How to Make the Preconditioned Conjugate Gradient Method Resilient Against Multiple Node Failures

Carlos Pachajoa
University of Vienna
Faculty of Computer Science
Vienna, Austria
carlos.pachajoa@univie.ac.at

Markus Levonyak
University of Vienna
Faculty of Computer Science
Vienna, Austria
markus.levonyak@univie.ac.at

Wilfried N. Gansterer*
University of Vienna
Faculty of Computer Science
Vienna, Austria
wilfried.gansterer@univie.ac.at

Jesper Larsson Träff
TU Wien
Faculty of Informatics
Vienna, Austria
traff@par.tuwien.ac.at

ABSTRACT

We study algorithmic approaches for recovering from the failure of several compute nodes in the parallel preconditioned conjugate gradient (PCG) solver on large-scale parallel computers. In particular, we analyze and extend an exact state reconstruction (ESR) approach, which is based on a method proposed by Chen (2011). In the ESR approach, the solver keeps redundant information from previous search directions, so that the solver state can be fully reconstructed if a node fails unexpectedly. ESR does not require checkpointing or external storage for saving dynamic solver data and has low overhead compared to the failure-free situation.

In this paper, we improve the fault tolerance of the PCG algorithm based on the ESR approach. In particular, we support recovery from simultaneous or overlapping failures of several nodes for general sparsity patterns of the system matrix, which cannot be handled by Chen's method. For this purpose, we refine the strategy for how to store redundant information across nodes. We analyze and implement our new method and perform numerical experiments with large sparse matrices from real-world applications on 128 nodes of the Vienna Scientific Cluster (VSC). For recovering from three simultaneous node failures we observe average runtime overheads between only 2.8% and 55.0%. The overhead of the improved resilience depends on the sparsity pattern of the system matrix.

CCS CONCEPTS

• **Mathematics of computing** → **Solvers**; **Mathematical software performance**; • **Computing methodologies** → **Parallel algorithms**.

*Corresponding author

ICPP 2019, August 5–8, 2019, Kyoto, Japan

ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337849

ACM Reference Format:
Carlos Pachajoa, Markus Levonyak, Wilfried N. Gansterer, and Jesper Larsson Träff. 2019. How to Make the Preconditioned Conjugate Gradient Method Resilient Against Multiple Node Failures. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337849

1 INTRODUCTION

One of the major challenges in current and, even more, in future high-performance computing (HPC) is the increasing failure rate caused by increasing complexity and an ever-growing number of interconnected components [16, 29]. Among the most common types of failures in large-scale parallel computers are *fail-stop failures*, where a failing process stops and its data is lost [11, 14, 26, 27]. In order to support long-running scientific applications also on failure-prone HPC systems, new strategies and resilient algorithms are necessary [17].

We consider the *preconditioned conjugate gradient* (PCG) method (cf. Sec. 2.1), an important iterative method for solving symmetric and positive-definite (SPD) sparse linear systems on large-scale parallel computers. We study settings where the PCG solver is executed on parallel computers that are susceptible to *node failures*, a common type of fail-stop failures which is critical in practice [17]. In contrast to previous work that also aims to make the PCG method resilient against unexpected node failures without expensive checkpointing (cf. Sec. 1.2), we do not only consider single node failures but multiple node failures which occur simultaneously or are overlapping in time. The *exact state reconstruction* (ESR) approach (cf. Sec. 2.2), which was introduced by Chen [11] and refined by Pachajoa et al. [23], was shown to be the most efficient checkpointing-free algorithmic fault-tolerance technique for protecting the PCG solver against unexpected single node failures [23].

In this paper, we assume that a large sparse linear system $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is SPD, is given and shall be solved with the PCG method on a parallel computer. The preconditioner $\boldsymbol{M}$ for solving this linear system is either explicitly or implicitly given. We enhance the ESR approach so that it becomes capable of protecting the PCG method against unexpected simultaneous or overlapping failures of multiple nodes, theoretically analyze the communication overhead for this improved resilience, and demonstrate low runtime overheads

in numerical experiments. Although we cannot provide details due to space restrictions, our proposed algorithmic modifications can also be applied to the ESR approach [11] for the *Jacobi*, *Gauss-Seidel*, *successive overrelaxation* (SOR), *symmetric successive overrelaxation* (SSOR), *split preconditioner conjugate gradient* (SPCG) [23] and *preconditioned bi-conjugate gradient stabilized* (BiCGSTAB) algorithms in order to make them resilient against multiple simultaneous or overlapping node failures.

1.1 Problem setting and assumptions

We consider the parallel execution of the PCG solver on N compute nodes of a distributed-memory parallel computer, which communicate over an interconnection network. Each compute node consists of m processors such that the solver is executed on $M := N \times m$ processors in total. Each processor shares its memory with all other processors on the same compute node.

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 take over the workload of the failed processor. The same holds for *multiple processor failures* where at least one processor per compute node survives. In this paper, we consider a more complicated event: a *node failure*, where a compute node fails as a whole and the data in the memory of the affected node is lost.

Node failures may occur for several reasons: 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 talk about *multiple node failures*. Nodes that continue working after a node failure and keep all their data are called *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 called a *replacement node* (which could be a spare node or one of the surviving nodes).

1.1.1 Failure detection and node replacement. 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) [20]. This particularly comprises efficient collective communication capabilities like the *Allreduce* function of MPI. 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) [6, 21] 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 which nodes have failed, and a mechanism for providing replacement nodes.

1.1.2 Data distribution. Analogously to [11], we assume that the fundamental problem-defining static input data, i.e., the system matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, the right-hand-side vector $\boldsymbol{b} \in \mathbb{R}^n$, and the preconditioner $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ (cf. Sec. 2.1), can be retrieved from reliable external storage (e.g., from a checkpoint prior to entering the linear solver, cf. the *ABFT&PeriodicCkpt* algorithm [7, 8, 17], see Sec. 1.2), and thus does not have to be reconstructed after a node failure.

In accordance with typical distributions of sparse matrices in widely used high-performance numerical libraries like PETSc [4], we consider a block-row data distribution of all matrices and vectors among the N nodes of the parallel computer. More precisely, every node owns blocks of n/N contiguous rows (if $n = cN$ with $c \in \mathbb{N}$, otherwise some nodes own $\lfloor n/N \rfloor$ and others $\lceil n/N \rceil$ rows) of both the matrices $\boldsymbol{A}$ and $\boldsymbol{M}$ as well as each of the vectors $\boldsymbol{b}, \boldsymbol{x}$, and all other vectors maintained by the PCG solver (cf. Sec. 2.1). On one 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 node owns block rows of all matrices and vectors, a node failure affects a part of each matrix and vector. Block rows of dynamic data which were owned by the failed node are lost and need to be reconstructed on the replacement node (cf. Sec. 2.2).

1.1.3 Notation. We use a notation similar to [2] to denote sections of matrices and vectors. We refer to the set of all indices as $I := \{1, 2, \ldots, n\}$. 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 $\boldsymbol{v}^{(j)}$, where j denotes the iteration number of the linear solver, $\boldsymbol{v}^{(j)}_{I_i}$ refers to the subset of elements of the vector at iteration j owned by node i. Row and column selections of a matrix $\boldsymbol{B}$ are designated with index sets as well: $\boldsymbol{B}_{I_i, I_k}$ refers to the selection of rows and columns of $\boldsymbol{B}$ 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. With the *state* of an iterative solver we mean the—not necessarily minimal—set of data that completely defines the future behavior of this iterative solver. The state of the PCG solver in iteration j can be defined as comprising the iterate $\boldsymbol{x}^{(j)}$ (i.e., the current approximation to the solution $\boldsymbol{x}$), the residual $\boldsymbol{r}^{(j)}$, the preconditioned residual $\boldsymbol{z}^{(j)}$, and the search direction $\boldsymbol{p}^{(j)}$.

1.2 Related work

The existing literature on resilient iterative linear solvers distinguishes between two different kinds of failures: soft errors and node failures. The former refers to spontaneous changes of the state of the solver (e.g., bit flips), potentially leading to a wrong result. In this category, we find work by Sao and Vuduc [25] which proposes strategies to ensure that the *conjugate gradient* (CG) method will converge to the right solution after a soft error. However, their approach requires some operations to be performed reliably.

Bronevetsky and de Supinski [9] evaluate the effects of soft errors on iterative linear solvers including the CG method. Bit flips are introduced at random times and positions, and the effects are classified according to the resulting runtimes and solution errors. Dichev and Nikolopoulos [13] 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. [28] and Fasi et al. [15]. All these approaches have in common that they are not applicable to our problem of the PCG solver subject to node failures.

Pachajoa and Gansterer [22] experimentally evaluate the inherent resilience of the CG method after a single node failure. The currently in practice most commonly used class of fault-tolerance techniques to cope with node failures is *checkpoint/restart* (*C/R*). These techniques frequently save the current state of a running application and roll back to the latest saved state in case of a node failure. C/R has been investigated as a general-purpose technique (e.g. [30]), and also for specific problem settings (e.g. [19]). Herault et al. [17] provide a comprehensive overview of different variants of this approach.

To avoid the overhead of continuously saving the state, Chen [11] exploits specific properties of the PCG solver and other iterative methods such that the state of the solver can be recovered without checkpointing after a single node failure occurred. For storing the necessary redundant information, he chooses the intuitive approach of sending it to the closest node (cf. Sec. 3). Pachajoa et al. [23] refine Chen's approach [11] by distinguishing different common types of preconditioners.

In [23], the approach from [11] is experimentally compared to a heuristic strategy proposed by Langou et al. [18]. This heuristic *interpolation/restart* strategy is applicable to iterative linear solvers in general to recover from a node failure by approximating the lost iterate. The submatrix of the replacement node is used to produce an interpolated approximation of the iterate before the node failure occurred. Agullo et al. [1, 2] extend this approach by using all the information of the matrix in the interpolation, 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.

Bosilca et al. [7, 8, 17] introduce the *ABFT&PeriodicCkpt* algorithm, which combines *algorithm-based fault tolerance* (ABFT) with periodic checkpointing in order to make entire applications (and not just operations that can be protected by ABFT) resilient to node failures. The longer the phases protected by ABFT, the fewer checkpoints are necessary and the cheaper resilience against node failures usually becomes (assuming that the used ABFT method is more efficient than checkpointing).

1.3 Contributions of this work

To the best of our knowledge, there has been neither a detailed discussion nor a thorough analysis of a generalized ESR approach for protecting the PCG method against *multiple* node failures. In this paper, we propose a strategy that precisely defines how and where to store redundant information so that the PCG solver becomes resilient against multiple node failures. In particular, we analyze the communication overhead and evaluate the performance penalty for the improved resilience capabilities of being able to tolerate multiple simultaneous node failures. In numerical experiments on the *Vienna Scientific Cluster* (VSC), a medium-scale HPC system, we eventually demonstrate the low runtime overhead of our new strategies for the improved resilience.

The remainder of this paper is organized as follows. In Sec. 2, we briefly review the PCG method and the ESR approach as presented in [11, 23]. Afterwards, in Sec. 3, we summarize the concept of Chen [11] for keeping redundant information in the ESR approach such that single node failures can be tolerated. Then, in Sec. 4, we propose modifications and extensions of the ESR approach for systems that are prone to multiple simultaneous or overlapping node failures. Apart from that, we theoretically analyze the communication overhead of our novel algorithm for supporting multiple node failures. In Sec. 5, we discuss the most important aspects regarding the impact of the sparsity pattern of the system matrix. Next, in Sec. 6, we summarize our implementation for conducting numerical experiments. Subsequently, in Sec. 7, we present our experimental results and discuss how they relate to our theoretical analysis. Finally, our conclusions are summarized in Sec. 8.

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

Algorithm 1: Preconditioned conjugate gradient (PCG) method [24, Alg. 9.1]

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

Algorithm 2: ESR reconstruction phase for the PCG method on the replacement node f ($\boldsymbol{P} := \boldsymbol{M}^{-1}$ is given) [23, Alg. 4]

2 ALGORITHMIC BACKGROUND

In this section, we first review the PCG method in Sec. 2.1 and afterwards the ESR approach [11, 23] in Sec. 2.2. In Sec. 3, we then look at the details of how Chen [11] is keeping redundant information in order to guarantee protection against single node failures.

2.1 Preconditioned conjugate gradient method

The (P)CG method iteratively solves a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where both the SPD matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and the right-hand-side vector $\boldsymbol{b} \in \mathbb{R}^n$ are given. The search directions $\boldsymbol{p}^{(j)}$, along which the quadratic potential defined by $\boldsymbol{A}$ is minimized, are chosen to be

$\boldsymbol{A}$-orthogonal, i.e., $\boldsymbol{p}^{(j)\mathsf{T}}\boldsymbol{A}\boldsymbol{p}^{(k)} = 0$ for all $j \neq k$. The residual $\boldsymbol{r}^{(j)}$ is defined as $\boldsymbol{r}^{(j)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(j)}$.

In the PCG method, which is listed in Alg. 1, a preconditioner $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is used for accelerating convergence. Instead of the original system, the linear system $\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}$ is solved. $\boldsymbol{M}$ is assumed to be an SPD matrix as well [24, p. 276] and is to be chosen such that $\kappa(\boldsymbol{M}^{-1}\boldsymbol{A}) < \kappa(\boldsymbol{A})$, where $\kappa(\boldsymbol{B})$ denotes the condition number of a matrix $\boldsymbol{B}$. The PCG method stores the distributed vectors $\boldsymbol{x}^{(j)}$, $\boldsymbol{r}^{(j)}$, $\boldsymbol{z}^{(j)}$, $\boldsymbol{p}^{(j)}$, and $\boldsymbol{A}\boldsymbol{p}^{(j)}$ as well as the replicated scalars $\alpha^{(j)}$ and $\beta^{(j)}$, which in total require memory for $5n + 2N$ floating-point numbers (not including the static data $\boldsymbol{A}$, $\boldsymbol{M}$, and $\boldsymbol{b}$).

2.2 Exact state reconstruction (ESR)

In contrast to checkpoint/restart methods, which—even in the failure-free case—impose a usually considerable runtime overhead due to continuously saving the state of the solver [17], the ESR approach [11, 23] is able to exploit the algorithmic properties of the PCG solver so that the complete state can be reconstructed after a node failure with low overhead. During the failure-free PCG iterations, only little or—depending on the sparsity structure of $\boldsymbol{A}$—no additional communication is necessary for achieving this. Only the local memory requirements are slightly higher than in the standard (non-resilient) PCG variant.

At iteration j of the PCG algorithm, the product $\boldsymbol{A}\boldsymbol{p}^{(j)}$ is computed in a sparse matrix-vector multiplication (SpMV) operation (cf. Alg. 1, lines 3 and 5). In the non-resilient PCG solver, all but the own block $\boldsymbol{p}_{I_i}^{(j)}$ can be dropped on node i after the product has been computed. In the resilient algorithm, we also drop most of $\boldsymbol{p}^{(j)}$ on each node after the SpMV. The crucial difference to non-resilient PCG is that now each node also stores the elements of at least one other node in addition to its own block (for details see Sec. 3).

Overall, there is a redundant copy of each element of $\boldsymbol{p}^{(j)}$ after computing $\boldsymbol{A}\boldsymbol{p}^{(j)}$. 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. For the reconstruction of the complete state after a node failure, we need to recover the two most recent search directions [11, Sec. 5.2] and thus have to keep redundant copies of each element of not only $\boldsymbol{p}^{(j)}$ but also $\boldsymbol{p}^{(j-1)}$. The local memory overhead of $2n/N$ vector elements per node and $2n$ vector elements in total is negligible compared to the overall memory requirement $O(n^2)$ of the PCG solver (cf. Sec. 2.1).

The procedure to reconstruct the complete state of the PCG solver is outlined in Alg. 2. It is assumed that a preconditioner $\boldsymbol{P} := \boldsymbol{M}^{-1}$ is given. Variants for cases where $\boldsymbol{M}$ (not $\boldsymbol{M}^{-1}$) or a split preconditioner $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathsf{T}}$ is given are shown in [23, Alg. 3 and 5]. The scalar $\beta^{(j-1)}$ can easily be recovered since it is replicated on every node (cf. Sec. 2.1), i.e., $\beta^{(j-1)}$ has the same value on all nodes. Furthermore, the reconstruction of the lost parts $\boldsymbol{r}_{I_f}^{(j)}$ and $\boldsymbol{z}_{I_f}^{(j)}$ of the (preconditioned) residuals $\boldsymbol{r}^{(j)}$ and $\boldsymbol{z}^{(j)}$ takes place on the replacement node f. Matrix $\boldsymbol{A}_{I_f,I_f}$ of the linear system in line 8 of Alg. 2 is SPD, has full rank, and is much smaller than the full system matrix $\boldsymbol{A}$. Therefore, this linear system can be solved locally on the replacement node f (only the involved vectors need to be gathered first on node f). Detailed derivations of the ESR approach can be found in [11, 23].

3 SINGLE NODE FAILURE

We now discuss details of how and where to store the redundant copies of the two most recent search directions. For this purpose, we review the strategy proposed by Chen [11] for protecting the PCG method against a single node failure. However, as we will see, this strategy is not suitable for multiple simultaneous or overlapping node failures. Later, in Sec. 4, we present our new strategy for handling multiple simultaneous or overlapping node failures.

In PCG, the SpMV operation for obtaining $\boldsymbol{u}^{(j)} = \boldsymbol{A}\boldsymbol{p}^{(j)}$, which can be rewritten as

$$\begin{aligned}
\boldsymbol{u}_{I_1}^{(j)} &= \boldsymbol{A}_{I_1,I_1}\boldsymbol{p}_{I_1}^{(j)} + \boldsymbol{A}_{I_1,I_2}\boldsymbol{p}_{I_2}^{(j)} + \cdots + \boldsymbol{A}_{I_1,I_N}\boldsymbol{p}_{I_N}^{(j)} \\
\boldsymbol{u}_{I_2}^{(j)} &= \boldsymbol{A}_{I_2,I_1}\boldsymbol{p}_{I_1}^{(j)} + \boldsymbol{A}_{I_2,I_2}\boldsymbol{p}_{I_2}^{(j)} + \cdots + \boldsymbol{A}_{I_2,I_N}\boldsymbol{p}_{I_N}^{(j)} \\
&\vdots \\
\boldsymbol{u}_{I_N}^{(j)} &= \boldsymbol{A}_{I_N,I_1}\boldsymbol{p}_{I_1}^{(j)} + \boldsymbol{A}_{I_N,I_2}\boldsymbol{p}_{I_2}^{(j)} + \cdots + \boldsymbol{A}_{I_N,I_N}\boldsymbol{p}_{I_N}^{(j)},
\end{aligned} \tag{1}$$

is performed at each iteration (cf. lines 3 and 5 of Alg. 1). When ignoring possible optimizations due to the sparsity pattern of $\boldsymbol{A}$, $\boldsymbol{p}_{I_i}^{(j)}$ is sent from node i to node k in iteration j such that $\boldsymbol{u}_{I_k}^{(j)} = \boldsymbol{A}_{I_k,I_1}\boldsymbol{p}_{I_1}^{(j)} + \cdots + \boldsymbol{A}_{I_k,I_i}\boldsymbol{p}_{I_i}^{(j)} + \cdots + \boldsymbol{A}_{I_k,I_N}\boldsymbol{p}_{I_N}^{(j)}$ can be computed on node k. For a more optimized algorithm that, during SpMV, only sends the minimum set of elements required due to the sparsity pattern of $\boldsymbol{A}$, we define

$$\begin{aligned}
S_i &:= \text{all elements of } \boldsymbol{p}_{I_i}^{(j)}, \\
S_{ik} &:= \text{elements of } \boldsymbol{p}_{I_i}^{(j)} \text{ sent to node } k \text{ computing } \boldsymbol{A}\boldsymbol{p}^{(j)}, \\
R_i &:= \left(\bigcup_{k=1}^{i-1} S_{ik}\right) \cup \left(\bigcup_{k=i+1}^{N} S_{ik}\right), \text{ and } R_i^c := S_i - R_i
\end{aligned} \tag{2}$$

in accordance with Chen [11]. $\boldsymbol{p}_{I_i}^{(j)}$ can be completely recovered after a node failure if $R_i = S_i$. In order to ensure this, Chen proposes to send R_i^c to node $d_i := (i+1) \bmod N$ (together with S_{id_i}).

Unfortunately, this strategy is not capable of coping with simultaneous or overlapping failures of multiple nodes. For example, if both nodes i and $i+1$ fail simultaneously and $R_i^c \neq \emptyset$, the search direction vector elements in R_i^c are lost and the state of the PCG solver cannot be reconstructed. The problem obviously worsens if more than two nodes fail simultaneously.

4 MULTIPLE NODE FAILURES

In this section, we extend the ESR approach for coping with multiple simultaneous or overlapping node failures. Originally, the ESR method considers exactly one node failure at a time, i.e., it is assumed that the reconstruction process finishes before another node failure occurs (cf. Sec. 2.2 and Sec. 3). For coping with up to $\phi < N$ uniformly distributed node failures that may overlap in time, we need to keep ϕ redundant copies of each block of the two most recent search directions $\boldsymbol{p}^{(j-1)}$ and $\boldsymbol{p}^{(j)}$ on ϕ different compute nodes [23]. In the following, we design a resilient algorithm based on this idea in detail and analyze its communication overhead.

4.1 Tolerating multiple node failures

To keep ϕ redundant copies of each block of the two most recent search direction vectors, a similar strategy can be pursued as in the special case $\phi = 1$. Let (S_i, m_i) be a multiset with the multiplicity

$$\begin{aligned} m_i \colon S_i &\to \mathbb{N}_0 \\ s &\mapsto \text{number of nodes } s \text{ is sent to} \\ &\quad \text{during the computation of } \boldsymbol{A}\boldsymbol{p}^{(j)}. \end{aligned} \tag{3}$$

Note that we assume here—as it is common for SpMV—that s is only sent to nodes with corresponding non-zero entries in their rows of $\boldsymbol{A}$. Hence, the number of nodes s is sent to depends on the sparsity pattern of $\boldsymbol{A}$. Comparing the definition of the multiplicity m in Eqn. (3) with the definition of R_i^c in Eqn. (2), it follows that

$$R_i^c = \{s \in S_i \mid m_i(s) = 0\}. \tag{4}$$

For supporting up to ϕ simultaneous (or overlapping) node failures, we need to store at least ϕ redundant copies of each element of $\boldsymbol{p}_{I_i}^{(j)}$, $i \in \{1, 2, \ldots, N\}$, on ϕ different nodes other than node i. Let

$$d_{ik} \coloneqq \begin{cases} \left(i + \left\lceil \frac{k}{2} \right\rceil\right) \bmod N, & \text{if } k \text{ odd} \\ \left(i - \frac{k}{2}\right) \bmod N, & \text{if } k \text{ even} \end{cases} \tag{5}$$

and let $g_i(s)$ denote the number of sets $S_{id_{ik}}$ with $s \in S_{id_{ik}}$ for all $k \in \{1, 2, \ldots, \phi\}$. Then, the required redundancy for tolerating up to ϕ simultaneous node failures can be achieved by sending

$$R_{ik}^c \coloneqq \left\{ s \in S_i \;\middle|\; s \notin S_{id_{ik}} \;\wedge\; m_i(s) - g_i(s) \leq \phi - k \right\} \tag{6}$$

to node d_{ik} for all $i \in \{1, 2, \ldots, N\}$ and $k \in \{1, 2, \ldots, \phi\}$. Note that the sets R_{ik}^c are of minimal size such that the required number ϕ of redundant copies of each search direction vector element is ensured. It holds that $|R_{i1}^c| \geq |R_{i2}^c| \geq \cdots \geq |R_{i\phi}^c|$.

The strategy proposed in Eqn. (5) for selecting the nodes to keep the redundant copies of $\boldsymbol{p}_{I_i}^{(j-1)}$ and $\boldsymbol{p}_{I_i}^{(j)}$ is a reasonably good heuristic for minimizing communication overheads during SpMV if we assume that the entries of the system matrix $\boldsymbol{A}$ are mostly clustered around the diagonal (since it then is likely that there are some elements which have to be sent anyway from node i to node d_{ik} and, thus, there is no extra latency for establishing a new connection; see Sec. 5 for a more detailed discussion). For matrices with very different sparsity patterns, strategies different from Eqn. (5) may be preferable. A comprehensive analysis of the interaction between a given sparsity pattern and the optimal choice of the "backup nodes" is work in progress.

When the backup strategy based on Eqns. (5) and (6) is employed, we have $\phi + 1$ copies of each element of $\boldsymbol{p}_{I_i}^{(j-1)}$ and $\boldsymbol{p}_{I_i}^{(j)}$ on $\phi + 1$ different nodes (including node i that owns the block) and the two most recent search directions $\boldsymbol{p}^{(j-1)}$ and $\boldsymbol{p}^{(j)}$ can be fully recovered after a simultaneous or overlapping failure of up to ϕ arbitrary nodes. If the node failures do not happen simultaneously but are overlapping in time, i.e., more node failures occur during the reconstruction phase, the reconstruction process 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 subsequent node failures).

We consider $\psi \leq \phi$ node failures. Let $f_1, f_2, \ldots, f_\psi$ denote the nodes that fail. Then, we can define $I_f \coloneqq I_{f_1} \cup I_{f_2} \cup \cdots \cup I_{f_\psi}$ and use a similar reconstruction procedure as in the case of a single node failure. Some of the reconstruction steps of Alg. 2 can be performed locally on each of the replacement nodes $f_1, f_2, \ldots, f_\psi$. However, for computing the matrix-vector products and solving the linear systems in lines 5 to 8 of Alg. 2, additional communication between the ψ replacement nodes is necessary. In Sec. 4.2, we show analytically that the capability of tolerating up to ϕ node failures may incur increased communication cost. Hence, ϕ should be chosen only as large as necessary for handling the expected number of simultaneous or overlapping node failures on a given parallel computer.

4.2 Analysis

Communication time usually is the main cost for a parallel algorithm, dominating the computation time [5]. For analyzing the communication overhead of sending the additional elements of the sets R_{ik}^c as defined in Eqn. (6), we adopt a latency-bandwidth communication model [10] and assume that the communication cost solely depends on latencies $\lambda_{ik} > 0$—which may vary for different sending nodes i and corresponding receiving nodes d_{ik} according to Eqn. (5)—and cost $\mu > 0$ per vector element. We further assume that each node is able to send and receive exactly one element at a time. If $S_{id_{ik}} \neq \emptyset$, the additional elements of R_{ik}^c are sent together with the elements of $S_{id_{ik}}$, which have to be sent anyway during computing the sparse matrix-vector product $\boldsymbol{A}\boldsymbol{p}^{(j)}$.

If this is the case for all pairs of nodes for a fixed $k \in \{1, 2, \ldots, \phi\}$, which we refer to as communication round k, no extra latency cost applies in that round, and the overhead is $\max_i |R_{ik}^c|\mu$. In contrast, if $\forall i \in \{1, 2, \ldots, N\} \colon S_{id_{ik}} = \emptyset$ in communication round $k \in \{1, 2, \ldots, \phi\}$, extra latencies λ_{ik} incur for all pairs of nodes, and the overhead is $\max_i(\lambda_{ik} + |R_{ik}^c|\mu) \leq \max_i \lambda_{ik} + \max_i |R_{ik}^c|\mu$. Hence, in communication round $k \in \{1, 2, \ldots, \phi\}$, it holds for the communication overhead O that

$$\begin{aligned} 0 &\leq \max_i \left|R_{ik}^c\right| \mu \leq O \\ &\leq \max_i \left(\lambda_{ik} + \left|R_{ik}^c\right| \mu\right) \leq \max_i \lambda_{ik} + \max_i \left|R_{ik}^c\right| \mu, \end{aligned}$$

where $i \in \{1, 2, \ldots, N\}$ and $\max_i |R_{ik}^c|\mu = 0$ if and only if $\forall i \colon |R_{ik}^c| = 0$, i.e., there are at least $\phi - k + 1$ redundant copies of all elements of $\boldsymbol{p}^{(j)}$ due to the sparsity pattern of $\boldsymbol{A}$. For all ϕ communication rounds, it follows that

$$\begin{aligned} 0 &\leq \max_i \sum_{k=1}^{\phi} \left|R_{ik}^c\right| \mu \leq \sum_{k=1}^{\phi} \max_i \left|R_{ik}^c\right| \mu \leq O \\ &\leq \sum_{k=1}^{\phi} \max_i \left(\lambda_{ik} + \left|R_{ik}^c\right| \mu\right) \leq \sum_{k=1}^{\phi} \left(\max_i \lambda_{ik} + \max_i \left|R_{ik}^c\right| \mu\right) \\ &= \sum_{k=1}^{\phi} \underbrace{\max_i \lambda_{ik}}_{\leq \lambda_{\max}} + \sum_{k=1}^{\phi} \underbrace{\max_i \left|R_{ik}^c\right|}_{\leq \lceil \frac{n}{N} \rceil} \mu \leq \phi \lambda_{\max} + \phi \left\lceil \frac{n}{N} \right\rceil \mu, \end{aligned}$$

where $\lambda_{\max} \coloneqq \max_{i,k} \lambda_{ik}$. Hence, the communication overhead O for keeping ϕ redundant copies of all elements of $\boldsymbol{p}^{(j)}$ lies between

the lower bound 0 and the upper bound $\phi\,(\lambda_{\max} + \lceil \frac{n}{N} \rceil \mu)$. The actual communication overhead within that interval entirely depends on the sparsity pattern of $\boldsymbol{A}$, which determines the elements in R^c_{ik}.

5 INFLUENCE OF THE SPARSITY PATTERN

In this section, we take a closer look at the implications of our theoretical analysis in Sec. 4. As we showed in Sec. 4.2, there exist matrices with sparsity patterns which lead to *zero* communication overhead during the failure-free execution of the PCG solver, i.e., no extra communication is necessary for distributing ϕ redundant copies of all elements of the search direction vector $\boldsymbol{p}^{(j)}$ during the computation of the sparse matrix-vector product $\boldsymbol{A}\boldsymbol{p}^{(j)}$. On the other hand, other matrix sparsity patterns may lead to significant communication overhead during the SpMV operation.

Independent of the particular strategy for selecting the backup nodes, it is clearly beneficial for having a low communication overhead if $m_i(s) \geq \phi$ (cf. Eqn. (3)) holds for most or even all $s \in S_i$ and $i \in \{1, 2, \ldots, N\}$ or, in other words, if most or all elements of $\boldsymbol{p}^{(j)}$ anyway—i.e., due to the sparsity pattern of $\boldsymbol{A}$—have to be communicated to at least ϕ different nodes during the computation of the product $\boldsymbol{A}\boldsymbol{p}^{(j)}$.

If this is not the case, a possibly considerable number of extra vector elements has to be transferred during the SpMV operation. Since the number of extra elements to be sent is determined by the sparsity pattern of $\boldsymbol{A}$, communication cost can then only be reduced by avoiding extra latencies, i.e., by sending the extra elements to nodes where also other elements have to be sent to. In general, our strategy defined by Eqns. (5) and (6) performs well if $\boldsymbol{A}$ is not *too* sparse within a bandwidth of $\lceil \phi n/(2N) \rceil$ around the diagonal (but can still be very sparse overall). More formally, if for all $i \in \{1, 2, \ldots, N\}$ and $k \in \{1, 2, \ldots, \phi\}$ at least one element in each submatrix $\boldsymbol{A}_{I_{d_{ik}}, I_i}$ is not equal to zero, no cost because of extra latencies is incurred. The corresponding proofs are straightforward, but we are omitting them due to space restrictions.

6 IMPLEMENTATION

Up to this point, we have analyzed the algorithms in theoretical terms. We now describe how the reconstruction is realized in a real-life, finite-precision machine.

We implement Chen's algorithm [11] and our novel extensions as described in Sec. 3 and Sec. 4 using the PETSc framework [3, 4]. PETSc provides the CG solver and linear algebra operations, and it also manages communication between nodes. We use operations offered by the framework to transfer the information required for the recovery from node failures. We use a block Jacobi as a preconditioner during the regular operation of the solver, solving the preconditioner blocks exactly.

To impart fault tolerance, the SpMV is modified to transfer the additional data required to obtain the desired level of data redundancy. In PETSc, the SpMV operation is realized with a generalized scatter: A node determines, from the non-zero entries in its matrix rows, what vector components it requires from its neighbors to perform the SpMV product. With this information, PETSc collectively creates a *communication context* for the generalized scatter operation, defining what entries of a distributed vector are communicated, and where they must be communicated to.

In our experiments, we simulate node failures. Instead of taking down nodes and producing replacements during the reconstruction phase, a node will perform the operations and communication required to restore the solver state. The reconstruction process requires sending the surviving entries of the search direction vector to replacement nodes. In our experiments, this is achieved by reversing the communication that takes place during the matrix-vector product. PETSc already provides this functionality. However, reversing the communication that occurs during the matrix-vector product is not a well-defined operation. To see this, imagine a communication context that dictates that the entry in position i_0 of a vector, located in some node A, must be transferred to positions i_1, in node B, and i_2, in node C. In the reverse communication process, entries in positions i_1 and i_2 will hold candidates for the value of that entry to be transferred to position i_0. In the absence of node-failures, both candidates will be the same value, because they are copies of the original entry in i_0, and the operation is then well defined, but, in the event of a node failure, it is possible that one of these entries is lost and the resulting candidates are different, conflicting values. Therefore, in such cases, the reverse communication process used in the reconstruction could be non-deterministic. We cope with this issue by keeping the search directions in the nodes simulating a node failure. If this information is stored, communication with the reversed context is deterministic, because there would never be a conflict between the candidates.

This problem arises because we use the reverse of a communication context intended for SpMV. In a more optimized implementation, a tailored communication context can be produced after the node failure takes place, which avoids this problem altogether by selecting the entries that we need for the reconstruction.

In our implementation, the linear system arising in line 8 of Alg. 2, is solved using a PCG solver assembled with global operations. In particular, the matrix-vector products of the submatrix $\boldsymbol{A}_{I_f, I_f}$ and the subvector $\boldsymbol{x}^{(j)}_{I_f}$ are performed by multiplying the entire matrix $\boldsymbol{A}$ with a modified vector $\boldsymbol{x}^{(j)}$, whose appropriate entries were set to zero. The desired $\boldsymbol{A}_{I_f, I_f}\boldsymbol{x}^{(j)}_{I_f}$ is a subvector of the result of this global operation. This is less efficient than working with the actual submatrix of $\boldsymbol{A}$, but some of the changes we introduced to increase redundancy conflict with PETSc' ability to create submatrices. The cost to reconstruct the solution, however, remains very small compared to the overall runtime. The CG solver for the subsystem uses a block Jacobi preconditioner, with blocks matching the process' index set. We use an approximate solver based on ILU factorization for the blocks.

Avoiding loss of orthogonality

For this section it is useful to distinguish between the *solver residual*, that is, vector $\boldsymbol{r}^{(j)}$ from Alg. 1, and the vector $\boldsymbol{b} - A\boldsymbol{x}^{(j)}$. These vectors are, in general, not equal in a finite-precision machine.

The CG algorithm in floating-point arithmetic undergoes loss of orthogonality, where roundoff error accumulates and the conjugacy of the search directions is lost as the solver progresses. Consequently, in regular PCG the solver residual and the vector $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ will differ slightly after convergence. Because we work with finite precision, and because we solve the local linear system in the

reconstruction process (line 8 of Alg. 2) iteratively, our algorithm only reconstructs an approximation of the solver state before the node failures take place, thus potentially further contributing to this loss of orthogonality: Consequently, the ESR solver residual after convergence can be larger than the solver residual of PCG. The largest source of deviations is the solution of the local linear system of line 8 of Alg. 2. The loss of orthogonality relative to regular PCG can be controlled with the tolerance of this linear system.

To compare the accuracies of ESR and PCG in this regard, we define the *relative residual difference* metric:

$$\Delta_{\text{ESR}} = \frac{\|\boldsymbol{r}_{\text{ESR}}\|_2 - \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{\text{ESR}}\|_2}{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{\text{ESR}}\|_2} \qquad \Delta_{\text{PCG}} = \frac{\|\boldsymbol{r}_{\text{PCG}}\|_2 - \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{\text{PCG}}\|_2}{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{\text{PCG}}\|_2}. \tag{7}$$

Here, $\boldsymbol{r}_{\text{ESR}}$ and $\boldsymbol{r}_{\text{PCG}}$ are the solver residual vectors of ESR with reconstruction and reference PCG respectively after convergence, and $\boldsymbol{x}_{\text{ESR}}$ and $\boldsymbol{x}_{\text{PCG}}$ are the corresponding iterands.

A side-by-side comparison of the ESR method and regular PCG can show that the former is as accurate as the latter. In Sec. 7.2 we show that the effects of using finite-precision arithmetic can be made negligible. Since node failures are uncommon and reconstruction is relatively cheap to perform, we can set the tolerance for the local system to a very small value, so that ESR converges, while the reconstruction overhead remains low.

7 NUMERICAL EXPERIMENTS

Now we summarize our experimental setup and discuss our results.

7.1 Experimental setup

We use SPD matrices from the SuiteSparse Matrix Collection [12] as test problems, selecting problems from different application areas. Their properties are summarized in Table 1. We select medium (M1, M2) and large (M3-M8) size problems. The latter are among the largest available SPD matrices in the SuiteSparse Matrix Collection. There are problems with different numbers of non-zeros. Large matrices with relatively few non-zeros are common, but problems with more non-zero entries are more expensive to compute and would benefit the most from resilience schemes to protect the resource investment.

Our experiments are run on 128 nodes of the VSC3 system of the Vienna Scientific Cluster. Although our algorithm is well suited for multiple processes per node, we use only one process per node in our experiments. The argument for this decision is twofold: Firstly, from the point of view of resilience, the number of processes per node (cf. Sec. 1.1) makes no difference since the redundant vector elements always have to be stored on a different node (not just in the memory of another process). Secondly, the runtimes obtained in our experiments are based on simulations of node failures without ULFM (cf. Secs. 1.1 and 6). Hence, the relative runtime differences are more significant than the absolute runtimes (and, thus, improvements of the absolute runtimes due to multiple processes per node). Experiments for a matrix are run on the same set of nodes of VSC3. The system's topology is a fat tree. We use the following libraries: • Intel MPI 5.1.3 • PETSc 3.10.4 • Intel MKL 2018.3. We use the Intel C compiler 18.0.5 with compiler flags `-O3`, `-march=native` and `-mtune=native`. We terminate the solver once the relative residual norm has been reduced by a factor of 10^8. The local linear system used during the reconstruction is terminated once its residual norm is reduced by a factor of 10^{14}.

Table 1: SPD matrices from [12] used in the experiments. n: Problem size. NNZ: Number of non-zero entries. The matrices are ordered by increasing number of non-zero entries, with a larger ID indicating a larger number of non-zeros.

Name	Id	Problem type	n	NNZ
parabolic_fem	M1	Fluid dynamics	525 825	3 674 625
offshore	M2	Electromagnetics	259 789	4 242 673
G3_circuit	M3	Circuit simulation	1 585 478	7 660 826
thermal2	M4	Thermal	1 228 045	8 580 313
Emilia_923	M5	Structural	923 136	40 373 538
Geo_1438	M6	Structural	1 437 960	60 236 322
Serena	M7	Structural	1 391 349	64 131 971
audikw_1	M8	Structural	943 695	77 651 847

We measure the runtime of the solver for several problem settings. Node failures are introduced once for each simulation, with either one, three or eight simultaneous node failures taking place at either 20%, 50% or 80% of the solver progress (measured in number of iterations). These failures are placed in contiguous ranks. Simultaneous node failures can well be caused by a faulty switch, therefore it seems like a realistic assumption that they are clustered: The node failures are introduced in neighbouring ranks either at the beginning or in the center of the vector, starting from rank 0 or 64 respectively. Additionally, we have results for runs without failures with either one, three or eight redundant copies. Each measurement in this test constellation is repeated at least 5 times.

7.2 Experimental results

Our experimental results are summarized in Tab. 2. Statistics for node failures are computed over at least 15 values: at least 5 measurements for node failures introduced at either 20%, 50% and 80% of the solver's progress.

As expected, a larger number of redundant copies leads to larger overheads. In general, the reconstruction time remains small, with larger relative reconstruction costs for matrices with smaller runtimes, where the absolute cost is consequently smaller.

The overall relative overhead with node failures, shown in the last three columns of Tab. 2, corresponds to the sum of the relative overhead of the undisturbed case plus the relative runtime cost of the reconstruction. These columns roughly match the sum, as expected, with some variation arising from the variation in runtime of running the code in a real machine, and from differences in the number of iterations caused by the reconstruction (cf. [23]).

Tab. 2 also shows that the location of the node failure does affect the reconstruction cost, as the runtime is, in general, different for node failures at the start (lower indexes) or the center (middle indexes) of the vectors. These differences come from different diagonal linear systems in the reconstruction: Submatrices formed from the index sets of different failed nodes have different properties, and

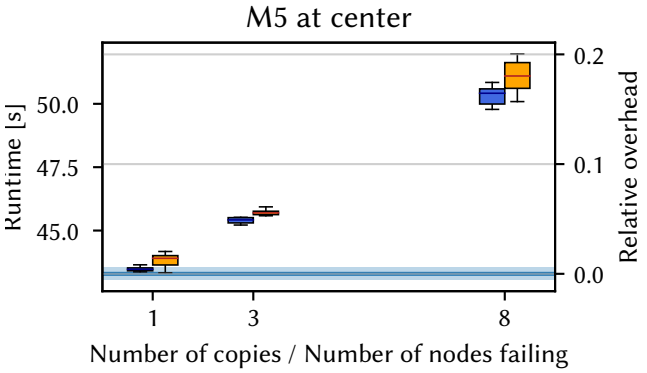

Figure 1: Runtimes and relative overhead for the matrix M5 of Tab. 1, for node failures introduced close to the center of the vector. The blue line at the bottom of the figure represents the reference time obtained solving the system using PETSc without modifications, with respect to which the relative overhead is measured, and the band around it extends for one standard deviation in each direction. The x-axis indicates the number of copies the resilient solver holds. Boxes include points in the interquartile range, and whiskers extend up to 1.5 times the width of the interquartile range. Blue boxes (to the left of a group) represent runs with the resilient solver without node failures. Orange boxes (to the right of a group) represent runs with node failures. In experiments with node failures, we introduce as many simultaneous failures as the solver can tolerate (one, three or eight failures respectively.) Orange boxes include experiments for failures introduced at 20%, 50% or 80% progress of the solver.

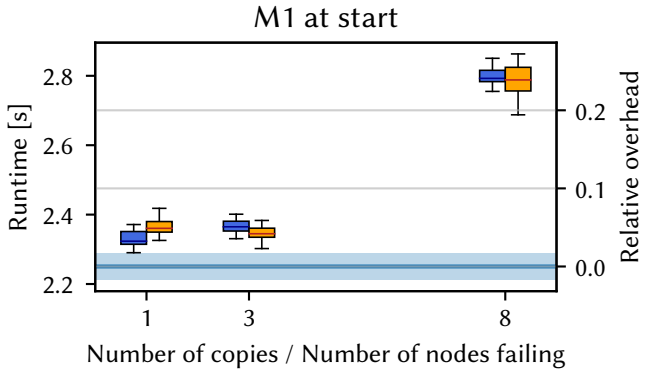

Figure 2: Runtimes and relative overheads for matrix M1 with node failures occurring close to the start (lower indices) of the vectors. This figure uses the same conventions as Fig. 1, and showcases a situation, for three redundant copies, where the solver converges faster after performing reconstruction after a node failure due to the reduction of the number of iterations until convergence.

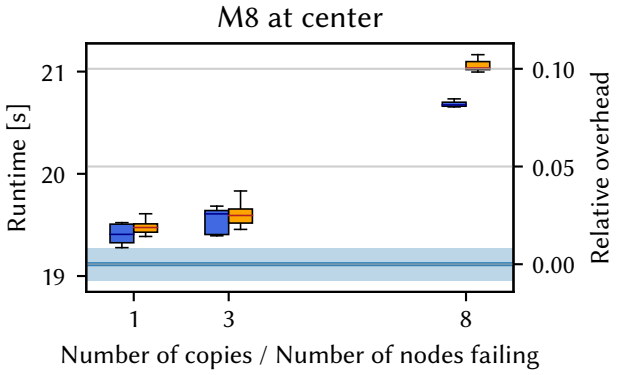

Figure 3: Runtimes and relative overheads for matrix M8 with node failures occurring close to the start of the vectors. This figure uses the same conventions as Fig. 1, and shows superlinear increase of the overhead with respect to the number of redundant copies held.

they will not converge at the same rate with an ILU preconditioner, thus affecting the reconstruction time.

Our experiments show that our algorithm is very efficient for matrices with many non-zeros contained in a band close to the diagonal. Relatively denser matrices also take longer to reach convergence because of the longer time required for the matrix-vector product, so it makes a lot of sense to protect the time investment in the solution process with a resilience technique like the one presented in this paper.

Fig. 1 is an example of the runtimes and overheads obtained with our novel resilient algorithm for the matrix M5. For this matrix, the state reconstruction operation takes very little time: The runtimes for cases with failures, (orange boxes) are very close to the failure-free cases (blue boxes). In this case, the overhead for the method comes predominantly from the additional communication required to maintain redundant data.

In Fig. 2 the boxes corresponding to three redundant copies indicate a *smaller* runtime for simulations with node failures than for the failure-free case. As mentioned before, this can happen, since the number of iterations required after reconstruction can be a smaller than in the failure-free run (cf. [23]).

Fig. 3 shows a test case where the overhead required to keep redundant data increases superlinearly with the number of node failures tolerated, As explained in Sec. 5, the growth of the overhead with the number of node failures tolerated strongly depends on the sparsity pattern of $\boldsymbol{A}$. The matrix M8 contains many non-zeros in a band around the diagonal in the middle indexes. As expected from the analysis of Sec. 5, this is a particularly favorable case for our method: It can resist three simultaneous node failures with an overhead of around 2.5%, and eight node failures with an overhead of around 10%.

In general, the iteration at which the node failures are introduced has little influence on the runtime of the solver. Fig. 4 illustrates this for M5. We observed the same behaviour for the other test cases.

In Sec. 6, we discuss the potential loss of orthogonality that occurs when performing the reconstruction. Tab. 3 shows that the

Table 2: Summarized experimental results. Values are aggregated for experiments with different times (iteration numbers) when the failures are introduced. Matrices are ordered by descending reference runtime, with the ID number of a matrix growing with increasing number of non-zeros. t_0: average reference time for the reference runs with regular (non-fault tolerant) PCG. ϕ: number of redundant copies for the search direction. ψ: number of simultaneous node failures introduced. The column "relative overhead undisturbed" shows the mean overhead of modified ESR-capable PCG, with a given number of redundant copies ϕ, with respect to the reference time t_0 if no reconstruction takes place. The columns marked "relative reconstruction time" indicate the time that it takes to reconstruct the state of the solver, expressed as a percentage of the reference time t_0, plus/minus a standard deviation. The columns marked "relative overhead with failures" show the mean, plus/minus a standard deviation, of the overall relative overhead, that is, the relative overhead for the total time until convergence, when node failures occur and the state of the solver is reconstructed, with respect to the reference runtime t_0.

ID	t_0 [s]	Relative overhead undisturbed [%]			Failure location	Relative reconstruction time [%]			Overhead with failures [%]		
		$\phi = 1$	$\phi = 3$	$\phi = 8$		$\psi = \phi = 1$	$\psi = \phi = 3$	$\psi = \phi = 8$	$\psi = \phi = 1$	$\psi = \phi = 3$	$\psi = \phi = 8$
M5	43.29	0.4	5.1	16.3	start	0.2 ± 0.1	0.3 ± 0.1	0.6 ± 0.1	1.2 ± 0.5	6.0 ± 0.3	19.6 ± 0.7
					center	0.0 ± 0.1	0.1 ± 0.1	0.4 ± 0.1	1.2 ± 0.6	5.6 ± 0.3	20.5 ± 6.0
M8	19.12	1.5	2.2	8.2	start	1.4 ± 0.1	3.5 ± 0.2	10.7 ± 0.2	3.8 ± 1.0	5.6 ± 0.7	19.9 ± 0.9
					center	0.4 ± 0.1	0.8 ± 0.1	1.4 ± 0.1	1.9 ± 0.4	2.8 ± 1.0	10.4 ± 1.1
M6	11.70	0.3	3.1	15.1	start	1.3 ± 0.1	2.4 ± 0.2	5.3 ± 0.2	1.4 ± 0.3	5.9 ± 0.4	21.2 ± 1.2
					center	0.3 ± 0.1	0.6 ± 0.1	1.6 ± 0.1	0.7 ± 0.4	4.0 ± 0.3	18.3 ± 0.7
M7	6.48	0.2	4.3	13.6	start	2.6 ± 0.1	8.2 ± 0.3	23.4 ± 0.6	2.9 ± 0.3	13.7 ± 0.6	39.6 ± 1.4
					center	1.2 ± 0.1	1.5 ± 0.1	21.1 ± 0.7	1.2 ± 0.2	7.1 ± 0.6	36.7 ± 1.7
M4	6.31	8.2	22.8	65.6	start	2.0 ± 0.1	3.1 ± 0.1	4.9 ± 0.3	9.6 ± 0.8	26.4 ± 1.1	66.0 ± 7.9
					center	1.3 ± 0.1	3.8 ± 0.2	5.2 ± 0.4	9.2 ± 2.6	35.7 ± 2.2	63.4 ± 9.3
M1	2.25	3.6	5.1	24.5	start	0.3 ± 0.1	0.3 ± 0.1	0.4 ± 0.1	5.1 ± 1.2	4.3 ± 0.9	23.8 ± 2.1
					center	0.3 ± 0.1	0.3 ± 0.1	0.4 ± 0.1	5.0 ± 1.0	5.1 ± 1.3	25.4 ± 1.8
M2	1.58	8.0	8.1	21.1	start	4.0 ± 0.2	7.0 ± 0.3	9.7 ± 0.4	7.3 ± 4.6	15.6 ± 2.3	30.1 ± 5.5
					center	2.1 ± 0.3	3.5 ± 0.2	4.4 ± 0.3	10.0 ± 4.6	14.2 ± 2.6	23.8 ± 3.0
M3	1.16	5.0	24.1	91.3	start	0.6 ± 0.1	1.0 ± 0.1	1.9 ± 0.1	6.3 ± 1.7	24.8 ± 1.8	93.6 ± 6.0
					center	8.0 ± 0.5	32.2 ± 1.2	58.1 ± 1.4	14.8 ± 2.0	55.0 ± 3.2	147.6 ± 5.0

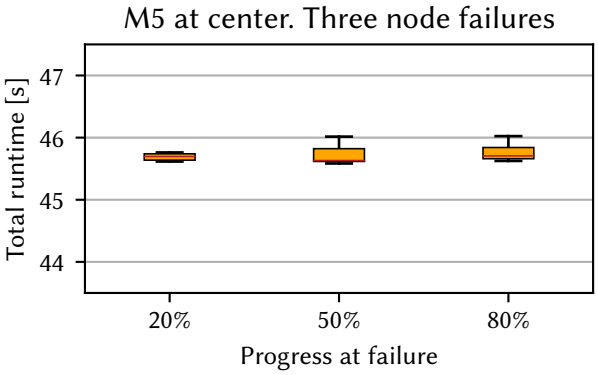

Figure 4: Runtime for matrix M5 when introducing the failure of three nodes at the center (middle indexes) of the vectors and at different iterations along the progress of the solver. The boxes contain runtimes in the interquartile range, and the whiskers extend up to 1.5 times the width of the interquartile range.

relative residual difference for our method is comparable to the one for the reference results, even for its maximum value among all experiments for a given matrix. The deviations for both methods are tiny in comparison to the reduction of the residual norm by a factor of 10^8 from the solver.

8 CONCLUSIONS

In this paper, we first reviewed the ESR approach, which was initially proposed by Chen [11] and later refined by Pachajoa et al. [23], an efficient fault-tolerance technique to protect the PCG method against a single node failure (cf. Secs. 2.2 and 3). We then proposed an enhancement to the ESR approach that allows the PCG method to tolerate simultaneous or overlapping failures of *multiple* nodes (cf. Sec. 4.1). Our new strategy determines where to efficiently store redundant information in order to support up to $\phi < N$ simultaneous node failures. In a theoretical analysis, we found that the communication overhead due to the distribution of the required additional redundant vector copies strongly depends on the sparsity pattern of the given system matrix (cf. Secs. 4.2 and 5).

In order to investigate the effects of floating-point arithmetic and the runtime performance of our novel algorithm, we implemented

Table 3: Evaluation of the metric of Eqn. (7). The first column shows the maximum value of the relative residual deviation for all experiments with node failures for a matrix. The second column shows the relative residual deviation for the reference run.

ID	max Δ_{ESR}	Δ_{PCG}
M1	3.46×10^{-7}	-1.63×10^{-7}
M2	2.24×10^{-7}	1.86×10^{-7}
M3	1.57×10^{-7}	1.57×10^{-7}
M4	1.96×10^{-7}	9.06×10^{-8}
M5	3.59×10^{-5}	1.81×10^{-6}
M6	8.80×10^{-8}	-3.31×10^{-8}
M7	1.32×10^{-7}	-7.24×10^{-8}
M8	2.64×10^{-3}	1.49×10^{-3}

it based on the widely used library PETSc (cf. Sec. 6) and conducted numerical experiments on 128 nodes of the Vienna Scientific Cluster (cf. Sec. 7). The results of our experiments with eight large sparse matrices from real-world applications show that the proposed enhanced ESR approach is very efficient. Compared to a non-resilient PCG run, we measured runtime overheads between 2.2% and 24.1% for an undisturbed run with up to three tolerated simultaneous node failures and between 2.8% and 55.0% for a run with three actual simultaneous node failures including reconstruction.

In future work, we are going to further enhance the ESR approach such that it automatically adapts to different sparsity patterns of matrices. Moreover, we want to investigate communication-avoiding PCG methods. Another interesting future direction is to find a variant of the ESR approach that does not depend on the availability of replacement nodes for failed nodes.

ACKNOWLEDGMENTS

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-113. The computational results presented have been achieved using the Vienna Scientific Cluster (VSC).

REFERENCES

[1] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. 2013. *Towards resilient parallel linear Krylov solvers: recover-restart strategies.* Research Report RR-8324. INRIA.

[2] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. 2016. Numerical recovery strategies for parallel resilient Krylov linear solvers. *Numer. Lin. Algebra. Appl.* 23, 5 (2016), 888–905.

[3] 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. Curfman McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. 2018. *PETSc Users Manual.* Technical Report ANL-95/11 - Revision 3.9. Argonne National Laboratory.

[4] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. 1997. *Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries.* Birkhäuser Boston, 163–202.

[5] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. 2014. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numer.* 23 (2014), 1–155.

[6] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. 2013. Post-failure recovery of MPI communication capability: Design and rationale. *Int. J. High Perform. Comput. Appl.* 27, 3 (2013), 244–254.

[7] G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra. 2014. Assessing the Impact of ABFT and Checkpoint Composite Strategies. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops.* 679–688.

[8] G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra. 2015. Composing resilience techniques: ABFT, periodic and incremental checkpointing. *International Journal of Networking and Computing* 5, 1 (2015), 2–25.

[9] G. Bronevetsky and B. R. de Supinski. 2008. Soft Error Vulnerability of Iterative Linear Algebra Methods. In *Proceedings of the 22nd Annual International Conference on Supercomputing.* ACM, 155–164.

[10] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn. 2007. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 19, 13 (2007), 1749–1783.

[11] Z. Chen. 2011. Algorithm-based Recovery for Iterative Methods Without Checkpointing. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing.* ACM, 73–84.

[12] T. A. Davis and Y. Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software* 38, 1 (2011), 1:1–1:25.

[13] K. Dichev and D. S. Nikolopoulos. 2016. TwinPCG: Dual Thread Redundancy with Forward Recovery for Preconditioned Conjugate Gradient Methods. In *2016 IEEE International Conference on Cluster Computing (CLUSTER).* 506–514.

[14] E. N. Elnozahy, L. Alvisi, Y. Wang, and D. B. Johnson. 2002. A Survey of Rollback-recovery Protocols in Message-passing Systems. *ACM Comput. Surv.* 34, 3 (2002), 375–408.

[15] M. Fasi, J. Langou, Y. Robert, and B. Uçar. 2016. A backward/forward recovery approach for the preconditioned conjugate gradient method. *J. Comput. Sci.* (2016), 522–534.

[16] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari. 2017. 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 (SC '17).* ACM, 44:1–44:12.

[17] T. Herault and Y. Robert (Eds.). 2015. *Fault-Tolerance Techniques for High-Performance Computing.* Springer International Publishing.

[18] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. 2007. Recovery Patterns for Iterative Methods in a Parallel Unstable Environment. *SIAM J. Sci. Comput.* 30, 1 (2007), 102–116.

[19] H. Ltaief, E. Gabriel, and M. Garbey. 2008. Fault tolerant algorithms for heat transfer problems. *J. Parallel Distrib. Comput.* 68, 5 (2008), 663 – 677.

[20] Message Passing Interface Forum. 2015. MPI: A Message-Passing Interface Standard. https://www.mpi-forum.org/docs/.

[21] Message Passing Interface Forum. 2017. User Level Failure Mitigation. http://fault-tolerance.org/.

[22] C. Pachajoa and W. N. Gansterer. 2018. On the Resilience of Conjugate Gradient and Multigrid Methods to Node Failures. In *Euro-Par 2017: Parallel Processing Workshops.* Springer, 569–580.

[23] C. Pachajoa, M. Levonyak, and W. N. Gansterer. 2018. Extending and Evaluating Fault-Tolerant Preconditioned Conjugate Gradient Methods. In *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS).* 49–58.

[24] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2^{nd} ed.). SIAM.

[25] P. Sao and R. Vuduc. 2013. Self-stabilizing Iterative Solvers. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '13).* ACM, 4:1–4:8.

[26] R. D. Schlichting and F. B. Schneider. 1983. Fail-stop Processors: An Approach to Designing Fault-tolerant Computing Systems. *ACM Trans. Comput. Syst.* 1, 3 (1983), 222–238.

[27] B. Schroeder and G. Gibson. 2010. A Large-Scale Study of Failures in High-Performance Computing Systems. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 337–350.

[28] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. 2012. Fault Tolerant Preconditioned Conjugate Gradient for Sparse Linear System Solution. In *Proceedings of the 26th ACM International Conference on Supercomputing.* ACM, 69–78.

[29] 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. Van Hensbergen. 2014. Addressing failures in exascale computing. *Int. J. High Perform. Comput. Appl.* 28, 2 (2014), 129–173.

[30] D. Tiwari, S. Gupta, and S. S. Vazhkudai. 2014. Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* 25–36.