

Extending and Evaluating Fault-Tolerant Preconditioned Conjugate Gradient Methods

Carlos Pachajoa^{*} and Markus Levonyak[†] and Wilfried N. Gansterer[‡]

[‡]Corresponding author

University of Vienna, Faculty of Computer Science, Vienna, Austria

^{*}carlos.pachajoa,[†]markus.levonyak,[‡]wilfried.gansterer@univie.ac.at

Abstract—We compare and refine exact and heuristic fault-tolerance extensions for the *preconditioned conjugate gradient* (PCG) and the *split preconditioner conjugate gradient* (SPCG) methods for recovering from failures of compute nodes of large-scale parallel computers. In the *exact state reconstruction* (ESR) approach, which is based on a method proposed by Chen (2011), the solver keeps extra information from previous search directions of the (S)PCG solver, so that its state can be fully reconstructed if a node fails unexpectedly. ESR does not make use of checkpointing or external storage for saving dynamic solver data and has only negligible computation and communication overhead compared to the failure-free situation. In exact arithmetic, the reconstruction is exact, but in finite-precision computations, the number of iterations until convergence can differ slightly from the failure-free case due to rounding effects. We perform experiments to investigate the behavior of ESR in floating-point arithmetic and compare it to the heuristic *linear interpolation* (LI) approach by Langou et al. (2007) and Agullo et al. (2016), which does not have to keep extra information and thus has lower memory requirements. Our experiments illustrate that ESR, on average, has essentially zero overhead in terms of additional iterations until convergence, whereas the LI approach incurs much larger overheads.

Index Terms—preconditioned conjugate gradient method, split preconditioner conjugate gradient method, extreme-scale parallel computing, node failures, resilience, algorithmic fault tolerance

I. INTRODUCTION

As the computing power of large-scale parallel computers continues to grow, resilience becomes a more prominent problem. The expected time-between-failures of systems with a growing number of interconnected components decreases [1], [2]. New strategies are required to deal with this reliability problem on large-scale and future extreme-scale systems with millions of processors and compute nodes [3].

The *conjugate gradient* (CG) method is a widely used algorithm for the solution of *symmetric and positive-definite* (SPD) linear systems that arise in the context of many science and engineering problems. For improving the efficiency and robustness of this iterative algorithm, *preconditioning* is used for transforming the original linear system to one that has the same solution but can be solved more easily. In this paper, we consider the general *preconditioned conjugate gradient* (PCG) method, which essentially works with any SPD preconditioner. As a special case, we specifically consider the

split preconditioner conjugate gradient (SPCG) [4] method, which is optimized for the important case of preconditioners that are given by their Cholesky factors.

Given is the following situation: solve the sparse linear system $Ax = b$ by applying the PCG or SPCG method, where the matrix A is SPD and the solver runs on a large-scale parallel computer which is susceptible to processor and node failures. The preconditioner for solving this sparse linear system may be given explicitly or implicitly. The current approximation to the solution x (which we call the *iterate*), the residual, the preconditioned residual, and the search direction are distributed among the compute nodes of the parallel computer. Wanted is an algorithm which can continue after an unexpected node failure occurred and that is guaranteed to converge to the correct solution.

In this paper, we review and refine an algorithm originally proposed by Chen [5], which we refer to as the *exact state reconstruction* (ESR) approach and that significantly improves the resilience of the PCG and SPCG methods in the event of a node failure with negligible or even zero overhead during the failure-free execution of those methods. The ESR approach is based on recovering the complete state, i.e., all of the above mentioned vectors, of the last iteration prior to the node failure, thus allowing the solver to converge following the same steps as the undisturbed solver. We compare the ESR approach with a heuristic approach proposed by Langou et al. [6] and Agullo et al. [7] which is based on restarting the solver with an approximation of the lost iterate and, therefore, potentially requires many additional iterations after a node failure.

A. Problem setting, terminology and assumptions

We consider a situation where the (S)PCG solver is executed in parallel on N compute nodes of a distributed-memory parallel computer. Different compute nodes communicate over a—usually high-speed—interconnection network. It is furthermore assumed that each compute node consists of m processors, i.e., the solver is executed on a total of $M := N \times m$ processors. Each processor shares its memory with all other processors on the same compute node.

We distinguish failures of processors and compute nodes. In the event of a *single processor failure*, exactly one processor on one compute node fails, but the shared memory on this node stays intact. Hence, no data is lost and the remaining

processors on the node can easily take over the workload of the failed processor. The same holds for *multiple processor failures* where at least one processor per compute node survives. A more severe event, which we focus on in this paper, is a *node failure*, where a compute node fails as a whole and the data in the memory of the affected node is lost.

There are different reasons for a node failure, for example, all m processors of a node fail, the shared memory of a node gets corrupted, or a node loses its connection to the interconnection network. In case of a *single node failure*, exactly one compute node fails at a time. If more than one node fails at a time, we are talking about a *multiple node failure*. Nodes that continue working after a node failure and keep all their data are *surviving nodes*. A node that becomes unavailable after a node failure is referred to as a *failed node*, and a node that takes the place of a failed node in the recovery process is a *replacement node*.

1) *Failure detection and node replacement*: For executing the (S)PCG solver on a parallel computer, we assume the availability of a parallel runtime environment that provides functionality comparable to state-of-the-art implementations of the industry-standard *Message Passing Interface* (MPI) [8]. This particularly comprises efficient collective communication capabilities like the *Gather*, *Allgather*, and *Allreduce* functions of MPI. Like in [7], beyond that we assume that the underlying runtime environment supports some basic fault-tolerance features. A prototypical example is the MPI extension *User Level Failure Mitigation* (ULFM) [9], [10] which supports the detection of node failures, the prevention of indefinitely blocking synchronizations or communications, the notification of the surviving nodes that a failure has occurred (and also which nodes have failed), and a mechanism for providing replacement nodes.

2) *Data distribution*: For the sparse linear system $\mathbf{Ax} = \mathbf{b}$ to be solved by the (S)PCG method, we assume, similar to [5]–[7], that the fundamental static data which defines the problem, i.e., the system matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, the right-hand-side vector $\mathbf{b} \in \mathbb{R}^n$, and the preconditioner $\mathbf{M} \in \mathbb{R}^{n \times n}$ (cf. Sec. II-A) can be retrieved from reliable external storage (e.g., a checkpoint prior to entering the linear solver, cf. the *ABFT&PeriodicCkpt* algorithm [3], [11], [12], see Sec. I-B). Therefore, static data does not have to be reconstructed after a node failure occurred.

In accordance with typical distributions of sparse matrices of widely used high-performance numerical libraries like PETSc [13], we assume a block-row data distribution of all matrices and vectors among the N nodes of the parallel computer. This means that every node owns contiguous blocks of n/N rows (if $n = cN$, $c \in \mathbb{N}$, else some nodes own $\lfloor n/N \rfloor$ and others $\lfloor n/N \rfloor + 1$ rows) of both the matrices $\mathbf{A}$ and $\mathbf{M}$ as well as each of the vectors $\mathbf{b}$, $\mathbf{x}$, and all others maintained by the PCG or SPCG solver (cf. Sec. II-A). On one compute node, the owned block rows, which are stored in the shared memory of the node, are evenly distributed among the m processors, i.e., each processor owns (approximately) n/M rows of each of the matrices and vectors. Globally used scalars are replicated on all N nodes.

Since each compute node owns block rows of all matrices and vectors, a part of each matrix and vector is affected by a node failure. Block rows of non-static (i.e., *dynamic*) data which were owned by the failed node are lost and need to be reconstructed on the replacement node (cf. Sec. II-B).

3) *Notation*: We use a notation similar to [7] to denote sections of matrices and vectors. We refer to the set of all indices as $I := \{1, 2, \dots, n\}$. The cardinality n of I is equal to the size of the vectors. The index subset representing the rows assigned to node i is denoted as I_i . Given a vector $\mathbf{v}^{(j)}$, where j denotes the iteration number of the linear solver, $\mathbf{v}_{I_i}^{(j)}$ refers to the subset of elements of the vector at iteration j corresponding to node i . Row and column selections of a matrix $\mathbf{B}$ are designated with index sets as well: $\mathbf{B}_{I_i, I_k}$ refers to the selection of rows and columns of $\mathbf{B}$ corresponding to the index subsets I_i and I_k , respectively. The failed node is referred to as node f , and its index set is hence denoted as I_f . The $\mathbf{B}$ -norm of a vector $\mathbf{v}$, written as $\|\mathbf{v}\|_{\mathbf{B}}$, is defined as $\|\mathbf{v}\|_{\mathbf{B}}^2 = \mathbf{v}^\top \mathbf{B} \mathbf{v}$.

The *state* of an iterative solver is the—not necessarily minimal—set of data that completely defines its future behavior. In case of the PCG solver, the state in iteration j can be defined as being composed of the iterate $\mathbf{x}^{(j)}$, the residual $\mathbf{r}^{(j)}$, the preconditioned residual $\mathbf{z}^{(j)}$, and the search direction $\mathbf{p}^{(j)}$. For the SPCG variant, the vector $\hat{\mathbf{r}}^{(j)}$ replaces $\mathbf{r}^{(j)}$ and $\mathbf{z}^{(j)}$ (cf. Sec. II-A). The *trajectory* of the solver is the sequence of states it goes through until it converges.

B. Related work

Existing work on resilience for iterative linear solvers distinguishes between soft errors and node failures. The former refers to spontaneous changes of the state of the solver, potentially leading to a wrong result. In this category we find work by Sao and Vuduc [14], where they propose strategies to ensure that the CG method will converge to the right solution after a soft error. Their approach requires the ability to perform some operations reliably, thus bringing the solver back to a trajectory that will converge to the solution.

Bronevetsky and de Supinski [15] evaluate the effects of soft errors on iterative linear solvers including the CG method. Error bits are introduced at random times as well as random positions and the effects are classified according to the resulting runtimes and errors in the solutions. Dichev and Nikolopoulos [16] propose and experimentally evaluate a specific form of *Dual Modular Redundancy*, where all computations are performed twice for improved redundancy, in order to detect and correct soft errors in the PCG method. More work in the area of soft error detection and correction in the CG and PCG methods has been published by Shantharam et al. [17] and Fasi et al. [18]. These approaches have in common that there is no immediate application to our problem of the (S)PCG solver subject to node failures.

Herauld et al. [3] provide a comprehensive overview of different *checkpoint/restart* approaches, which currently is the most commonly used class of fault-tolerance techniques to cope with node failures. These checkpoint/restart approaches

frequently save the current state of a running application and roll back to the last saved state in case of a node failure. To avoid the overhead of continuously saving the state, Chen [5] exploits specific properties of the PCG solver and other iterative methods such that the state of the solver can be recovered without checkpointing. We review this method in Sec. II and refine it in Sec. III.

Langou et al. [6] propose a heuristic *interpolation/restart* strategy, applicable to iterative linear solvers in general, to recover from a node failure by approximating the lost iterate. They use the rows of the matrix corresponding to the indices of the failed node to produce an interpolated approximation of the iterate before the node failure occurred. Agullo et al. [7], [19] extend this approach. They perform interpolation using *all* the information of the matrix, which produces a reconstructed iterate whose error norm is guaranteed to be smaller than the error norm of the iterate before the node failure, albeit with significant communication overhead (cf. Sec. II-B).

Bosilca et al. [3], [11], [12] introduce an algorithm called *ABFT&PeriodicCkpt*, which combines *algorithm-based fault tolerance* (ABFT)—a term originally used for specific resilient matrix operations but today often used in a broader sense including resilient iterative linear solvers—with periodic checkpointing in order to make whole applications (and not just operations that can be protected by ABFT) resilient to node failures. The longer the phases protected by ABFT, the less checkpoints are necessary and the cheaper resilience against node failures usually becomes (assuming that the used ABFT method is more efficient than checkpointing).

C. Contributions of this work

To the best of our knowledge, there is no prior work that thoroughly compares the numerical effects on the convergence of PCG and the performance of the heuristic approach by Langou et al. and Agullo et al. [6], [7], [19] and the exact method by Chen [5]. Moreover, we refine the exact method—which we call the ESR approach—by distinguishing different common types of preconditioners and by considering multiple node failures at a time. In particular, we distinguish whether a general preconditioner M , its inverse $P = M^{-1}$, or a split preconditioner $M = LL^\top$ is given. For the split preconditioner, we consider the specialized SPCG solver and present a modified ESR approach for it.

The remainder of this paper is structured as follows. In Sec. II, we briefly review the PCG and SPCG methods as well as the heuristic resilience techniques by Langou et al. and Agullo et al. [6], [7], [19]. Subsequently, we review and refine the ESR approach as presented by Chen [5] for making the (S)PCG method resilient to node failures. In Sec. III, we present refinements for the ESR method to work with different types of preconditioners and during multiple node failures overlapping in time. In Sec. IV, we present numerical experiments that compare exact and heuristic approaches. We use large real-world matrices from the *SuiteSparse Matrix Collection* [20] for investigating the effects of floating-point arithmetic. Finally, we summarize our results in Sec. V.

```

1:  $\mathbf{r}^{(0)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}, \mathbf{z}^{(0)} := \mathbf{M}^{-1}\mathbf{r}^{(0)}, \mathbf{p}^{(0)} := \mathbf{z}^{(0)}$ 
2: for  $j = 0, 1, \dots$ , until convergence do
3:    $\alpha^{(j)} := \mathbf{r}^{(j)\top} \mathbf{z}^{(j)} / \mathbf{p}^{(j)\top} \mathbf{A} \mathbf{p}^{(j)}$ 
4:    $\mathbf{x}^{(j+1)} := \mathbf{x}^{(j)} + \alpha^{(j)} \mathbf{p}^{(j)}$ 
5:    $\mathbf{r}^{(j+1)} := \mathbf{r}^{(j)} - \alpha^{(j)} \mathbf{A} \mathbf{p}^{(j)}$ 
6:    $\mathbf{z}^{(j+1)} := \mathbf{M}^{-1} \mathbf{r}^{(j+1)}$ 
7:    $\beta^{(j)} := \mathbf{r}^{(j+1)\top} \mathbf{z}^{(j+1)} / \mathbf{r}^{(j)\top} \mathbf{z}^{(j)}$ 
8:    $\mathbf{p}^{(j+1)} := \mathbf{z}^{(j+1)} + \beta^{(j)} \mathbf{p}^{(j)}$ 
9: end for

```

Algorithm 1: Preconditioned conjugate gradient (PCG) method [4, Alg. 9.1]

```

1:  $\mathbf{r}^{(0)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}, \hat{\mathbf{r}}^{(0)} := \mathbf{L}^{-1} \mathbf{r}^{(0)}, \mathbf{p}^{(0)} := \mathbf{L}^{-\top} \hat{\mathbf{r}}^{(0)}$ 
2: for  $j = 0, 1, \dots$ , until convergence do
3:    $\alpha^{(j)} := \hat{\mathbf{r}}^{(j)\top} \hat{\mathbf{r}}^{(j)} / \mathbf{p}^{(j)\top} \mathbf{A} \mathbf{p}^{(j)}$ 
4:    $\mathbf{x}^{(j+1)} := \mathbf{x}^{(j)} + \alpha^{(j)} \mathbf{p}^{(j)}$ 
5:    $\hat{\mathbf{r}}^{(j+1)} := \hat{\mathbf{r}}^{(j)} - \alpha^{(j)} \mathbf{L}^{-1} \mathbf{A} \mathbf{p}^{(j)}$ 
6:    $\beta^{(j)} := \hat{\mathbf{r}}^{(j+1)\top} \hat{\mathbf{r}}^{(j+1)} / \hat{\mathbf{r}}^{(j)\top} \hat{\mathbf{r}}^{(j)}$ 
7:    $\mathbf{p}^{(j+1)} := \mathbf{L}^{-\top} \hat{\mathbf{r}}^{(j+1)} + \beta^{(j)} \mathbf{p}^{(j)}$ 
8: end for

```

Algorithm 2: Split preconditioner conjugate gradient (SPCG) method [4, Alg. 9.2]

II. METHODOLOGICAL BACKGROUND

In this section, we review the PCG and SPCG methods, along with a more detailed look at the major heuristic methods for coping with node failures that have—in contrast to checkpoint/restart procedures—no overhead during the failure-free execution of the solver, and the ESR approach.

A. Preconditioned conjugate gradient method

The CG method as well as its preconditioned variants (PCG and SPCG) iteratively solve a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, where the SPD matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the right-hand-side vector $\mathbf{b} \in \mathbb{R}^n$ are given. The search directions $\mathbf{p}^{(j)}$, along which the quadratic potential defined by $\mathbf{A}$ is minimized, are chosen to be $\mathbf{A}$ -orthogonal, i.e., $\mathbf{p}^{(j)\top} \mathbf{A} \mathbf{p}^{(k)} = 0$ for all $j \neq k$. The residual $\mathbf{r}^{(j)}$ is defined as $\mathbf{r}^{(j)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$.

In the case of PCG and SPCG, a preconditioner $\mathbf{M} \in \mathbb{R}^{n \times n}$ is used for accelerating convergence. Instead of the original system, the linear system $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ is then solved. $\mathbf{M}$ is assumed to be an SPD matrix as well [4, p. 276] and is to be chosen such that $\kappa(\mathbf{M}^{-1}\mathbf{A}) < \kappa(\mathbf{A})$, where $\kappa(\mathbf{B})$ denotes the condition number of a matrix $\mathbf{B}$. For the special case of a *split preconditioner* in the context of the SPCG method, we have $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is lower triangular (the Cholesky factor of $\mathbf{M}$). The standard (non-resilient) PCG and SPCG methods are listed in Alg. 1 and 2, respectively.

The PCG method stores the distributed vectors $\mathbf{x}^{(j)}$, $\mathbf{r}^{(j)}$, $\mathbf{z}^{(j)}$, $\mathbf{p}^{(j)}$, and $\mathbf{A}\mathbf{p}^{(j)}$, plus the scalars $\alpha^{(j)}$ and $\beta^{(j)}$ replicated on every node. The method requires space for about $5n + 2N$ floating-point numbers to store its dynamic data, that is, solver data excluding the matrix $\mathbf{A}$ and the right-hand-side vector $\mathbf{b}$. Likewise, the SPCG method stores the distributed vectors

$\mathbf{x}^{(j)}$, $\hat{\mathbf{r}}^{(j)}$, $\mathbf{p}^{(j)}$, and $\mathbf{A}\mathbf{p}^{(j)}$, plus $\alpha^{(j)}$ and $\beta^{(j)}$ again, and needs space for only about $4n + 2N$ floating-point numbers. As already defined in Sec. I-A, we consider the state of the PCG or SPCG solver as being composed of the just mentioned distributed vectors except for $\mathbf{A}\mathbf{p}^{(j)}$. Notice that if we have all the vectors from a state in a trajectory that converges to the solution, the method will follow this same trajectory.

B. Resilience to node failures

There are basically three classes of approaches for improving the resilience of iterative linear solvers against node failures (cf. Sec. I-B). Firstly, there are various types of checkpoint/restart methods, which have in common that they—even in the failure-free case—impose a usually considerable runtime overhead since they continuously need to save the state of the solver [3]. Secondly, there are heuristic interpolation/restart methods, which try to reconstruct the most current approximation to the solution vector after a node failure without requiring any overhead during the failure-free execution of the linear solver. Thirdly, there are methods which can exactly reconstruct the complete state of an iterative linear solver without checkpointing, as we show later in this section. In this paper, we are interested in the latter two approaches due to their lower overheads.

The state-of-the-art interpolation/restart methods were proposed by Langou et al. [6] and Agullo et al. [7], which consider general Krylov subspace methods in a problem setting similar to ours (cf. Sec. I-A) including the requirement that failure detections and node replacements are handled at the system layer (cf. Sec. I-A1). Agullo et al. present two algorithms for the reconstruction of a lost part of the iterate after a node failure occurred. After the reconstruction by one of those interpolation methods, the linear solver has to be restarted with the reconstructed iterate as the new initial guess for the solver. The first method, which was initially introduced by Langou et al., needs to *locally*, i.e., on one node, solve a—much smaller than the original—linear system during the reconstruction process and is thus called the *linear interpolation* (LI) method:

$$\begin{aligned} \mathbf{x}_{I_i}^{(\text{LI})} &:= \mathbf{x}_{I_i}^{(j)} \quad \forall i \neq f, \\ \mathbf{x}_{I_f}^{(\text{LI})} &:= \mathbf{A}_{I_f, I_f}^{-1} \left(\mathbf{b}_{I_f} - \sum_{i \neq f} \mathbf{A}_{I_f, I_i} \mathbf{x}_{I_i}^{(j)} \right), \end{aligned} \quad (1)$$

where the superscript (LI) refers to the vectors that have been reconstructed by the LI method after a node failure. The failed and afterwards replaced node is denoted by f , while i designates any given node.

The LI approach requires the diagonal block $\mathbf{A}_{I_f, I_f}$ of $\mathbf{A}$ to be non-singular, which is fulfilled if $\mathbf{A}$ is an SPD matrix and, thus, always for the CG, PCG, and SPCG solvers. It can be proven [7] that the $\mathbf{A}$ -norm of the forward error $\mathbf{e}^{(\text{LI})} := \mathbf{x} - \mathbf{x}^{(\text{LI})}$ after the reconstruction is smaller than the forward error $\mathbf{e}^{(j)} := \mathbf{x} - \mathbf{x}^{(j)}$ before the node failure occurred, i.e., $\|\mathbf{e}^{(\text{LI})}\|_{\mathbf{A}} \leq \|\mathbf{e}^{(j)}\|_{\mathbf{A}}$. In the (mostly hypothetical) case that the surviving nodes already had their parts of the solution $\mathbf{x}$ of the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, the LI method would reconstruct

the solution of the system on the replacement node f [7]. In general, the LI method will reconstruct an iterate that is different from the iterate prior to the node failure.

The second interpolation/restart method by Agullo et al. [7] needs to *globally*, i.e., on all nodes, solve a least squares problem during the reconstruction phase and is therefore referred to as the *least squares interpolation* (LSI) method:

$$\begin{aligned} \mathbf{x}_{I_i}^{(\text{LSI})} &:= \mathbf{x}_{I_i}^{(j)} \quad \forall i \neq f, \\ \mathbf{x}_{I_f}^{(\text{LSI})} &:= \arg \min_{\mathbf{x}_{I_f}^{(j)}} \left\| \left(\mathbf{b} - \sum_{i \neq f} \mathbf{A}_{I, I_i} \mathbf{x}_{I_i}^{(j)} \right) - \mathbf{A}_{I, I_f} \mathbf{x}_{I_f}^{(j)} \right\|, \end{aligned}$$

where the superscript (LSI) refers to vectors reconstructed using the LSI method. This approach is more general than the LI method since the submatrix $\mathbf{A}_{I_f, I_f}$ is not required to be non-singular and only $\mathbf{A}$ needs to have full rank. This significantly increases the communication cost due to the involved global least squares problem but is beneficial for iterative linear solvers that operate on matrices $\mathbf{A}$ which are not SPD. However, this higher generality is not relevant for the (S)PCG solver considered in this paper. Thus, we focus on the LI method from now on.

For the LSI method, it holds that $\|\mathbf{r}^{(\text{LSI})}\|_2 \leq \|\mathbf{r}^{(j)}\|_2$ [7]. As with the LI method, the correct solution of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ would be generated if the surviving nodes already contained their parts of the solution. In both cases, the reconstructed iterate guarantees that the error of the process is monotonically decreasing when measured with the $\mathbf{A}$ -norm. However, since they are interpolation/restart methods which are agnostic to the iterative linear solver being used, both the LI and LSI methods can only approximate the iterate but are not able to reconstruct the complete state of the solver.

Finally, the ESR approach by Chen [5] is able to reconstruct all vectors required for continuing without any loss of information at the iteration the node failure happened. During the failure-free PCG iterations, only negligible or—depending on the linear system to be solved—no additional communication is necessary for achieving this. The only overhead required for the recovery of the state of the solver is slightly higher local memory requirements than the non-resilient standard variants.

At iteration j of the PCG algorithm, the matrix-vector product $\mathbf{A}\mathbf{p}^{(j)}$ has to be computed (cf. Alg. 1, lines 3 and 5, and Alg. 2, lines 3 and 5). To illustrate the fundamental concepts, let us first consider the hypothetical case that $\mathbf{A}\mathbf{p}^{(j)}$ is computed by a dense matrix-vector multiplication kernel. Since we assume a row-wise distribution of $\mathbf{A}$, the whole vector $\mathbf{p}^{(j)}$ needs to be communicated to all nodes (e.g., by an *Allgather* operation) in this scenario. In the standard non-resilient PCG and SPCG solvers, all but the own block $\mathbf{p}_{I_i}^{(j)}$ can be dropped on node i after the product has been computed.

In the resilient algorithm, we can also drop most of $\mathbf{p}^{(j)}$ on each node after the matrix-vector multiplication. The crucial difference to standard non-resilient (S)PCG is that now each node also stores the block of one of its neighbors in addition to its own block. In particular, this means that there is a redundant

- 1: Retrieve the static data $\mathbf{A}_{I_f, I}$, $\mathbf{M}_{I_f, I}$, and $\mathbf{b}_{I_f}$
- 2: Gather $\mathbf{x}_{I \setminus I_f}^{(j)}$ and $\mathbf{z}_{I \setminus I_f}^{(j)}$
- 3: Retrieve the redundant copies of $\beta^{(j-1)}$, $\mathbf{p}_{I_f}^{(j-1)}$, and $\mathbf{p}_{I_f}^{(j)}$
- 4: Compute $\mathbf{z}_{I_f}^{(j)} := \mathbf{p}_{I_f}^{(j)} - \beta^{(j-1)} \mathbf{p}_{I_f}^{(j-1)}$
- 5: Compute $\mathbf{r}_{I_f}^{(j)} := \mathbf{M}_{I_f, I} \mathbf{z}_{I_f}^{(j)}$
- 6: Compute $\mathbf{w} := \mathbf{b}_{I_f} - \mathbf{r}_{I_f}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)}$
- 7: Solve $\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{w}$ for $\mathbf{x}_{I_f}^{(j)}$

Algorithm 3: Exact reconstruction of the state on the replacement node f for the PCG method (cf. Alg. 1) if the preconditioner $\mathbf{M}$ (not $\mathbf{M}^{-1}$) is given

copy of every block of $\mathbf{p}^{(j)}$ on another node aside from its owner in a round-robin fashion, i.e., the redundant copy of $\mathbf{p}_{I_i}^{(j)}$ is always kept on node $i - 1$, or on node $N - 1$ if $i = 0$. Hence, in case of a node failure, this redundant copy can be sent to the replacement node in order to completely recover the most recent search direction. We need the two most recent search directions [5, SS5.2], we can keep redundant copies of each block of both $\mathbf{p}^{(j-1)}$ and $\mathbf{p}^{(j)}$ in a similar manner and, thus, are able to recover the two most recent search directions after a node failure. Therefore, the local memory overhead is $2n/N$ vector elements per node and $2n$ vector elements in total, which is negligible compared to the overall memory requirement $\mathcal{O}(n^2)$ of the PCG solvers (cf. Sec. II-A).

In the more realistic case of sparse $\mathbf{A}$, it is more efficient to compute $\mathbf{A}\mathbf{p}^{(j)}$ with a *sparse matrix-vector multiplication* (SpMV) kernel. Chen explains in [5] that all the necessary redundant data will be present on another node under the condition that all columns of matrices $\mathbf{A}_{I \setminus I_j, I_j}$, $j \in \{1..N\}$ are non-zero. If this condition is not fulfilled, as it is frequently the case, it is necessary to send all the entries of a node, and not only the ones required for the SpMV operation, to some other node. As a consequence, complete redundancy of the vector incurs additional communication and memory during the normal operation of the CG algorithm, and it is desirable to minimize the number of additional entries. For our simulations, we use the strategy proposed by Chen [5], where node i sends the totality of its contents to node $(i+1) \bmod N$. The procedure to reconstruct the state of the PCG solver is outlined in Alg. 3 and Alg. 4. These algorithms represent the method in [5]. Their separation is discussed in section III-A.

An important observation here is that the scalar $\beta^{(j-1)}$ is replicated on every node (cf. Sec. I-A and Sec. II-A), i.e., $\beta^{(j-1)}$ has the same value on all nodes. Furthermore, the reconstruction of $\mathbf{z}$ takes place in the replacement node. The matrix of the linear system of line 7 are positive definite and much smaller than the full system matrix, and can be solved locally in the replacement node.

Lines 6 and 7 of Alg. 3, and lines 7 and 8 of Alg. 4, are solving the following equation:

$$\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{b}_{I_f} - \mathbf{r}_{I_f}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)}, \quad (2)$$

- 1: Retrieve the static data $\mathbf{A}_{I_f, I}$, $\mathbf{P}_{I_f, I}$, and $\mathbf{b}_{I_f}$
- 2: Gather $\mathbf{r}_{I \setminus I_f}^{(j)}$ and $\mathbf{x}_{I \setminus I_f}^{(j)}$
- 3: Retrieve the redundant copies of $\beta^{(j-1)}$, $\mathbf{p}_{I_f}^{(j-1)}$, and $\mathbf{p}_{I_f}^{(j)}$
- 4: Compute $\mathbf{z}_{I_f}^{(j)} := \mathbf{p}_{I_f}^{(j)} - \beta^{(j-1)} \mathbf{p}_{I_f}^{(j-1)}$
- 5: Compute $\mathbf{v} := \mathbf{z}_{I_f}^{(j)} - \mathbf{P}_{I_f, I \setminus I_f} \mathbf{r}_{I \setminus I_f}^{(j)}$
- 6: Solve $\mathbf{P}_{I_f, I_f} \mathbf{r}_{I_f}^{(j)} = \mathbf{v}$ for $\mathbf{r}_{I_f}^{(j)}$
- 7: Compute $\mathbf{w} := \mathbf{b}_{I_f} - \mathbf{r}_{I_f}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)}$
- 8: Solve $\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{w}$ for $\mathbf{x}_{I_f}^{(j)}$

Algorithm 4: Exact reconstruction of the state on the replacement node f for the PCG method (cf. Alg. 1) if the preconditioner $\mathbf{P} := \mathbf{M}^{-1}$ is given

where $\mathbf{A}_{I_f, I_f}$ has full rank. Thus, by comparing Eq. (2) to Eq. (1), one can see that the ESR approach can be interpreted as a refinement of the LI method proposed in [6] and [7] (cf. Sec. II-B). However, due to the subtraction of the residual $\mathbf{r}_{I_f}^{(j)}$, the ESR method not only *approximates* but *exactly reconstructs* the lost part $\mathbf{x}_{I_f}^{(j)}$ of the iterate. For solving the linear system in Eq. (2) locally on the replacement node f , the only communication involved is the gathering of $\mathbf{x}_{I \setminus I_f}^{(j)}$. The lost part $\mathbf{r}_{I_f}^{(j)}$ of the residual $\mathbf{r}^{(j)}$ has to be restored before the linear system can be solved.

III. REFINEMENTS OF THE ESR APPROACH

In this section, we present our refinements to the ESR approach of [5].

A. Algorithm selection depending on how the preconditioner is provided

Previous work does not explain the details of how to operate depending on how preconditioners are provided. We present algorithms for the case where the preconditioner is given as $\mathbf{M}$ in Alg. 3, or as its inverse $\mathbf{P} = \mathbf{M}^{-1}$ in Alg. 4, as it is necessary to gather either $\mathbf{z}$ or $\mathbf{r}$ to the replacement node, respectively, to be able to use the relation $\mathbf{r} = \mathbf{M}\mathbf{z}$.

B. Reconstruction of the state of the SPCG solver

First, we derive the ESR method for the SPCG solver (cf. Sec. II-A and Alg. 2). Specifically, we explain how the lost parts of the residual $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^{-1} \mathbf{r}^{(j)}$ and the iterate $\mathbf{x}^{(j)}$ can be recomputed after the two most recent search directions $\mathbf{p}^{(j-1)}$ and $\mathbf{p}^{(j)}$ have been recovered. To ensure an exact reconstruction, we derive all required equations directly from the linear solver relations, in this case, from the SPCG relations in Alg. 2. First, we receive an equation for reconstructing the lost part $\hat{\mathbf{r}}_{I_f}^{(j)}$ of the residual $\hat{\mathbf{r}}^{(j)}$.

Lemma 1. Let $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$, $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^{-1} \mathbf{r}^{(j)}$, $\mathbf{p}^{(j-1)}$, $\mathbf{p}^{(j)}$, and $\beta^{(j-1)}$ be given as in Alg. 2. Then, it holds that

$$\hat{\mathbf{r}}_{I_f}^{(j)} = \mathbf{L}_{I_f, \cup_{i=f}^N I_i}^\top \left(\mathbf{p}_{\cup_{i=f}^N I_i}^{(j)} - \beta^{(j-1)} \mathbf{p}_{\cup_{i=f}^N I_i}^{(j-1)} \right).$$

Proof. From line 7 of Alg. 2, it immediately follows that $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^\top (\mathbf{p}^{(j)} - \beta^{(j-1)} \mathbf{p}^{(j-1)})$. Hence, it furthermore holds that $\hat{\mathbf{r}}_{I_f}^{(j)} = \mathbf{L}_{I_f, I}^\top (\mathbf{p}^{(j)} - \beta^{(j-1)} \mathbf{p}^{(j-1)})$. Since $\mathbf{L}^\top$ is an upper triangular matrix and, thus, $\mathbf{L}_{I_f, \cup_{i=1}^{f-1} I_i}^\top = \mathbf{0}$, we obtain the statement of the lemma. $\square$

We note here again that $\beta^{(j-1)}$ has the same value on all nodes and, hence, the node $i \neq f$ that transmits the redundant copies of $\mathbf{p}_{I_f}^{(j-1)}$ and $\mathbf{p}_{I_f}^{(j)}$ can also send $\beta^{(j-1)}$ to the replacement node f in the reconstruction phase after a node failure. A significant difference between reconstructing $\mathbf{z}_{I_f}^{(j)}$ for the PCG method according to Alg. 4 and recomputing $\hat{\mathbf{r}}_{I_f}^{(j)}$ for the SPCG method based on Lem. 1 is that for reconstructing the residual $\hat{\mathbf{r}}_{I_f}^{(j)}$ not only the local parts $\mathbf{p}_{I_f}^{(j-1)}$ and $\mathbf{p}_{I_f}^{(j)}$ of the two most recent search direction vectors but instead the larger parts $\mathbf{p}_{\cup_{i=f}^N I_i}^{(j)}$ and $\mathbf{p}_{\cup_{i=f}^N I_i}^{(j-1)}$ are needed, which have to be gathered first from other nodes before $\hat{\mathbf{r}}_{I_f}^{(j)}$ can be computed locally on the replacement node f . On the other hand, $\hat{\mathbf{r}}^{(j)}$ is the only residual vector that needs to be reconstructed for the SPCG solver, while for the PCG solver both $\mathbf{z}^{(j)}$ and $\mathbf{r}^{(j)}$ have to be reconstructed, where recomputing the lost part $\mathbf{r}_{I_f}^{(j)}$ of the residual $\mathbf{r}^{(j)}$, in agreement with Alg. 4, also requires the gathering of a vector.

The last missing vector after the search direction $\mathbf{p}^{(j)}$ and the residual $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^{-1} \mathbf{r}^{(j)}$ for a complete reconstruction of the state of iteration j of the SPCG solver is the iterate $\mathbf{x}^{(j)}$. Analogous to the reconstruction of the iterate $\mathbf{x}^{(j)}$ for the PCG solver (cf. Alg. 4), we have to solve a—much smaller than the original—linear system for recomputing the lost part $\mathbf{x}_{I_f}^{(j)}$ of the iterate $\mathbf{x}^{(j)}$ for the SPCG solver after a node failure occurred. We again use the residual relation $\mathbf{r}^{(j)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)}$ (cf. Sec. II-A) for deriving this linear system.

Lemma 2. *Let $\mathbf{y}_{I_f}^{(j)}$ be the lost part of a vector $\mathbf{y}^{(j)} \in \mathbb{R}^n$ after a node failure, where $\mathbf{B}\mathbf{y}^{(j)} = \mathbf{c}^{(j)}$ with an SPD matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ and $\mathbf{c}^{(j)} \in \mathbb{R}^n$. Then, $\mathbf{y}_{I_f}^{(j)}$ can be reconstructed by solving the linear system*

$$\mathbf{B}_{I_f, I_f} \mathbf{y}_{I_f}^{(j)} = \mathbf{c}_{I_f}^{(j)} - \mathbf{B}_{I_f, I \setminus I_f} \mathbf{y}_{I \setminus I_f}^{(j)},$$

where $\mathbf{B}_{I_f, I_f}$ has full rank.

Proof. Due to $\mathbf{B}\mathbf{y}^{(j)} = \mathbf{c}^{(j)}$, it holds that $\mathbf{B}_{I_f, I} \mathbf{y}^{(j)} = \mathbf{c}_{I_f}^{(j)}$. By reordering the columns of $\mathbf{B}_{I_f, I}$ and the rows of $\mathbf{y}^{(j)}$, we furthermore receive

$$\begin{aligned} & (\mathbf{B}_{I_f, I_f} \quad \mathbf{B}_{I_f, I \setminus I_f}) \begin{pmatrix} \mathbf{y}_{I_f}^{(j)} \\ \mathbf{y}_{I \setminus I_f}^{(j)} \end{pmatrix} = \mathbf{c}_{I_f}^{(j)} \\ \iff & \mathbf{B}_{I_f, I_f} \mathbf{y}_{I_f}^{(j)} = \mathbf{c}_{I_f}^{(j)} - \mathbf{B}_{I_f, I \setminus I_f} \mathbf{y}_{I \setminus I_f}^{(j)}. \end{aligned}$$

Since $\mathbf{B}$ is SPD, the square diagonal block $\mathbf{B}_{I_f, I_f}$ is non-singular and, thus, the linear system has a unique solution. $\square$

Corollary 1. *Let $\mathbf{A}$, $\mathbf{b}$, $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$, $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^{-1} \mathbf{r}^{(j)}$, and $\mathbf{x}^{(j)}$ be given as in Alg. 2. Then, it holds that*

$$\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{b}_{I_f} - \mathbf{L}_{I_f, \cup_{i=1}^f I_i} \hat{\mathbf{r}}_{\cup_{i=1}^f I_i}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)},$$

- 1: Retrieve the static data $\mathbf{A}_{I_f, I}$, $\mathbf{L}_{I_f, I}$, $\mathbf{L}_{I_f, I}^\top$, and $\mathbf{b}_{I_f}$
- 2: Gather $\mathbf{p}_{\cup_{i=f+1}^N I_i}^{(j-1)}$, $\mathbf{p}_{\cup_{i=f+1}^N I_i}^{(j)}$, $\hat{\mathbf{r}}_{\cup_{i=1}^f I_i}^{(j)}$, and $\mathbf{x}_{I \setminus I_f}^{(j)}$
- 3: Retrieve the redundant copies of $\beta^{(j-1)}$, $\mathbf{p}_{I_f}^{(j-1)}$, and $\mathbf{p}_{I_f}^{(j)}$
- 4: Compute $\hat{\mathbf{r}}_{I_f}^{(j)} := \mathbf{L}_{I_f, \cup_{i=f}^N I_i}^\top \left(\mathbf{p}_{\cup_{i=f}^N I_i}^{(j)} - \beta^{(j-1)} \mathbf{p}_{\cup_{i=f}^N I_i}^{(j-1)} \right)$
- 5: Compute $\mathbf{w} := \mathbf{b}_{I_f} - \mathbf{L}_{I_f, \cup_{i=1}^f I_i} \hat{\mathbf{r}}_{\cup_{i=1}^f I_i}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)}$
- 6: Solve $\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{w}$ for $\mathbf{x}_{I_f}^{(j)}$

Algorithm 5: Exact reconstruction of the state on the replacement node f for the SPCG method (cf. Alg. 2)

where $\mathbf{A}_{I_f, I_f}$ has full rank.

Proof. Due to $\mathbf{r}^{(j)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(j)} \iff \mathbf{A}\mathbf{x}^{(j)} = \mathbf{b} - \mathbf{r}^{(j)}$ and $\hat{\mathbf{r}}^{(j)} = \mathbf{L}^{-1} \mathbf{r}^{(j)} \iff \mathbf{r}^{(j)} = \mathbf{L} \hat{\mathbf{r}}^{(j)}$, it follows that $\mathbf{A}\mathbf{x}^{(j)} = \mathbf{b} - \mathbf{L} \hat{\mathbf{r}}^{(j)}$. Since $\mathbf{A}$ is required to be an SPD matrix for the SPCG method, we can apply Lem. 2 with $\mathbf{B} := \mathbf{A}$, $\mathbf{y}^{(j)} := \mathbf{x}^{(j)}$, and $\mathbf{c}^{(j)} := \mathbf{b} - \mathbf{L} \hat{\mathbf{r}}^{(j)}$ in order to obtain the linear system $\mathbf{A}_{I_f, I_f} \mathbf{x}_{I_f}^{(j)} = \mathbf{b}_{I_f} - \mathbf{L}_{I_f, I} \hat{\mathbf{r}}_{I_f}^{(j)} - \mathbf{A}_{I_f, I \setminus I_f} \mathbf{x}_{I \setminus I_f}^{(j)}$, where $\mathbf{A}_{I_f, I_f}$ has full rank. Because $\mathbf{L}$ is a lower triangular matrix and, therefore, $\mathbf{L}_{I_f, \cup_{i=f+1}^N I_i} = \mathbf{0}$, we finally receive the asserted linear system. $\square$

After the subvectors $\hat{\mathbf{r}}_{\cup_{i=1}^f I_i}^{(j)}$ of the residual $\hat{\mathbf{r}}^{(j)}$ and $\mathbf{x}_{I \setminus I_f}^{(j)}$ of the iterate $\mathbf{x}^{(j)}$ have been gathered, the linear system of Cor. 1 can be solved locally on the replacement node f . The complete reconstruction process for the ESR-SPCG method from the perspective of the replacement node f is listed in Alg. 5. Although there are some similarities between Alg. 5 and Alg. 4, the vector elements to be communicated and the number of operations to be executed on the replacement node in the reconstruction phase clearly differ.

C. Generalization to multiple node failures

So far, we considered exactly one node failure at a time, i.e., we assumed that the reconstruction process finishes before another node failure occurs. In this section, we briefly outline how the ESR approach can be extended to multiple node failures that may overlap in time. The key idea for making the ESR technique capable of coping with up to $k < N$ node failures at a time is to keep k redundant copies of each block of the two most recent search directions $\mathbf{p}^{(j-1)}$ and $\mathbf{p}^{(j)}$ on k different compute nodes.

Chen [5] describes how to ensure one redundant copy of each block $\mathbf{p}_{I_i}^{(j-1)}$ and $\mathbf{p}_{I_i}^{(j)}$, $i \in \{0, 1, \dots, N-1\}$, such that the two most recent search directions $\mathbf{p}^{(j-1)}$ and $\mathbf{p}^{(j)}$ can be fully recovered in the event of a single node failure. We can use a similar strategy to keep k redundant copies of each block of the two most recent search directions. If we assume that the entries of the system matrix $\mathbf{A}$ are clustered around the diagonal, it would be a reasonably good strategy to keep those k redundant copies of $\mathbf{p}_{I_i}^{(j-1)}$ and $\mathbf{p}_{I_i}^{(j)}$ on nodes $i-1, i+1, i-2, i+2, \dots, i-\frac{k}{2}, i+\frac{k}{2}$ if k is even or on nodes $i-1, i+1, i-2, i+2, \dots, i-\lceil \frac{k}{2} \rceil$ if k is odd, where any rank

beyond 0 or $N - 1$ is wrapped around cyclically. A possibly more efficient scheme for keeping the redundant copies could be found when considering the actual network topology of the parallel computer.

Since then we have $k + 1$ copies of each $\mathbf{p}_{I_i}^{(j-1)}$ and $\mathbf{p}_{I_i}^{(j)}$ on $k + 1$ different nodes (including node i that owns the block), the two most recent search directions $\mathbf{p}^{(j-1)}$ and $\mathbf{p}^{(j)}$ can be fully recovered after a simultaneous or overlapping failure of up to k nodes. If the $l \leq k$ node failures do not happen simultaneously but are overlapping, i.e., more node failures occur during the reconstruction phase, the reconstruction process according to Alg. 4 or Alg. 5 must be restarted after each node failure (an efficient implementation can of course skip steps that have already been performed and are not affected by the further node failures). Let $f_1, f_2, \dots, f_l$ denote the nodes that fail. Then, we can define $I_f := I_{f_1} \cup I_{f_2} \cup \dots \cup I_{f_l}$ and use the same reconstruction procedures as in case of a single node failure.

Some of the reconstruction steps of Alg. 4 and Alg. 5 can be performed locally on each of the replacement nodes $f_1, f_2, \dots, f_l$. For computing the matrix-vector products and solving the linear systems, additional communication between the l replacement nodes is necessary. During both the failure-free execution of the linear solver and the reconstruction phase, the support for up to k node failures may incur increased memory and communication costs. Hence, k should be chosen only as large as necessary for handling the expected number of overlapping node failures on a specific parallel computer. A complete formulation of the outlined scheme would require a thorough cost analysis, which is beyond the scope of this paper.

IV. NUMERICAL EXPERIMENTS

We perform experiments comparing the ESR approach with the LI algorithm [7] in order to investigate the number of iterations until convergence after the recovery. Our experiments run in the REPEAL2 machine of the University of Vienna, in a single NUMA node of 16 CPUs. Simulations are run on 16 processes. We use PETSc [13], [21] as a framework to perform the experiments. This library was extended with implementations of the ESR algorithm of this paper and the LI algorithm.

As test cases, we selected the SPD matrices from the SuiteSparse Matrix Collection [20] summarized in Tab. I. The right-hand sides of the linear systems are generated randomly using different seeds, normalized relative to the largest entry of the matrix.

We experimentally investigate the following scenarios: • Jacobi and block Jacobi preconditioners. • Nodes with rank 0, 4, 8 and 12. • Iterations corresponding to 10%, 30%, 50%, 70% and 90% of the iteration count for the corresponding fault-free case. • Three seeds to generate the right-hand side. We also perform fault-free runs for the used random seeds.

We define the metric *relative overhead due to node failures* as $(i_f + i_r - i_0)/i_0$, where i_0 is the number of iterations without a node failure, i_f is the number of iterations from the start to

the failure, and i_r is the number of iterations from the node failure until convergence (cf. [22, Fig. 1]). The convergence criterion is a residual norm reduction by a factor of 10^5 .

Name	Type of problem	Size n	Non-zeros
1138_bus	Power network	1138	4054
nos3	Structural	2541	7361
offshore	Electromagnetics	259789	4242673
parabolic_fem	Fluid dynamics	525825	3674625
audikw_1 ^a	Structural	943695	77651847
G3_circuit	Circuit simulation	1585478	7660826

Table I: SPD matrices from [20] used in the experiments.

^aUses only two seeds, instead of three, to generate right-hand sides for the solver.

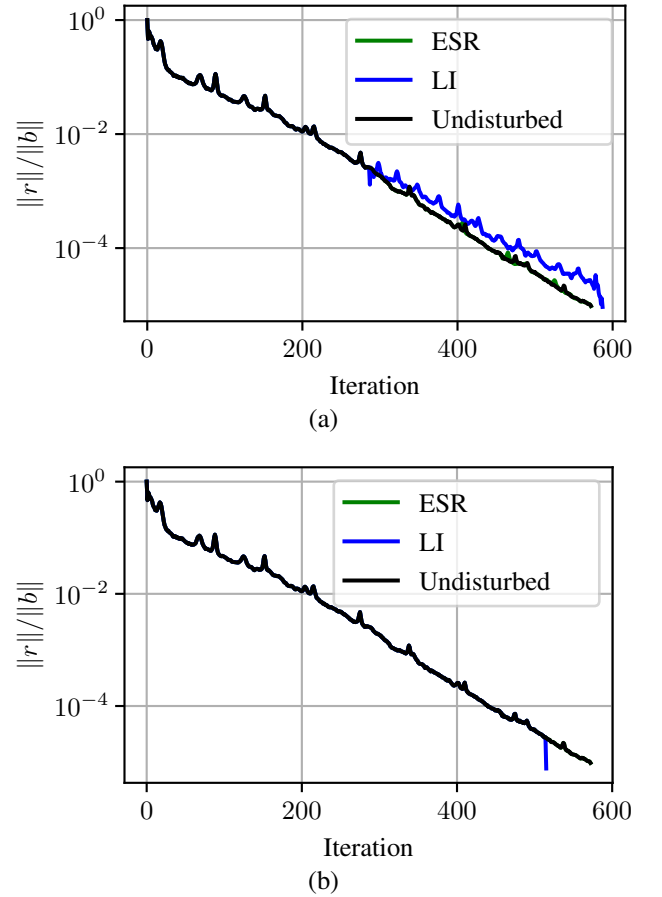


Figure 1: Residual norm history for the `offshore` matrix with the Jacobi preconditioner and the node failure happening at 50% (a) and 90% (b) of progress of the solution process. The results for ESR and LI overlap before the node failure. In (b), the residual norm drop of LI after the reconstruction makes it converge immediately in the event of a failure.

Fig. 2 show the relative overhead of ESR and LI compared

for selected matrices of Tab. I. We make some remarks. First, the relative overhead is, in general, smaller for the ESR method, usually hovering around zero. ESR also has considerably smaller variance, making the cost of the reconstruction process more predictable when compared to LI. Second, the overhead for the LI method tends to be smaller if the node failure happens close to the end of the process. In some cases, the overhead is even negative (cf. Fig. 1(b)). The explanation for this is that the residual norm always dips after the interpolation step of LI, as proven in [7] and as illustrated in Fig. 1, sometimes even fulfilling the residual criterion for convergence immediately.

The results of our experiments are summarized in Tab. II and Tab. III. The tables have the maximum, mean and minimum relative overheads for the Jacobi and block Jacobi preconditioners respectively.

V. CONCLUSIONS

In this work, we examined the PCG and SPCG methods for parallel computers that are susceptible to node failures. Previous algorithmic techniques for coping with such node failures either have the drawback of permanent overhead even in the failure-free case (like checkpoint/restart approaches) or heuristically determine a new initial guess for restarting (like interpolation/restart methods). The ESR approach, proposed by Chen [5] and refined in this work, reconstructs the complete state of the PCG or SPCG solver after a node failure, thus minimizing the iteration overhead until convergence caused by the node failure. Moreover, this approach causes only negligible or—depending on the linear system to be solved—no overhead at all if no node failure occurs.

Since each part of the search direction vector has to be available on more than one node during the SpMV operation at every PCG and SPCG iteration, redundant copies of all parts of the search direction vectors can be kept on different nodes with insignificant memory overhead. After a node failure, these redundant copies enable the inexpensive recovery of lost search direction parts. In ESR, the received local linear system for exactly reconstructing the iterate turned out to be a refinement of the LI method proposed by Agullo et al. [7], which is the most efficient interpolation/restart method for the PCG and SPCG solvers.

In numerical experiments we showed that, on average, the ESR approach has essentially zero iteration overhead in floating-point arithmetic. On the other hand, the LI method causes iteration overheads of up to 214% in our experiments. This clearly indicates that the ESR method constitutes a significant improvement over the existing techniques.

Our next step is to compare the actual runtimes of the different resilience approaches on large computer clusters. We also want to investigate how the ESR-PCG and ESR-SPCG methods behave on top of already existing, but mostly still experimental, ULFM implementations of current MPI libraries. Another interesting question is how the ESR approach can be combined with *communication-avoiding* CG algorithms like the *s-step* [23], [24] and *pipelined* [25] CG methods since

the avoidance of global communication is considered one of the key techniques, besides fault tolerance, for next-generation extreme-scale parallel computing.

REFERENCES

- [1] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, “Addressing failures in exascale computing”, *Int. J. High Perform. Comput. Appl.*, vol. 28, pp. 129–173, 2014.
- [2] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, “Failures in large scale systems: Long-term measurement, analysis, and implications”, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’17, ACM, 2017, 44:1–44:12.
- [3] T. Herault and Y. Robert, Eds., *Fault-Tolerance Techniques for High-Performance Computing*. Springer International Publishing, 2015.
- [4] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition. SIAM, 2003.
- [5] Z. Chen, “Algorithm-based recovery for iterative methods without checkpointing”, in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC ’11, ACM, 2011, pp. 73–84.
- [6] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra, “Recovery patterns for iterative methods in a parallel unstable environment”, *SIAM J. Sci. Comput.*, vol. 30, pp. 102–116, 2007.
- [7] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon, “Numerical recovery strategies for parallel resilient Krylov linear solvers”, *Numer. Lin. Algebra. Appl.*, vol. 23, pp. 888–905, 2016.
- [8] Message Passing Interface Forum, *MPI: A message-passing interface standard*, <https://www.mpi-forum.org/docs/>, 2015.
- [9] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra, “Post-failure recovery of mpi communication capability: Design and rationale”, *Int. J. High Perform. Comput. Appl.*, vol. 27, pp. 244–254, 2013.
- [10] Message Passing Interface Forum, *User level failure mitigation*, <http://fault-tolerance.org/>, 2017.
- [11] G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra, “Assessing the impact of ABFT and checkpoint composite strategies”, in *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, 2014, pp. 679–688.
- [12] G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra, “Composing resilience techniques: ABFT, periodic and incremental checkpointing”, *IJNC*, vol. 5, pp. 2–25, 2015.

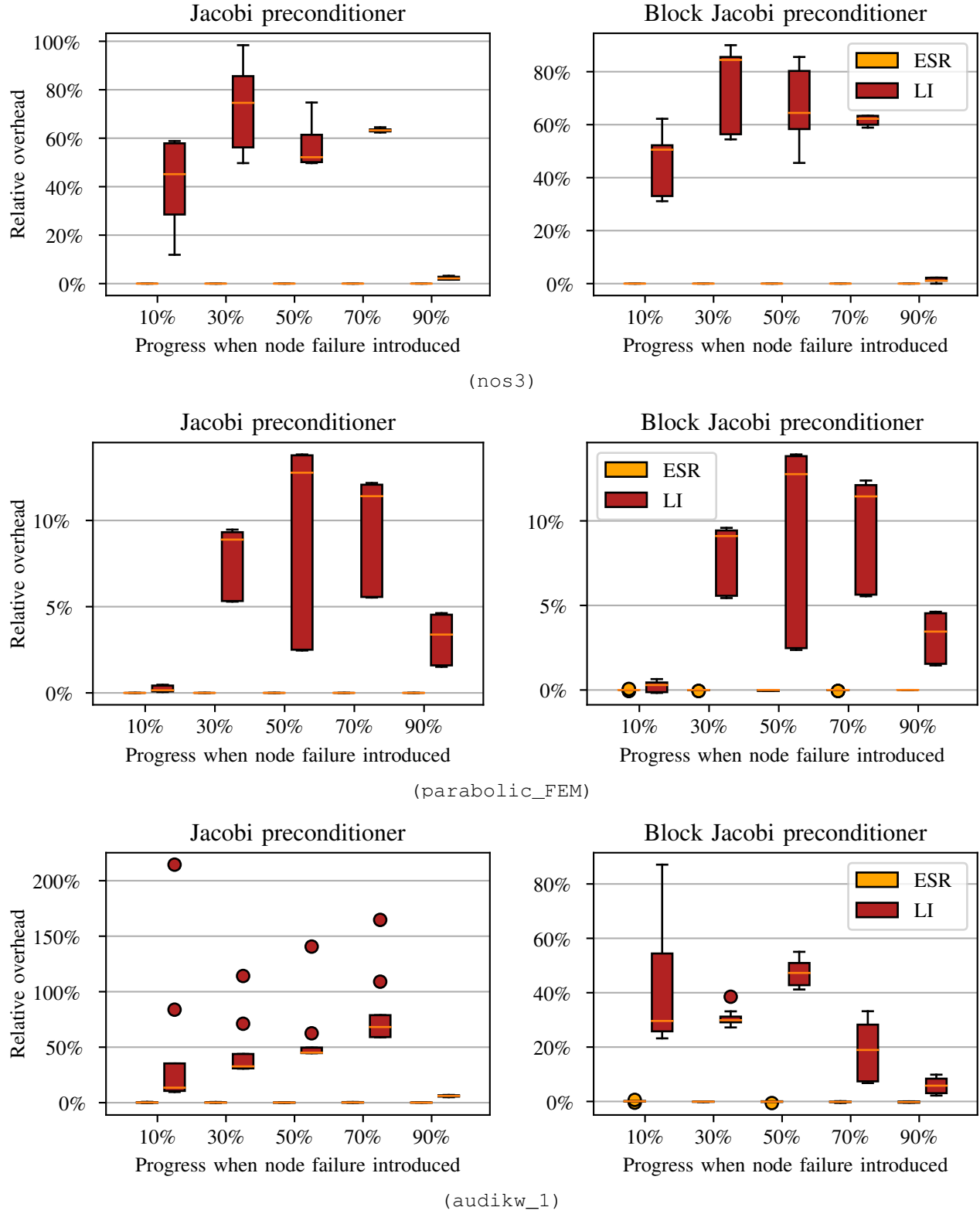


Figure 2: Relative overhead barplot for selected matrices. The boxes contain points between Q1 and Q3. The whiskers extend to the farthest point within $1.5 \cdot (Q3 - Q1)$ above and below the boxes. Outliers are represented with points.

	1138	AudiKW 1	G3 Circuit	NOS3	Offshore	Parabolic FEM
Iterations if no failure (Avg)	934.0	6058.5	2009.3	186.0	1292.7	2019.3
Max	2.04%	0.74%	0.00%	0.00%	0.61%	0.00%
ESR Mean	0.25%	0.15%	0.00%	0.00%	0.01%	0.00%
Min	0.00%	-0.35%	0.00%	0.00%	-0.24%	0.00%
Max	75.16%	214.46%	14.99%	98.39%	13.84%	13.83%
LI Mean	31.28%	48.25%	4.90%	47.79%	4.32%	6.13%
Min	10.72%	4.94%	-0.63%	1.62%	0.15%	0.05%

Table II: Aggregated results for the relative overhead for the ESR and LI algorithms for the Jacobi preconditioner

	1138	AudiKW 1	G3 Circuit	NOS3	Offshore	Parabolic FEM
Iterations if no failure (Avg)	656.3	2133.0	669.3	90.0	579.0	1772.7
Max	4.86%	0.58%	0.00%	0.00%	1.24%	0.06%
ESR Mean	1.51%	-0.10%	0.00%	0.00%	0.16%	-0.01%
Min	-0.31%	-0.59%	0.00%	0.00%	0.00%	-0.06%
Max	82.72%	87.10%	20.03%	90.00%	7.68%	13.93%
LI Mean	51.31%	28.85%	3.51%	50.30%	1.18%	6.18%
Min	10.11%	2.18%	-9.97%	0.00%	-9.95%	-0.17%

Table III: Aggregated results for the relative overhead for the ESR and LI algorithms for the block Jacobi preconditioner

- [13] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object-oriented numerical software libraries", in *Modern Software Tools for Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Boston, 1997, pp. 163–202.
- [14] P. Sao and R. Vuduc, "Self-stabilizing iterative solvers", in *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. *Scala '13*, ACM, 2013, 4:1–4:8.
- [15] G. Bronevetsky and B. R. de Supinski, "Soft error vulnerability of iterative linear algebra methods", in *Proceedings of the 22nd Annual International Conference on Supercomputing*, ACM, 2008, pp. 155–164.
- [16] K. Dichev and D. S. Nikolopoulos, "TwinPCG: Dual thread redundancy with forward recovery for preconditioned conjugate gradient methods", in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 506–514.
- [17] M. Shantharam, S. Srinivasmurthy, and P. Raghavan, "Fault tolerant preconditioned conjugate gradient for sparse linear system solution", in *Proceedings of the 26th ACM International Conference on Supercomputing*, ser. *ICS '12*, ACM, 2012, pp. 69–78.
- [18] M. Fasi, J. Langou, Y. Robert, and B. Uçar, "A backward/forward recovery approach for the preconditioned conjugate gradient method", *J. Comput. Sci.*, pp. 522–534, 2016.
- [19] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon, "Towards resilient parallel linear Krylov solvers: recover-restart strategies", INRIA, Research Report, 2013.
- [20] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection", *ACM Trans. Math. Software*, vol. 38, 1:1–1:25, 2011.
- [21] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc users manual", Argonne National Laboratory, Tech. Rep., 2018.
- [22] C. Pachajoa and W. N. Gansterer, "On the resilience of conjugate gradient and multigrid methods to node failures", in *Euro-Par 2017: Parallel Processing Workshops*, Springer, 2018, pp. 569–580.
- [23] A. T. Chronopoulos and C. W. Gear, "S-step iterative methods for symmetric linear systems", *J. Comput. Appl. Math.*, vol. 25, pp. 153–168, 1989.
- [24] M. F. Hoemmen, "Communication-avoiding krylov subspace methods", University of California at Berkeley, PhD thesis, 2010.
- [25] P. Ghysels and W. Vanroose, "Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm", *Parallel Computing*, vol. 40, pp. 224–238, 2014.