depto. Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, Spain

ARTICLE INFO

Article history:
Received 20 May 2016
Received in revised form
16 November 2016
Accepted 21 November 2016
Available online xxx

MSC:
15A06 (linear equations)

Keywords:
Sparse linear systems
Iterative solvers
Jacobi method
Fault tolerance
Bit flips
High performance computing

ABSTRACT

Resilience is considered a challenging under-addressed issue that the high performance computing community (HPC) will have to face in order to produce reliable Exascale systems by the beginning of the next decade. As part of a push toward a resilient HPC ecosystem, in this paper we propose an error-resilient iterative solver for sparse linear systems based on stationary component-wise relaxation methods. Starting from a plain implementation of the Jacobi iteration, our approach introduces a low-cost component-wise technique that detects bit-flips, rejecting some component updates, and turning the initial synchronized solver into an asynchronous iteration. Our experimental study with sparse incomplete factorizations from a collection of real-world applications, and a practical GPU implementation, exposes the convergence delay incurred by the fault-tolerant implementation and its practical performance.

1. Introduction

Fault resilience has been identified as a critical challenge for future supercomputers [1–3]. In particular, the growth in the number of transistors in CMOS integrated circuits, at the pace dictated by Moore's Law [4], will render a significant increase in the rate of permanent, intermittent, and transient system faults (e.g., due to the impact of alpha particles or radiation from chip packaging) [3]. This may eventually yield current checkpoint+restart methods too costly for large-scale high performance computing (HPC) systems, necessitating alternative fault tolerance mechanisms that ensure *reliable* exascale machines [3]. Without this protection, applications will not run to completion, or even worse, they will suffer soft errors (or silent data corruption, SDC), thereby returning an incorrect answer without any indication of error.

In this paper we consider the solution of sparse linear systems of equations via (resilient versions of) component-wise relaxation (i.e., stationary) solvers [5]. Despite the often superior convergence characteristics of Krylov methods, these component-wise iterations remain an important building block as smoothers for multigrid methods [6], and for the approximate solution of sparse triangular systems within an incomplete factorization preconditioner [7,8].

Recently, we introduced an algorithmic-based fault tolerance (ABFT) technique that enhances the natural resilience of stationary methods with respect to transient bit-flips (a class of SDC) [9]. Two key properties of our ABFT solution for the Jacobi method [5], compared with a similar approach analyzed in [10], are: (1) we employ a component-wise bit-flip protection mechanism; and (2) our protection technique is based on a low-cost error detection mechanism, which rejects the update of a component of the solution if the update is considered erroneous. In case of rejected updates, the original Jacobi method becomes an asynchronous solver. Nonetheless, our fault tolerance technique preserves the convergence of the iteration in case high bit-flip rates. The initial evaluation of the method in [9], with (the finite difference discretization of) the Laplace problem and four cases from the University of Florida Matrix Collection.[1] (UFMC), exposed the efficiency of the approach as well as the crucial role of the error threshold δ on the

* Corresponding author.
E-mail addresses: hanzt@icl.utk.edu (H. Anzt), dongarra@icl.utk.edu (J. Dongarra), quintana@uji.es (E.S. Quintana-Ortí).

[1] Available at https://www.cise.ufl.edu/research/sparse/matrices/ Last accessed: May 2016.

http://dx.doi.org/10.1016/j.jocs.2016.11.013
1877-7503/

number of detected bit-flips (DBF) and missed bit-flips (MBF), the number of false positives (FPs), and, ultimately, on the convergence rate of the iteration.

In this paper we extend the results presented in [9], making the following new contributions:

- We further refine the analysis by inspecting the impact of the error protection mechanism when applied every 2–5 iterations, instead of when applied at every iteration as done in [9]. We note that this relaxation increases the practical value of the ABFT solution by reducing the overhead of the error protection mechanism.
- We considerably augment the collection of real-world cases employed in the experimental validation of the approach, including a larger collection of cases from the UFMC benchmark collection.
- We provide strong evidence of the practical benefit of the approach by turning it from mostly a theoretical analysis, based on a MATLAB code, into an actual HPC code for graphics processing units (GPUs) built on top of MAGMA-sparse [11].
- We elaborate on the interaction between voltage scaling, energy efficiency, and extended runtime due to bit-flip errors. This is particularly important as near-threshold voltage computing (NTVC) [12] has been recently proposed as a means to tackle the power wall, allowing for integration of additional levels of hardware concurrency at the expense of more transistor area and large volumes of systems faults [13].

There are two sources of overhead for our fault-tolerant version of the Jacobi iteration: (1) the overhead introduced by the bit-flip protection mechanism; and (2) the potential increase in the number of iterations due to MBFs that corrupt the solution vector or to DBFs that result in missed updates on the solution vector. The bit-flip protection mechanism requires a few operations on vectors that can be easily vectorized. With some re-organization effort, these operations can be also pipelined, to reduce off-chip memory accesses, and their computational cost can be hidden by overlapping them with the more expensive iteration. This motivates us to focus our cost analysis on evaluating the second cause of overhead.

The rest of the paper is structured as follows. In Section 2 we review related research efforts that also analyze fault tolerance/resilience in the solution of sparse linear systems. In Section 3 we offer a brief introduction to stationary solvers, and in Section 4 we summarize the fault model, bit-flip injection methodology, and resilience-related metrics. In Section 5 we describe our fault tolerance technique for low-cost error resilience in this type of method. In Sections 6 and 7, we investigate the effect of bit-flips on the convergence rate and the practical performance of an ABFT GPU implementation, respectively. In Section 8 we explore the trade-off between undervolting and energy savings. We conclude in Section 9 with a few remarks.

2. Related work

Several recent works have considered the effect of soft errors, in most cases focusing on data corruption and (fundamental kernels for) iterative Krylov subspace methods [5]. For example, Bronevetsky and de Supinski [14] analyze the vulnerability to SDC of Krylov-based solvers. Chen [15] introduces an on-line verification of orthogonality and residual to detect soft errors during the execution of Krylov subspace solvers. Bridges et al [16] apply selective reliability to assemble a fault-tolerant version of GMRES, furnished with an inner-outer iteration, that converges at a rate that degrades with the fault rate. Elliott, Hoemmen, and Mueller [17] present a low cost fault detection mechanism for GMRES, and investigate the connection between errors in the IEEE 754 representation of real numbers, the dot product kernel, and the effect of normalizing the data [18]. Sao and Vuduc [19] depart from previous work by adopting "self-stabilization" to obviate the need for full state saving and fault detection for CG.

Compared with these efforts, we also address the iterative solution of sparse linear systems, but consider the error resilience of relaxation methods instead of Krylov subspace-based solvers. We note that our results are orthogonal and complementary to the analysis performed by Calhoun, Snir, Olson, and Garzaran [20] on how soft errors propagate through a sparse matrix-vector multiply (`SpMV`).

3. Stationary methods

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is sparse, $b \in \mathbb{R}^n$ is the right-hand side vector, and $x \in \mathbb{R}^n$ is the sought-after solution. Stationary solvers apply component-wise relaxation principles, iteratively updating each individual component of an approximated solution $x^{\{k\}}$. A popular example is the Jacobi iteration which, for a starting solution guess $x^{\{0\}}$, can be formulated as:

$$x^{\{k\}} := D^{-1}\left(b - (A - D)x^{\{k-1\}}\right) = D^{-1}b + Mx^{\{k-1\}}, \quad k = 1, 2, \ldots, \tag{1}$$

where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the diagonal entries of A [5].

An appealing property of the "Jacobi update" $D^{-1}b + Mx^{\{k-1\}}$ is that all components of $x^{\{k\}}$ can be obtained in parallel, as the arithmetic operations on any component $x_i^{\{k\}}$ only involve values from the previous iterate $x^{\{k-1\}}$.

If an implementation of the Jacobi iteration does not update all components, but some of them keep the value from the previous iteration, the method becomes asynchronous (or "chaotic") in the sense that, at a certain point during the iteration process, some components may differ in the number of times they have been updated. This is equivalent to an asynchronous iteration where each component is modified in an arbitrary manner, but always using the latest available values for the remaining components [21,22]. Convergence of the synchronous Jacobi is guaranteed if the spectral radius of the iteration matrix M fulfills

$$\rho(M) = \rho\left(I - D^{-1}A\right) < 1.$$

The asynchronous iterations require a stronger property of the iteration matrix to guarantee convergence, i.e. the spectral radius of the component-wise positive iteration matrix has to be smaller one [22]. Nonetheless, both these criteria are, for example, fulfilled for triangular matrices. Hence, both methods are suitable for approximate triangular solves in the context of incomplete factorization preconditioning [7,8].

The ABFT method we introduced in [9] builds upon an implementation of the Jacobi method, rejecting the update of those locations (components) where a low-cost error detection test indicates that a bit-flip occurred during the computation. In case of rejected updates, this turns the initial Jacobi solver into an asynchronous iteration, and as previously mentioned, imposes stronger convergence conditions on the iteration matrix. In the remainder of the paper we will refer to the bit-flip protected solver with the term *Fault-Tolerant Jacobi* (`FTJacobi`).

4. Fault model, errors, and metrics

4.1. Unreliable floating-point operations and data

We target the following scenario:

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

1. The floating-point register file and floating point units (FPUs) are unreliable, for example, due to the application of undervolting or because they are built from faulty (but energy-efficient) technology [23].
2. The cache and the main memory are reliable but, in order to save energy, feature no error protection mechanism (e.g., ECC). All data initially resides in the main memory and is correct. Loading an element from the cache or the main memory will fetch a correct value as long as it has not been overwritten by a corrupted (incorrect) result produced by the FPUs in a previous stage.
3. The computation of the bit flip mechanism is reliable, for example, because it is deviated to non-faulty hardware operating in a safe voltage scale.
4. All remaining data (integers, instructions, control, etc.) is reliable.

When running the Jacobi relaxation method under these conditions, it is necessary to test every update to the solution vector, in order to detect whether its computation was affected by a bit-flip. Precisely, the updated values in the floating-point registers need to be checked for correctness before they are pushed back to main memory. We address the realization of this bit-flip detection mechanism in Section 5.

4.2. Bit-flip injection methodology

In order to simulate the previously introduced scenario, we use the following injection strategy to artificially introduce bit-flips:

- We assume every iteration of the Jacobi method is affected by κ bit-flips.
- At the beginning of each iteration of the Jacobi method, single bit-flips are inserted into κ random positions of the un-corrupted positive iteration matrix M before computing the SpMV involving this matrix; see (1).
- As we want to simulate the scenario of transient errors affecting data that lies in the floating-point register file or is computed by the FPUs, we afterwards restore the matrix values in preparation for the next iteration.

We emphasize again that this reflects a scenario where the results of κ floating-point operations involving elements of M are corrupted, but the source of the data (e.g., in main memory or certain level of the cache) remains correct. Also, a single bit-flip in the k-th iteration will corrupt, at most, a single entry of x^k. However, with κ errors in M, the number of entries corrupted in x^k equals the number of different rows of M these errors affect, which can be smaller than κ.

4.3. Assessment criteria

We employ the following quantitative "metrics" to expose the properties of the fault tolerance technique:

- DBFs and MBFs: Obviously, we would like to detect κ bit-flips per iteration and miss none. In practice, because the bit-flips are detected by checking the result x^k, the maximum number of bit-flips that can be detected may be smaller than κ, as two or more errors in the same row of M propagate to the same entry of the result vector, corrupting only a single entry of it. Also, the effect of a bit-flip may be small enough to be within the threshold or covered by the noise of IEEE rounding.
- FPs: Detecting a false bit-flip in an entry of x^k implies that this entry is not updated during the current iteration and may slow down convergence. Therefore, we would like to keep the number of FPs low.
- Relative convergence delay μ. If bit-flips are detected, and the respective values are not pushed back to main memory, the convergence progress of the method is delayed. We define a metric μ to measure this effect as the ratio between the number of iterations required by a bit-flip protected method, to reach a relative residual bounded by τ in the presence of κ errors per iteration, and the number of iterations required by an un-protected method to reach the same residual threshold when no errors occur. Ideally, we would like the delay ratio to be close to 1, which is obviously impossible if we neglect some bit-flip affected computations.

5. Bit-flip protection for Jacobi

Synchronous relaxation methods in general, and the Jacobi method in particular, exhibit the convenient property of a monotonic residual decay [5]. Consider the relation between the residual $r^{\{k\}} = b - Ax^{\{k\}} \in \mathbb{R}^n$, at an iterate k, and the distance (error) between the approximation $x^{\{k\}}$ at that step and the exact solution $x^{\{*\}}$:

$$r^{\{k\}} = b - Ax^{\{k\}} = A\left(x^{\{*\}} - x^{\{k\}}\right). \tag{2}$$

For Jacobi, the residual decreases linearly with the iteration count, up to convergence in appropriate floating-point format. Hence, a straight-forward strategy for detecting errors occurred during the Jacobi iteration consists of checking for a monotonic decrease of the residual vector, and only accepting the new solution vector if it passes this test. Unfortunately, this approach is computationally expensive, as the residual computation (which can also be affected by errors) has about the same cost as the iteration itself. Furthermore, the method will not make any progress in cases of high bit-flip rates.

5.1. Component-wise bit-flip protection

In addition to the monotonic convergence, the Jacobi iteration also fulfills the contraction property of fixed-point iterations at the component level, which states that the difference between the current and last iterate decreases component-wise:

$$\begin{aligned} &\forall i \in [1,n], \quad \exists\ 0 < \theta_i < 1 \quad : \\ &[0.07in] \left|x_i^{\{k\}} - x_i^{\{k-1\}}\right| \le \theta_i \left|x_i^{\{k-1\}} - x_i^{\{k-2\}}\right| \le \theta_i^2 \left|x_i^{\{k-2\}} - x_i^{\{k-3\}}\right| \dots \\ &\equiv z_i^{\{k\}} \le \theta_i \ \ z_i^{\{k-1\}} \le \theta_i^2 \ \ z_i^{\{k-2\}} \dots . \end{aligned} \tag{3}$$

This allows replacing the costly residual test with a component-wise convergence test, based on the expected difference from the next iterate. For relaxation methods with a linear convergence rate, for a specific problem, the component-wise ratio:

$$c_i := \frac{z_i^{\{k-1\}}}{z_i^{\{k\}}}, \quad k = 2, 3 \dots \tag{4}$$

remains constant up to convergence in the respective format. This suggests one should compute c_i in some reliable mode, and exploit the tolerance-based estimation for the difference

$$\left| \frac{z_i^{\{k-1\}}}{z_i^{\{k\}}} - c_i \right| \le c_i \cdot \delta, \tag{5}$$

for some user-defined threshold δ, as (part of) a component-wise error detection mechanism. Thus, if this condition is not fulfilled, it might be an indicator that an error has occurred, and the update to obtain $x_i^{\{l\}}$ should be rejected. Hereafter we will refer to the test (5) as the *threshold condition* (T-COND).

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

Despite being a good indicator for the practical case of limited precision, the test T-COND is not perfect. In particular, the condition is not able to detect all errors since bit-flips with a small effect on the component value can "slip" through the threshold. This is acceptable, as errors of such magnitude will likely yield no serious harm to the convergence rate of the relaxation method. On the other hand, a failure to pass the test due to causes other than an actual bit-flip error can result in false positives (FPs). Possible causes are rounding effects either in the derivation of the contraction constants or in the fault detector, as well as delayed updates of some other components. These FPs may not cause the divergence of the iteration, but can still result in stagnation of the convergence. The threshold parameter δ provides a mechanism to balance these two effects.

A central question is how the bit-flip protection strategy handles a negative evaluation of T-COND. If the strategy rejects the complete Jacobi update, because (almost) any iteration carries a corruption, an FP, or both, this option will result in stagnation. Hence, it is beneficial to update only those components where the condition is fulfilled, while rejecting the changes to locations violating T-COND. However, accepting some component updates, but rejecting others, destroys the synchronism of the relaxation method. As a consequence, if a component update is rejected, the linear decay of the difference between the last and the new iteration solution approximations will no longer be guaranteed for the remaining components. Instead, components strongly depending on those entries which were not updated may exhibit a small increase of this quantity in the current iteration. These will expose themselves as FPs during the next iteration. Therefore, to also ensure progress in the case of FPs, we propose to extend the strategy by a second condition that becomes relevant if a component was not updated during the previous iteration(s). Concretely, a flag ($flag_i^{fp}$) accumulates how many consecutive times a certain component has not been updated, and the component-wise update is accepted if the *false-positive condition* (FP-COND) indicates that the component value does not explode:

$$\frac{z_i^{\{k-1\}}}{z_i^{\{k\}}} > 10^{-flag_i^{fp}} \tag{6}$$

with a certain $\phi \geq flag_i^{fp}$. This bound should be chosen according to the condition number of the system matrix and the sought-after relative residual reduction. Specifically, a higher bound in (6) permits convergence to a very small relative residual, with the danger of residual explosion for ill-conditioned systems. A tighter bound prevents residual explosion at the cost of potential stagnation after a certain residual reduction. In practice, we identified $flag_i^{fp} \leq \phi = 10$ to be a good choice for many problems, as this yields convergence to machine precision at the risk of a residual increase of—at most—ϕ orders of magnitude multiplied by the last difference $z_i^{\{k-1\}}$. We notice that although this strategy succeeds for the 55 test cases (respectively 110 incomplete triangular factors) we consider in this paper, there may exist problems for which the mechanism has to be set differently to detect false positives.

5.2. Implementation details

Fig. 1 offers a practical implementation of the bit-flip protection mechanism in MATLAB®. The function inputs correspond, in that order, to the problem dimension n, the threshold δ, the convergence component-wise ratio c, the values $x^{\{k-1\}}$ and $z^{\{k-1\}}$ from the *previous* iterate (xprev and zprev), the *current* $x^{\{k\}}$ (xcur), and values for flags associated with T-COND and FP-COND. After performing the appropriate operations, the function updates only those entries that pass the test, overwriting the "previous" xprev and zprev in preparation for the next iteration. This particular implementation exposes the favorable properties of the bit protection mechanism for vectorization and execution on an SIMD unit or a data-parallel architecture. The code involves 12 component-wise operations on vectors of length n, plus a few masked array updates (e.g., `flag_fp(fpcond) = 0`). Note in particular that this is also true for the max and min computations, which apply this operator component-wise.

We emphasize that our approach assumes a reliable computation of c, to ensure that the iteration has proceeded beyond the increase in the relative residual that can occur at the beginning of an asynchronous iteration.

In practice, we can ensure the precision of c by, for example, repeating the first few iterations of the Jacobi method, performing them on reliable hardware/mode, or use a spectral analysis for a more accurate derivation of the contraction constants.

In practice, the Jacobi method is rarely used as solver, iterating until convergence. Instead, it is rather used as a smoother or a preconditioner within a more sophisticated solver. In such settings, a few Jacobi sweeps are repeatedly applied, e.g., in every preconditioned application. The distinct preconditioned applications are likely to differ with respect to the right-hand side, as well as how close the initial guess is to the solution. Although the convergence rate mildly depends on the right-hand side vector and the initial solution guess, performing the first preconditioner application in reliable mode is likely to provide contraction constants that can be used for the detection mechanism in all subsequent preconditioner applications that are then realized in unreliable mode. Analyzing a high number of relaxation steps for a complete convergence analysis comprises different scenarios with respect to the quality of the initial guess, and allows for easier illustration of the effects encountered.

To conclude this discussion of practical aspects, let us inspect the update of $flag_{fp}$. Define $d_i = z_i^{\{k\}}$ and assume that, in the next iteration, $z_i^{\{k+1\}}$ does not pass the test and the corresponding component is not be updated. In iteration $k+2$, the update is accepted, provided $z_i^{\{k+2\}} < 10^1 \cdot d$. If rejected again, in iteration $k+3$, the update will be accepted if $z_i^{\{k+3\}} < 10^2 \cdot d$; and so on.

6. Experimental effect of bit-flips on the convergence rate

6.1. Setup and solvers

The experiments in this section, which investigate the roles of the threshold δ, the impact of the bit-flip location, and the frequency of the bit-flip protection test, were all carried out using MATLAB (release R2016a) and IEEE 754 double-precision (64-bit) arithmetic on a server equipped with two Intel Xeon E5-2670 sockets and 64 GB of RAM.

The baseline for the evaluation of our fault tolerance technique is a plain MATLAB implementation of the Jacobi iteration in (1). Obviously, this iterative scheme can be expected to experience slow convergence or even divergence in the presence of bit-flips. The fault-tolerant variant FTJacobi integrates the soft-error protection defined by (5)–(6). An implementation for the bit-flip protection is given in Fig. 1.

In both the original and the fault-tolerant codes, the iteration is stopped when the residual satisfies $\|r^k\|_2 < \tau \|x\|_2$, for a user-defined threshold τ.

6.2. Detailed evaluation of the Laplace benchmark

In order to expose some basic insights on the performance of the ABFT strategy, we initially target a finite difference discretization of the Laplace problem in 3D. The system matrix is derived from the use of a 27-pt stencil on a $16 \times 16 \times 16$ discretization. This

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

```
function [...] =
   Check_cond( n, delta, c, xprev, zprev, xcur, flag_t, flag_fp )

% Constant vectors
phi = 10;
vk  = 10.^[0:-1:-phi];

% Compute component-wise quotient
zcur   = max( abs(xcur - xprev), eps );
zratio = zprev ./ zcur;

% Evaluate T-COND
t1     = abs( zratio - c );
t2     = delta * c;
tcond  = ( t1 < t2 );

% Update FLAG_FP vector and evaluate FP-COND
flag_fp          = flag_fp + 1;
t3               = min( flag_fp, phi );
t4               = vk( t3 );
fpcond           = ( zratio > t4 );
flag_fp(fpcond) = 0;

% combine T-COND or ( flag_t == 1 & FP-COND )
ccond = tcond | ( flag_t == 1 & fpcond );

% update flag_t
flag_t         = ones(n,1);
flag_t(ccond) = 0;

% Update locations fulfilling T-COND or flag_t == 1 and FP-COND
zprev(ccond) = zcur(ccond);
xprev(ccond) = xcur(ccond);
```

Fig. 1. Bit-flip protection in MATLAB.

results in a regular system matrix with 97,336 nonzeros distributed over 4096 rows and a small condition number [5] of 9.36e+01. The purpose of the following evaluation is to assess the impact of these three factors on the metrics defining the quality of the fault tolerance technique:

- The parameter δ controlling the update in (5).
- The position of the bit-flips, which may occur in any of the 64 bits of the IEEE 754 representation of a double-precision floating-point number.
- The frequency of the bit-flip protection check f_c.

To make the effect of bit-flips more evident, in some of these cases we enforce $\kappa = 40$ errors per SpMV, an error rate that could be considered high. The experimental evaluation in the remainder of this section is based on data that is averaged over 100 repetitions of the experiments, with different random seeds in order to generate different error positions. A descriptive visualization of the bit-flip check, and more details about its effect on the relative convergence delay, can be found in [9].

6.2.1. Impact of δ

First, we explore and optimize the value for the threshold δ that balances FPs against MBFs. To illustrate the effect of this parameter, we take a "snapshot" of the iteration process at iteration 400, which, for this particular example, is somewhere in the middle of the convergence process. We analyze how many bit-flips are detected/missed and how many FPs appear at this specific iteration. The first row of plots in Fig. 2 show the number of DBFs/MBFs with a single and $\kappa = 40$ bit-flips per SpMV (left and right plots, respectively). This case motivates the selection of a small threshold δ in order to detect more errors. The plots in the second row also also include the number of FPs, which is about two orders of magnitude larger. This shows that accepting more updates, by choosing a larger threshold δ, decreases the number of FPs. In particular, in cases of a large bit-flip rate (right-hand side plots), the number of FPs dramatically decreases as $\delta \to 1$.

6.2.2. Bit-flip position

Our next experiment with the Laplace benchmark evaluates the impact of the bit-flip position in the detection properties of the bit-flip protection mechanism and the impact of MBFs as well as FPs on the convergence delay. Fig. 3 illustrates the behavior of these two metrics in four different scenarios corresponding to the bit-flips affecting: (1) the sign, (2) the exponent, (3) the lower half of the mantissa, or (4) the upper half of the mantissa. The first two rows show a significant impact on Jacobi, causing either stagnation (corruption in sign bit) or even explosion (corruption in exponent) of the residual (see left-hand side plots). The error protection mechanism of FTJacobi generally catches all these bit-flips. Only during the first iterations does this error type slip through. For the mantissa, the bit-flips become crucial (and are detected) only if the location is relevant in the convergence process. The more significant the position in the mantissa is, the higher the index of the position in the IEEE format, and the earlier the bit-flip becomes relevant for the Jacobi convergence, and the easier it is for the protection mechanism to detect this corruption. An important insight is that "small" errors in the mantissa are not important in the initial iterations, but become important later.

6.3. Relaxing the periodicity of the bit-flip protection test

Naturally, relaxing the error detection frequency f_c reduces the overhead introduced by the bit-flip protection mechanism

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

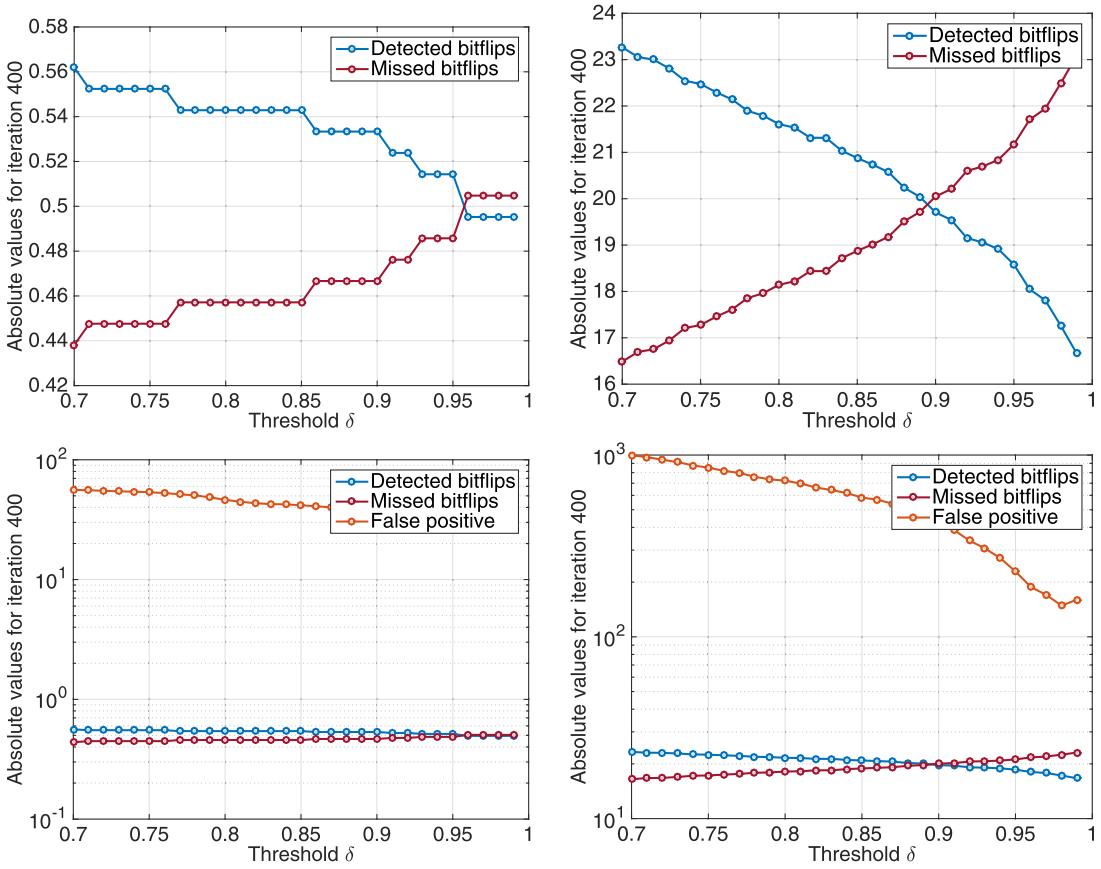

Fig. 2. DBF, MBF, and FPs for iteration 400 experienced by `FTJacobi` applied to the Laplace benchmark. $\delta \in [0.7, 1]$, $\kappa = 1$ (left) and $\kappa = 40$ (right).

compared to an un-protected Jacobi iteration. At the same time, bit-flips have a stronger impact when error checking occurs at a lower frequency: the errors propagate along the "dependencies" in the linear system, and more updates are affected and have to be disregarded. In Fig. 4 we display the behavior of the relative convergence delay μ for error checking tests performed at every iteration, every other iteration,. . ., up to every fifth iteration ($f_c = 1$, $1/2, \ldots f_c = 1/5$). We consider the two bit-flip rates $\kappa = 1$ (left hand-side plot in the figure), and $\kappa = 5$ (right hand-side plot). As expected, the convergence delay rapidly grows with an increasing bit-flip rate and a more relaxed checking frequency. At the limit, as soon as the combination of bit-flip rate and propagation exceeds the checking frequency, the bit-flip protected Jacobi fails to make progress toward the solution. Given this context, a high checking frequency should be used for strongly connected systems.

6.4. Benchmark problems

Finally, we turn our attention to a collection of real-world benchmarks to evaluate the efficiency of the bit-flip protection in a more general setting. For this purpose, we look into the approximate solution of sparse triangular systems arising for incomplete factorization preconditioners. Particularly, we consider the problems that are solved using relaxation without bit-flip protection in [8]. These arise from the incomplete LU factorization of the test matrices listed in Table 1. The matrices belong to the UFMC test suite or arise as finite difference discretization of the Laplace operator in 3D with Dirichlet boundary conditions. The sparsity plots for the test matrices are given in Fig. 5. In order to generate the incomplete LU factors, we apply Reverse Cuthill-McKee (RCM) ordering to reduce the matrix bandwidth and improve the incomplete factorizations' accuracy [24]. As an exception, we use the natural orderings for the Laplace problem and the DC test matrix as RCM brings no benefits for those two cases.

Table 2 illustrates some key characteristics of the resulting sparse triangular factors. A crucial difference with respect to the previous experiments is that, for incomplete factorization preconditioning, the sparse triangular systems do not need to be solved with high accuracy. Typically, an approximated solution reducing the initial residual by a moderate factor is sufficient to produce the same preconditioning benefits in the top-level solver as those attained from an "exact" triangular solve via forward-backward substitution [7,8]. For this reason, we investigate the performance of the bit-flip protection for the relative residual stopping criteria $\tau \in \{10^{-1}, 10^{-2}\}$.

Table 3 lists the relative convergence delay (together with other statistics) for scenarios with $\kappa = 1$ and $\kappa = 5$ bit-flips per `SpMV`. For STO, convergence is reached within the reliable iterations. Furthermore, it can be observed that in the case of 5 corruptions per update, the convergence delay for the upper triangular factor U of the DC problem is significant. The reason is that the nonzero entries in this factor are not evenly distributed, but lie all in one row. When inserting 5 bit-flips in random nonzero locations, the dense rows will almost always carry at least one corruption. Except for that case, `FTJacobi` provides satisfying convergence to the target accuracy with moderate convergence delay for all problems.

In order to validate the strategy on a larger set of problems, we conducted the same experiment with additional matrices from UFMC. Concretely, we selected 55 matrices from this collection, computed the incomplete LU factors, and applied `FTJacobi` in an SDC-prone environment where every `SpMV` is affected by $\kappa = 1$ and $\kappa = 5$ bit-flips, respectively. At this point, it is worth to pointing out that, without an experimental setup to investigate the practical behavior of the technology under distinct stressing scenarios,

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

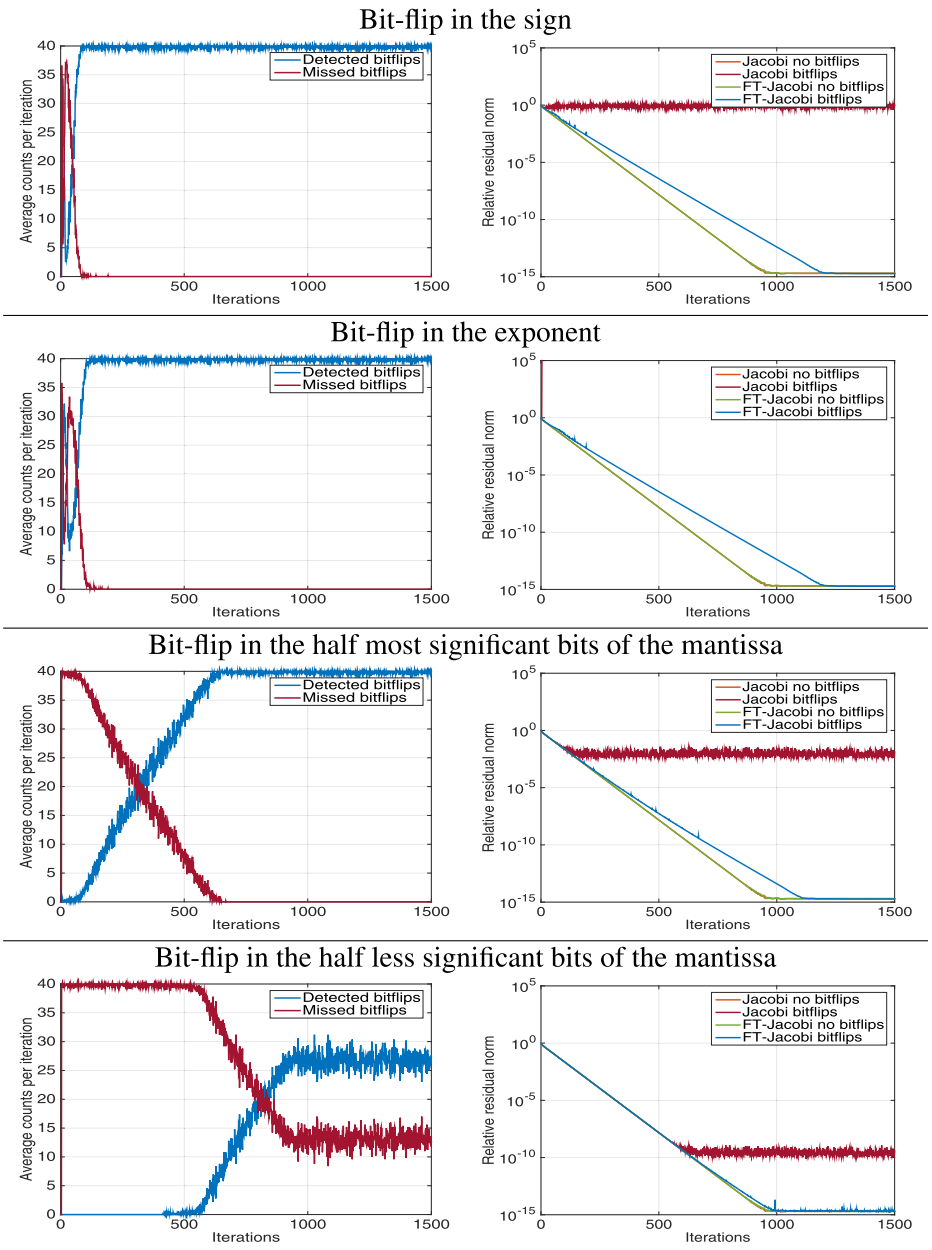

Fig. 3. DBF/MBF and convergence delay (right) experienced by `FTJacobi` applied to the Laplace benchmark. $\delta = 0.9$ and $\kappa = 40$ for different positions of the bit-flips.

Table 1
Test matrices.

	Name	Abbrev.	Description	Nonzeros n_z	Size n
UFMC	CHIPCOOLO	CHP	Convective thermal flow (FEM)	281,150	20,082
	DC1	DC	Circuit simulation matrix	766,396	116,835
	STOMACH	STO	3D electro-physical duodenum model	3,021,648	213,360
	VENKAT01	VEN	Unstructured 2D Euler solver (FEM)	1,717,792	62,424
	LAPLACE3D	LAP	3D Laplace problem (27-pt stencil)	6,859,000	262,144

it is difficult to determine whether 1/5 errors per iteration correspond to reasonable values. Therefore, we follow previous work: In [19] the authors experiment with 4 and 40 errors per iteration, but recognize that these error rates are extremely high relative to estimates in the literature. In [16], the authors considered the probability of faulty inner iterations to be in the range 0.1–0.5.

For brevity of presentation, we form the average values for DBF, MBF, and the delay μ, over all lower and upper sparse triangular systems included in the study. In Fig. 6 we visualize these averages for different relative residual stopping criteria. As could be expected from the evaluation on the bit-flip location in Section 6.2.2, DBF increases with the demanded approximation accuracy. The test suite contains some matrices with a circuit simulation origin that exhibit a high number of nonzeros in the same row of the upper triangular factor (compare with the DC problem in Fig. 5). Thus, in particular for high bit-flip rates, there is a significant chance that this row is affected by a randomly introduced bit-flip. As a result, the convergence delay of `FTJacobi` is typically higher for the upper

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

Table 2
Characteristics of the sparse triangular factors employed in the experimentation.

Matrix	Factor	Dimension	Nonzeros	Condition number
CHP	*L*	20,082	150,616	7.90e+05
	U	20,082	150,616	1.75e+11
DC	*L*	116,835	441,781	6.54e+10
	U	116,835	441,450	6.50e+09
STO	*L*	213,360	1,660,005	1.38e+07
	U	213,360	1,575,003	6.08e+07
VEN	*L*	62,424	890,108	1.85e+07
	U	62,424	890,108	2.51e+10
LAP	*L*	262,144	3,560,572	4.19e+00
	U	262,144	3,560,572	4.19e+00

Table 3
Relative convergence delay μ of `FTJacobi` applied to UFMC benchmarks for $\delta = 0.9$, $\kappa = 1$ (left) and $\kappa = 5$ (right). The number of DBPs, MBPs, and FPs is given relative to the number of actual corruptions.

		$\kappa=1$								$\kappa=5$							
Matrix		$\tau=10^{-1}$				$\tau=10^{-2}$				$\tau=10^{-1}$				$\tau=10^{-2}$			
Factor		μ	DBF	MBF	FPs	μ	DBF	MBF	FPs	μ	DBF	MBF	FPs	μ	DBF	MBF	FPs
CHP	L	1.00	12	88	4.16e+04	1.00	13	87	9.56e+04	1.00	10	90	8.31e+03	1.00	14	86	1.92e+04
	U	1.00	15	85	5.30e+03	1.00	12	88	1.16e+04	1.00	10	90	1.06e+03	1.00	12	88	2.39e+03
DC	L	1.17	66	34	6.97e+06	1.14	63	37	5.63e+06	1.17	61	39	1.39e+06	1.14	60	40	1.13e+06
	U	1.33	67	33	1.68e+06	1.57	71	29	1.80e+06	1.67	72	28	3.79e+05	1.43	72	28	3.79e+05
STO	L	1.00	*	*	*	1.00	41	59	4.36e+06	1.00	*	*	*	1.00	39	61	8.73e+05
	U	1.00	*	*	*	1.00	85	15	8.52e+06	1.00	*	*	*	1.00	84	16	1.70e+06
VEN	L	1.23	34	67	1.91e+06	1.22	41	59	2.08e+06	1.23	34	66	3.81e+05	1.22	42	58	4.15e+05
	U	1.12	34	66	1.78e+06	1.12	37	63	1.84e+06	1.12	33	67	3.56e+05	1.12	37	63	3.68e+05
LAP	L	1.00	11	89	7.20e+03	1.00	10	90	4.41e+05	1.00	5	95	1.44e+03	1.00	11	89	8.81e+04
	U	1.00	11	89	1.20e+03	1.00	11	89	3.27e+05	1.00	5	95	2.40e+02	1.00	11	89	6.99e+05

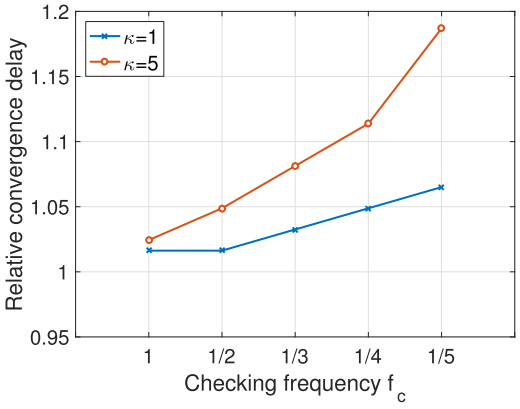

Fig. 4. Convergence delay experienced by `FTJacobi` applied to the Laplace benchmark. $\delta = 0.9$ and $\kappa = 1$ or 5 for different checking frequency f_c in the bit-flip protection mechanism.

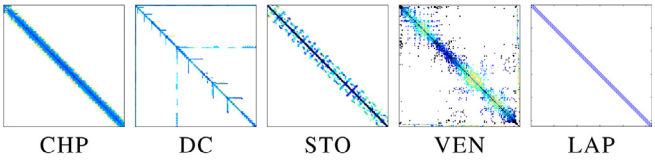

Fig. 5. Sparsity plots of test matrices listed in Table 1.

triangular systems; see right-hand side plot of Fig. 6. Comparing the delay for the two bit-flip scenarios $\kappa = 1$ and $\kappa = 5$, the higher bit-flip rate causes only a moderate increase in the convergence delay.

7. HPC implementation for GPUs

The production-code survey addressing the iterative solution of sparse triangular systems in this section employs a GPU implementation of the plain (unprotected) Jacobi solver (`Jacobi`) and a variant enhanced with the bit-flip protection, both available in the MAGMA-sparse package, part of the MAGMA open source software package [25]. The bit-flip protected `FTJacobi` differs from the plain implementation in that it integrates the bit-flip check given in (1) prior to writing the component updates back to main memory. MAGMA is compiled using CUDA and cuSPARSE in version 7.5 [26], and employs a default thread block size of 256. The GPU target architecture is an NVIDIA Tesla K40 GPU (Kepler microarchitecture) with a theoretical peak performance of 1682 (double precision) GFLOPS (billions of floating-point arithmetic operations). The 12 GB of GPU main memory can be accessed at a theoretical bandwidth of 288 GB/s. A bandwidth analysis using large data-streams reveals a realistic data access rate of about 193 GB/s [25]. The Jacobi solvers are GPU-only implementations, and all data resides in the GPU main memory. Nevertheless, for completeness we mention that the host is the same dual-socket Intel Xeon E5 architecture employed in the MATLAB analysis.

In the previous section, we evaluated the convergence process of `FTJacobi` when running in an SDC-prone environment. When applied to the sparse triangular factors arising in the incomplete LU preconditioning process, the convergence was delayed because of updates being rejected. This already makes the `FTJacobi` slower than the un-protected `Jacobi` by a factor μ. On top of this, the iteration process of `FTJacobi` is deccelerated as it performs the additional bit-flip check. In Fig. 7 we report the overhead introduced by this test in the GPU implementation of `FTJacobi` available in MAGMA-sparse. As expected, the overhead is very different for the distinct test cases. Overall, the relative overhead decreases with a rising cost of the `SpMV`. When the test is checked every other iteration, the overhead is below 33% for all problems except for the lower ILU factor of DC, where the combination of a low nonzero count and balanced distribution makes the sparse matrix-vector product inexpensive. For this case, the `SpMV` has a computational cost similar to the bit-flip check, which causes more than 90% overhead when the test is performed every iteration. The overhead of

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

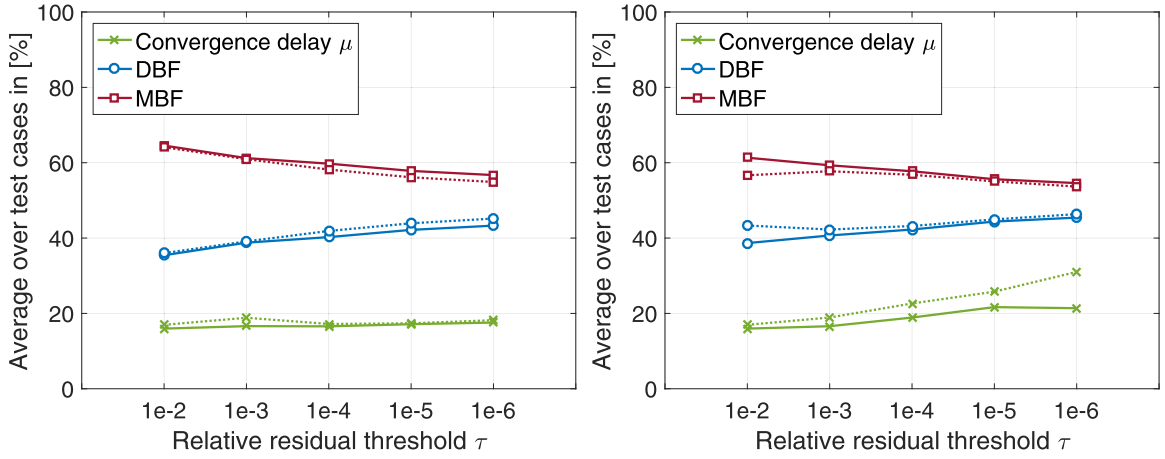

Fig. 6. Relative convergence delay μ, DBF and MBF for `FTJacobi` as average over a set of matrices from UFMC. Left side is for the scenario $\kappa = 1$, right side is for $\kappa = 5$. The solid lines represent the data for solving the lower triangular ILU(0) factors, the dashed lines are for solving the upper triangular ILU(0) factors. The matrices included have the unique UFMC ID 3, 23, 24, 26, 27, 30, 38, 44, 62, 159, 206, 207, 220, 221, 223, 315, 353–355, 422, 872–877, 889, 924, 1311–1315, 1331, 1347, 1358, 1360, 1363, 1406, 1409, 1506, 1621, 1623, 1625, 1847, 2203, 2204–2213.

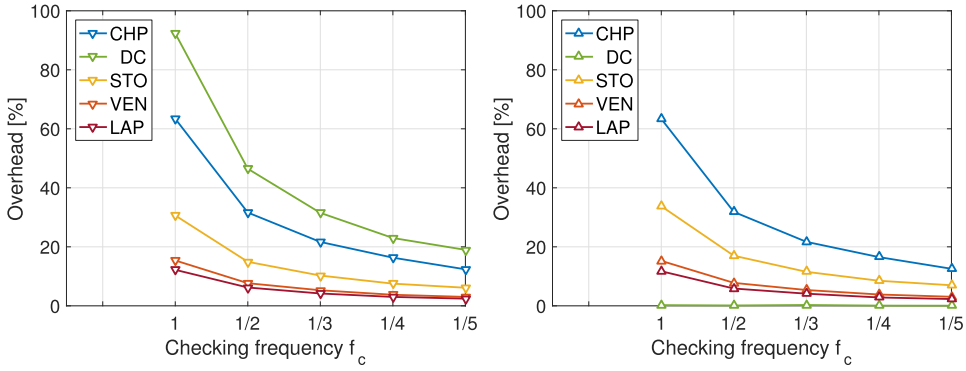

Fig. 7. Overhead of bit-flip detection for the GPU implementation of Jacobi and different checking frequencies f_c. The left plot shows the data for the lower triangular factors, the data for the upper triangular factors is on the right plot. Each curve shows the data for one sparse incomplete factor with the respective properties listed in Table 2.

the protection mechanism linearly decreases with the checking frequency.

8. Undervolting for energy efficiency

Dynamic voltage-frequency scaling (DVFS) is a common technology integrated in most current HPC and embedded architectures to reduce power dissipation of the processor and, for some applications, increase energy efficiency [27]. Compared with DVFS, undervolting is an appealing technique which advocates for reducing the processor voltage only, leaving the frequency, and, therefore, the processor performance, unchanged. However, pushing undervolting to the limits (a case of NTVC) can result in a high number of bit-flips. As a consequence, undervolting is only attractive for applications with some natural tolerance to errors (e.g., video/sound processing) or algorithms that are enhanced to deal with soft errors.

In this section we analyze the trade-off between extended runtime and reduced voltage that can potentially render energy gains for our bit-flip protected version of the Jacobi method. For this purpose, consider a particular voltage-frequency configuration defined by (V_1, f) that is "reliable", and an alternative configuration (V_2, f) where the voltage is below the safe minimum so that the floating-point hardware can introduce bit-flips.

The on-chip energy consumption of the Jacobi method, operating in the reliable configuration, can be modeled as:

$$E_1 = P_1 T_1 = \alpha V_1^2 f T_1, \tag{7}$$

where P_1 and T_1 denote the (average) power dissipation rate and execution time, respectively; and we consider that the energy consumption is proportional (by a factor α) to the product $V_1^2 f$ [28]. We note that this estimation only accounts for the dynamic voltage used in the computations, not the static energy draw constant and independent of the computational load. Also, the energy consumption of the main memory remains outside the scope of this analysis. For computations in unreliable mode/hardware, the dynamic voltage is reduced to V_2. Modeling the energy draw of the fault-tolerant Jacobi remains difficult as the bit-flip protection mechanism has to proceed in reliable mode. In most cases, however, from the perspective of energy, the consumption of the fault-tolerant Jacobi is dominated the sparse matrix-vector product, as the latter will have to retrieve the full matrix from off-chip memory. Therefore, in order to keep the model simple, we neglect the use of reliable computations in the detection mechanism, modeling the dynamic on-chip energy of the Jacobi protected against bit-flips as

$$E_2 = P_2 T_2 = \alpha V_2^2 f T_1 \mu (1 + O_p), \tag{8}$$

where μ is the convergence delay (due to MBF and FPs) while O_p represents the overhead due to the bit-flip protection mechanism. Here, the use of the same parameter α as in (7) reflects that this

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013

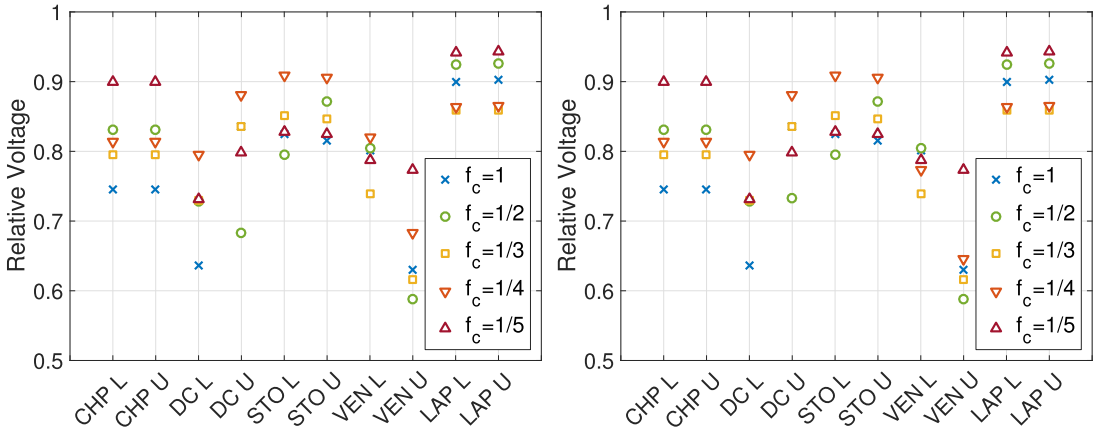

Fig. 8. Required voltage reduction for the Jacobi method operating in unreliable mode to offer the same energy efficiency as the unprotected Jacobi method executed in bit-flip free mode. The left visualizes a scenario where the voltage reduction results in one bit-flip per iteration, $\kappa = 1$ (left) and $\kappa = 5$ (right).

parameter depends on the properties of the operation being executed and the hardware [29].

Given certain overhead factors defined by μ and O_p, the unreliable configuration delivers the same energy cost as the reliable one if the equation

$$E_1 = \alpha V_1^2 f T_1 = \alpha V_2^2 f T_1 \mu (1 + O_p) = E_2, \tag{9}$$

holds. This implies, the unreliable computation has to be operated at the voltage level

$$V_2^2 = \frac{V_1^2}{\mu(1 + O_p)}. \tag{10}$$

At this point, we note that we reported experimental values for μ and O_p in Table 3 and Fig. 7, respectively. Armed with this data, and assuming a normalized voltage $V_1 = 1$, Fig. 8 visualizes the reduction ratio V_2^2/V_1^2 required by the unreliable configuration to match the energy consumption of the reliable one, for scenarios with a single bit-flip per iteration (left), five bit-flips per iteration (right), and moderate checking frequencies $f_c = 1, 1/2, \ldots, 1/5$. This experiment reveals that, for most cases, a reduction of voltage in a factor around 20% or less is sufficient to compensate the convergence delay and protection overhead. In addition, as long as the error bit-flip rate remains within these bounds, for many of these scenarios a larger reduction rate may even result in the unreliable mode rendering higher energy efficiency than the reliable configuration.

9. Concluding remarks

We have described a practical bit-flip protection mechanism that transforms a synchronized Jacobi iteration—with no inherent protection against errors—into an asynchronous fault tolerant relaxation method. The error protection scheme operates at the component level, individually accepting or rejecting updates at each iteration, depending on whether they pass certain tests based on a single user-defined threshold δ. Furthermore, the tests are composed of a few Level-1 BLAS-like operations which can be efficiently implemented in vector (SIMD) units, yielding an affordable overhead.

Our detailed experiments with several sparse benchmarks reveal a number of key insights:

- The threshold δ offers a means to balance the effects of MBFs vs FPs. This evaluation also reveals that, for reasonable bit-flip rates, the relative convergence delay stays within reasonable margins for all residual thresholds.
- As the bit-flip location moves toward the less significant part(s) of the number, errors become easier too miss but exert a milder effect on the relative residual until that level of accuracy is reached.
- For strongly connected systems, a high bit-flip rate can only be tackled via a high checking frequency.
- Unless the bit-flips repeatedly corrupt the same component, the protection mechanism is also able to ensure convergence for high bit-flip rates with a reasonable convergence delay.
- For most problems, the bit-flip protected GPU implementation of `FTJacobi` available in the MAGMA-sparse software package is only slightly slower than the baseline implementation of `Jacobi`.
- The bit-flip protection makes the bit-flip protected Jacobi method attractive for settings where undervolting is used to improve the energy efficiency at the cost of relaxed reliability.

Future work will focus on realizing `FTJacobi` in a SDC-prone hardware setting.

Acknowledgements

This material is based upon work supported in part by the U.S. Department of Energy (Award Number DE-SC-0010042) and NVIDIA. E. S. Quintana-Ortí was supported by project CICYT TIN2014-53495-R of MINECO and FEDER.

References

[1] M. Duranton, K.D. Bosschere, A. Cohen, J. Maebe, H. Munk, HiPEAC Vision 2015, 2015 https://www.hipeac.org/assets/public/publications/vision/hipeac-vision-2015_Dq0boL8.pdf.
[2] P. Kogge, et al., ExaScale Computing Study: Technology Challenges in Achieving ExaScale Systems, 2008 http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf.
[3] R. Lucas, et al., Top Ten Exascale Research Challenges, 2014 http://science.energy.gov//media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.
[4] G. Moore, Cramming more components onto integrated circuits, Electronics 38 (8) (1965) 114–117.
[5] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[6] U. Trottenberg, A. Schuller, Multigrid, Academic Press, Inc, 2001.
[7] E. Chow, A. Patel, Fine-grained parallel incomplete LU factorization, SIAM J. Sci. Comput. 37 (2) (2015) C169–C193.
[8] H. Anzt, E. Chow, J. Dongarra, Iterative sparse triangular solves for preconditioning, in: J.L. Träff, S. Hunold, F. Versaci (Eds.), Euro-Par 2015: Parallel Processing, Vol. 9233 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2015, pp. 650–661, http://dx.doi.org/10.1007/978-3-662-48096-0_50.

[9] H. Anzt, J. Dongarra, E.S. Quintana-Ortí, Tuning stationary iterative solvers for fault resilience, in: Proc. 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA'15, ACM, 2015, pp. 1:1–1:8.
[10] C. Chalios, D.S. Nikolopoulos, S. Catalán, E.S. Quintana-Ortí, Evaluating asymmetric multicore systems-on-chip and the cost of fault tolerance using iso-metrics, in: IET Computers & Digital Techniques Submitted. Available upon request from the authors. A preliminary version is available at arXiv:1503.08104 and was presented at WAPCO-HiPEAC, 2015.
[11] Innovative Computing Lab, Software Distribution of MAGMA version 2.0, 2016 http://icl.cs.utk.edu/magma/.
[12] U. Karpuzcu, N.S. Kim, J. Torrellas, Coping with parametric variation at near-threshold voltages, Micro IEEE 33 (4) (2013) 6–14.
[13] D. Kanter, Intel's Near-threshold Voltage Computing and Applications, September, 2012 http://www.realworldtech.com/near-threshold-voltage/.
[14] G. Bronevetsky, B. de Supinski, Soft error vulnerability of iterative linear algebra methods, in: Proc. 22nd Annual Int. Conf. on Supercomputing, ICS'08, 2008, pp. 155–164.
[15] Z. Chen, Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods, in: Proc. 18th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, PPoPP'13, 2013, pp. 167–176.
[16] P.G. Bridges, K.B. Ferreira, M.A. Heroux, M. Hoemmen, Fault-tolerant Linear Solvers via Selective Reliability, 2012, ArXiv e-prints 1206.1390.
[17] J. Elliott, M. Hoemmen, F. Mueller, Evaluating the impact of SDC on the GMRES iterative solver, in: Proc. 2014 IEEE 28th Int. Parallel and Distributed Processing Symp, IPDPS'14, 2014, pp. 1193–1202.
[18] J. Elliott, M. Hoemmen, F. Mueller, Exploiting data representation for fault tolerance, in: Proc. 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA'14, 2014, pp. 9–16.
[19] P. Sao, R. Vuduc, Self-stabilizing iterative solvers, in: Workshop Latest Advances in Scalable Algorithms for Large-Scale Systems, 2013, pp. 4:1–4:8.
[20] J. Calhoun, M. Snir, L. Olson, M. Garzaran, Understanding the propagation of error due to a silent data corruption in a sparse matrix vector multiply, in: IEEE Int. Conf. Cluster Computing 2015, 2015, pp. 541–542.
[21] D. Chazan, W. Miranker, Chaotic relaxation, Linear Algeb. Appl. 2 (7) (1969) 199–222.
[22] A. Frommer, D.B. Szyld, On asynchronous iterations, J. Comput. Appl. Math. 123 (2000) 201–216.
[23] S. Venkataramani, S.T. Chakradhar, K. Roy, A. Raghunathan, Approximate computing and the quest for computing efficiency, in: Proceedings of the 52nd Annual Design Automation Conference, DAC'15, 2015, pp. 120:1-120:6.
[24] I.S. Duff, G.A. Meurant, The effect of ordering on preconditioned conjugate gradients, BIT 29 (4) (1989) 635–657.
[25] H. Anzt, J. Dongarra, M. Kreutzer, M. Koehler, Efficiency of general Krylov methods on GPUs – an experimental study, in: The Sixth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), 2016.
[26] NVIDIA Corporation, CUDA Toolkit v7. 5, September 2015.
[27] HP Corp., Intel Corp., Microsoft Corp., Phoenix Tech. Ltd., Toshiba Corp., Advanced Configuration and Power Interface Specification, revision 5.0, 2011.
[28] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, 5th ed., Morgan Kaufmann Pub, 2012.
[29] J.I. Aliaga, M. Barreda, M.F. Dolz, A.F. Martín, R. Mayo, E.S. Quintana-Ortí, Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems, Cluster Comput. 17 (4) (2014) 1335–1348.

Hartwig Anzt is a research scientist in Jack Dongarra's Innovative Computing Lab (ICL) at the University of Tennessee. He received his Ph.D. in mathematics from the Karlsruhe Institute of Technology (KIT) in 2012. Dr. Anzt's research interests include simulation algorithms, sparse linear algebra—in particular iterative methods and preconditioning, hardware-optimized numerics for GPU-accelerated platforms, and power-aware computing.

Jack Dongarra holds an appointment at the University of Tennessee, Oak Ridge National Laboratory, and the University of Manchester. He specializes in numerical algorithms in linear algebra, parallel computing, use of advanced-computer architectures, programming methodology, and tools for parallel computers. He was awarded the IEEE Sid Fernbach Award in 2004; in 2008 he was the recipient of the first IEEE Medal of Excellence in Scalable Computing; in 2010 he was the first recipient of the SIAM Special Interest Group on Supercomputing's award for Career Achievement; in 2011 he was the recipient of the IEEE IPDPS Charles Babbage Award; and in 2013 he received the ACM/IEEE Ken Kennedy Award. He is a Fellow of the AAAS, ACM, IEEE, and SIAM and a member of the National Academy of Engineering.

Enrique S. Quintana-Ortí received his bachelor and Ph.D. degrees in Computer Sciences from the Universidad Politecnica de Valencia (Spain) in 1992 and 1996. Currently he is professor in Computer Architecture in the Universidad Jaume I of Castellon (Spain). He has published more than 100 papers in international conferences and journals, and has contributed to software libraries like SLICOT and libflame. His research interests include parallel programming, linear algebra, power consumption, as well as advanced architectures and hardware accelerators.

Please cite this article in press as: H. Anzt, et al., Fine-grained bit-flip protection for relaxation methods, J. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.jocs.2016.11.013