A Generic Strategy for Node-Failure Resilience for Certain Iterative Linear Algebra Methods

Carlos Pachajoa, Robert Ernstbrunner and Wilfried N. Gansterer
University of Vienna
Faculty of Computer Science
Vienna, Austria
{carlos.pachajoa,robert.ernstbrunner,wilfried.gansterer}@univie.ac.at

Abstract

Resilience is an important research topic in HPC. As computer clusters go to extreme scales, work in this area is necessary to keep these machines reliable.

In this work, we introduce a generic method to endow iterative algorithms in linear algebra based on sparse matrix-vector products, such as linear system solvers, eigensolvers and similar, with resilience to node failures. This generic method traverses the dependency graph of the variables of the iterative algorithm. If the iterative method exhibits certain properties, it is possible to produce an exact state reconstruction (ESR) algorithm, enabling the recovery of the state of the iterative method in the event of a node failure. This reconstruction is exact, except for small perturbations caused by floating point arithmetic. The generic method exploits redundancy in the matrix-vector product to protect the vector that is the argument of the product.

We illustrate the use of this generic approach on three iterative methods: the conjugate gradient method, the BiCGStab method and the Lanczos algorithm. The resulting ESR algorithms enable the reconstruction of their state after a node failure from a few redundantly stored vectors.

Unlike previous work in preconditioned conjugate gradient, this generic method produces ESR algorithms that work with general matrices. Consequently, we can no longer assume that local diagonal submatrices used to reconstruct vectors are non-singular. Thus, we also propose an approach for deriving non-singular local linear systems for the reconstruction process with reduced condition numbers, based on a communication-avoiding rank-revealing QR factorization with column pivoting.

Source and copyright notice

This work was published in the 2020 IEEE/ACM 10th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS).
978-0-7381-1080-6/20/$31.00 ©2020 IEEE.
DOI 10.1109FTXS51974.2020.00010 (link)

1 Introduction

To cover growing computational demands, computers increase in parallelism. Assuming that the reliability of the individual components does not scale to match (drastic improvements in reliability are unlikely with the push for smaller features in integrated-circuit fabrication), computer clusters will suffer faults more often at extreme scales. In this scenario, it is justified to start thinking of computer clusters as unreliable machines [14], where the possibility of failures has to be considered during algorithm and software development. In this work, we consider *node failures* which are events where a computational node in a computer cluster stops working and causes the data that was stored in the node to be lost.

Numerical linear algebra is central to computational science and engineering. Problems defined by sparse matrices—that is, by matrices that contain few non-zero entries—appear in many areas, like in the numerical solution of partial differential equations, or the solution of problems in graph theory. The sparsity property can be exploited for reducing memory and computational cost of a matrix-vector product, and thus favors iterative methods.

Currently, the most used strategy in practice to deal with node failures is *checkpoint-restart* (CR), where we periodically store the state of the application and, in the event of a node failure, revert to the last stored state. However, it has been pointed out that CR does not scale well [2, 3], and a checkpointing approach may not perform well in very-large scale systems. Tackling the problem in a different direction, *algorithm-based fault tolerance* (ABFT) exploits properties of the algorithm to provide fault tolerance [7]. Some recent work combines generic CR with specific properties of the preconditioned conjugate gradient (PCG) method, providing an instance of *algorithm-specific checkpointing* [13].

In this work, we develop strategies to provide numerical iterative linear algebra algorithms with resilience against node failures, by generating an exact state reconstruction (ESR) algorithm. The iterative algorithm is made to operate with an augmented version of the matrix-vector product (see Section 2). The ESR approach, as described in [4, 9–11, 13] holds redundant copies of vectors involved in matrix-vector products in the algorithm with very low memory and runtime overheads in the absence of node failures. After recovery from a node failure, its variables have the same values as in the undisturbed case except for small perturbations due to floating-point arithmetic.

1.1 Terminology and notation

We now introduce some terms that are used in this paper.

A node that stop working in the event of a node failure is referred to as a *lost node*. The information contained in it becomes inaccessible and can no longer be used by the iterative algorithm. A *spare node* is a node, supplied by the cluster, that remains in standby, and takes the place of a lost node in the event of a node failure. A spare node that replaces a lost node after a node failure is called a *replacement node*. A node that continues working and whose information remains accessible after a node failure is a *surviving node*. A node that performs the state reconstruction, whether it is a spare node or a surviving node, is called a *reconstruction node*.

To designate subsets for indices of vector entries or matrix rows, we use the notation in [4]. The set of all indices is referred to as I. A subindex will restrict this index set to the indicated node, or denote a subset of indices. For example, if F is the lost node, the set of indices corresponding to the lost elements is denoted as I_F, and we can refer to the set of indices for elements in surviving nodes as $I \setminus I_F$. We use index sets as subscripts to refer to entries of vectors and matrices, for example, if we have a distributed vector $\boldsymbol{x}$, we refer to the surviving vector elements as $\boldsymbol{x}_{I \setminus I_F}$, and for a matrix $\boldsymbol{A}$, the rows corresponding to the lost node can be written as $\boldsymbol{A}_{I_F,I}$. We say that a node *owns* an index, and thus the corresponding vector entry and matrix row, if the index is in the set assigned to it. In the context of iterative methods, a *superscript* in parentheses (as in $\boldsymbol{x}^{(j)}$) indicates the iteration number of a vector or scalar.

The *state* of an iterative algorithm refers to all of its dynamic data, that is, the vectors and scalars whose values change in every iteration. It does not include the static data which defines the problem, such as the system matrix, the preconditioner and the right-hand-side vector. The *trajectory* of the iterative algorithm is the sequence of states that it goes through until convergence. A given state fully defines the trajectory subsequently followed by an iterative algorithm.

At points, we use the following terms to distinguish between three different categories of algorithms. The *iterative algorithm* is the algorithm performing the task we want to complete, such as a linear solver, and which we want to protect in the event of a node failure. The *exact state reconstruction algorithm* (ESR) is created for a specific iterative algorithm, and its task is to reconstruct the state of the latter after a node failure. It is run by a reconstruction node. The *meta-algorithm* refers to the procedure used to create a new ESR algorithm for a given iterative algorithm.

We use the term *algorithmic pattern* to refer to the arrangement of operations executed by an iterative algorithm, which defines how the ESR algorithm performs the reconstruction. In the context of numerical linear algebra, these operations are matrix-vector products, vector additions, linear system solutions and the like.

1.2 Problem statement

In this paper, we work on resilience for iterative methods in linear algebra, which satisfy the conditions presented in Section 3.3, in a distributed-memory setting. The algorithms we work with involve matrix-vector products, and the matrices and vectors involved are held in the memory of a computer cluster in a block-row distribution.

We concern ourselves with situations where the computer cluster can suffer node failures, i.e., events in which its nodes stop working and the information regarding the iterative algorithm that is contained in them becomes inaccessible, such as blocks of vector entries, matrix rows, and the scalars residing in the nodes' memory.

We look for a generic way to derive exact state reconstruction (ESR) strategies for iterative linear algebra algorithms. These strategies reconstruct the information lost during a node failure in a reconstruction node, and, in exact arithmetic, recover the state of the iterative algorithm exactly. We assume that the static data that defines the problem is securely stored and can be retrieved during the reconstruction.

In this work, we assume the availability of a spare node to act as a replacement node, and that only one node can fail at a time. However, generalizing the methods described in this paper to deal with the simultaneous failure of multiple nodes [11] and to work without spare nodes [12] is possible.

1.3 Previous work

In [8], Langou et al. deal with the recovery of the iterand after a node failure for any iterative linear solver with block-row-distributed matrices and vectors. They reconstruct the iterand by interpolating the lost entries from the surviving entries and the right-hand-side vector. The residual norm after the reconstruction is within a constant factor from the residual norm before the node failure. Agullo et al. [1] improve on the work in [8] by performing least-squares minimization instead of performing linear interpolation to reconstruct the lost entries. As a result, the residual norm after the reconstruction is less than or equal to the residual norm before the failure. This method works on the same solvers as the method in [8].

In [4], Chen introduces a way to use the sparse matrix-vector product to redundantly store the input vector, as well as a way to reconstruct the entirety of the state of a preconditioned conjugate gradient (PCG) solver using the last two redundantly stored search directions and a scalar that was replicated in the cluster. This work was extended further in [11], where Pachajoa et al. consider the case of simultaneous failure of multiple nodes. Pachajoa et al. [12] describe how to make this method work without the use of spare nodes in the cluster, redistributing the data to continue working on the surviving nodes only. In [9], Levonyak et al. adapt the exact state reconstruction method to also work with pipelined PCG (PPCG), a variant of PCG that hides communication and is better suited for very large-scale systems. The resulting ESR algorithm preserves PPCG's desirable scalability properties. In [13], Pachajoa et al. improve the ESR method for PCG by reducing the frequency at which the state of the solver is stored, developing the approach explicitly into a form of algorithm-specific checkpoint-restart.

1.4 Contributions of this paper

The main contribution of this paper is a generic procedure for the derivation of exact state reconstruction strategies for iterative linear algebra methods satisfying the conditions presented in Section 3.3. Our approach is based on the dependency graph of the iterative algorithm, and moves backwards to recover the lost entries of its vectors.

We also propose ways to efficiently solve for variables in frequently occurring algorithmic patterns in these iterative methods.

In the recovery process after a node failure it is often necessary to solve a local linear system, using a diagonal submatrix of the system matrix, to recover the missing entries of a given vector. Some of the iterative algorithms that we consider in this work can deal with general, non-symmetric matrices, and it is therefore possible that the submatrices required during the reconstruction are rank deficient. Based on a communication-avoiding rank-revealing QR decomposition, we provide a strategy for finding an alternative submatrix that can then be used to find the missing entries.

This paper is structured as follows: Section 2 explains in more detail how the matrix-vector product is used to provide redundancy for the input vector. In Section 3, we introduce the generic approach for deriving ESR reconstruction strategies for iterative linear algebra algorithms. In Section 4, we present case studies for applying our approach to the CG method, the BiCGStab method and the Lanczos algorithm. In Section 5, we explain how to deal with frequently appearing algorithmic patterns, and also present an experiment for a strategy to handle poorly conditioned or singular diagonal submatrices in the recovery process. Finally, we conclude in Section 6.

2 Algorithmic background

The starting point for the algorithms in this work is the inherent resilience provided by the sparse matrix-vector product (SpMV) [4].

To compute the SpMV of a block-row-distributed matrix and a vector, the entries of the input vector have to be communicated from their owner node n to some other node n' where, following the sparsity pattern of the matrix, they are necessary to find the partial matrix-vector product for the indices owned by node n'. We denote with $S_{n,n'}$ the set of indices for entries sent from node n to node n' during the computation of the SpMV. This already provides some redundancy: At least the entries in $S_{n,n'}$ will exist in another node after the product. We define $R_n \coloneqq \bigcup_{n' \neq n} S_{n,n'}$, that is, the set of indices of all entries owned by node n that have to be sent to some other node during the matrix-vector product. SpMV provides inherent redundancy for the entries in R_n.

In order to provide redundancy for the entirety of the input vector, the SpMV operation has to be augmented, in an operation we call ASpMV. We have to identify the entries owned by node n that are not being sent to any other node, whose indices form the set $R_n^c \coloneqq I_n \setminus R_n$. To guarantee the complete redundancy of the input vector, each node n must send the entries in the set R_n^c to some other node, in addition to the entries required to complete the matrix vector product. After the node failure of a given node n, all of its entries will be available somewhere in the cluster, and it is possible to collect them in a reconstruction node. Thus, ASpMV provides redundancy for the input vector.

Descriptions of the algorithms for our test cases are introduced in their corresponding subsection, in Sec. 4.

3 Generic derivation of ESR

In this section, we present our generalization of the ESR method, applied to iterative methods where an SpMV operation is used.

3.1 Drawing a dependency graph

The value of a variable of an iterative algorithm depends on previous values of other variables. Thus, it is possible to create a dependency graph for the iterative algorithm, extending all the way back to its initialization. Nodes in this graph represent both a variable and the operation used to compute it.

While finding the path backwards in the graph, we can assume that all input vectors in the SpMV are known. If the iterative algorithm operates with a finite-term recurrence, it is only necessary to redundantly store the last few

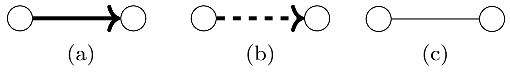

Figure 1: Edge annotations for the construction of the graph. (a): Dependency can be reverted. (b): Dependency cannot be reverted. (c): Reversible relation not explicit in the algorithm (between the iterand and the residual, for example).

of them. For example, the conjugate gradient method (see Sec. 4.1) is a two-term recurrence, and we must only redundantly store the last two search directions.

Let us consider a graph node m whose corresponding variable is used as an argument in the computation of the variable of node n. In this case, the edge will be directed from m to n. We identify the following annotations for the edge:

- The operation is reversible: If n and all nodes with an edge pointing to n are known, outside of m, it is possible to compute m.
- The operation is not reversible: The node m cannot be computed, even if n and all other nodes pointing to n are known. For example, it is not possible to determine the entries of a vector from its norm.
- A relationship between variables that is not explicit in the graph. This is usually a residual relation.

These annotations are displayed in Figure 1.

3.2 Derivation of an ESR algorithm

To produce an ESR algorithm that can be used in a reconstruction node, we follow these steps:

1. Identify all relevant variables, considering vectors and matrices as single variables.
2. Identify what variables form the state of the solver. We need enough information to be able to continue with the solution process as an undisturbed solver would.
3. Identify, in each line of the algorithm, which variables can be reconstructed if all other variables were present.
4. Construct the graph from the algorithm. Add relations that are not explicit in the algorithm, such as between the iterand and the residual of a linear solver.
5. From the last vector in the graph that is involved in a SpMV with the system matrix, start traversing the graph in breadth-first order:
   (a) Node n corresponds to an assignment in the algorithm. This assignment defines an equation. For each variable m in the equation (outside of the one corresponding to n), which corresponds to a node with an edge pointing to n, determine if it is possible to find it with the variables known already. See Section 5 for some frequently occurring algorithmic patterns in these assignments.
   (b) If the variable can be computed, mark it as known and add it to the list for the graph traversal.
6. Stop once the complete state of the solver can be reconstructed.

At this point, we have a path that enables us to reconstruct the state of the solver, such that it follows the same trajectory as an undisturbed solver. The equations obtained while solving for the variables in this workflow are the individual steps in the ESR algorithm. This reconstruction is performed in a single reconstruction node (since we are assuming that only a single node can fail at a time).

The generic meta-algorithm, used to produce ESR algorithms, is abridged in Alg. 1.

3.3 Scope of the method

A ESR algorithm can be derived for an iterative algorithm in linear algebra if the latter fulfills the following conditions:

- The iterative algorithm performs a finite-term recurrence. This enables us to reconstruct the state from a bounded amount of data.
- The iterative algorithm involves a matrix-vector product, such that redundant copies the input vector can be produced with low memory and runtime overhead.

A complete characterization of the class of iterative algorithms for which the meta-algorithm is applicable is topic of future work.

Algorithm 1 State reconstruction meta-algorithm from $\boldsymbol{w}^{(j)}$ as the last input vector to the ASpMV

```
1: $Q := \{\boldsymbol{w}^{(j)}\}$, Nodes for vectors involved in the ASpMV are marked as known.
2: repeat
3:     $n := \texttt{pop}(Q)$
4:     for all nodes $m$ with incoming connections to n do
5:         if $m$ is not marked as known and
6:            $m$ can be computed from $n$ then
7:             Mark $m$ as known.
8:             Push $m$ into $Q$.
9:         end if
10:    end for
11: until All nodes for a complete state are marked as known
```

3.4 Performance of the resulting ESR method

Since ASpMV transfers more information than the amount required to compute the matrix-vector product, its use entails an overhead compared to regular SpMV. This overhead depends on various aspects such as the sparsity pattern of the matrix, the topology of the computer cluster, and the selection of which nodes receive and redundantly store a given entry. Sending these additional entries, however, does not require additional MPI messages, or new collective-communication rounds. It can be performed with the MPI operation `Alltoallw`, the same as the regular SpMV. Thus, the significant cost of starting a new communication round is avoided.

In [13], it is shown how to apply the ASpMV with a reduced frequency (only every T iterations) and thus reduce the runtime overhead caused by holding redundant copies to basically zero, at the price of having to revert up to T iterations in the event of a node failure. This is analogous to a checkpoint-restart method with a checkpointing interval T. During failure-free operation, our methods transfer less than the information communicated when using an in-memory checkpoint-restart approach, which transfers the entirety of the local part of the state of the iterative method. However, the state reconstruction of our method, which takes place in the event of a node failure, is more expensive than the recovery stage of checkpoint-restart, which must only retrieve backed-up data. Most of the overhead for the reconstruction in ESR is caused by the solution of a local linear system. We expect to reduce this overhead with the technique presented in Sections 5.1.1 and 5.2, which is based on CARRQR.

4 Case studies

4.1 Case study: The conjugate gradient method

In this example, we show how the ESR algorithm for the conjugate gradient method (CG) is produced. For simplicity in the explanations, we only present the non-preconditioned CG method, although this strategy is also applicable to preconditioned CG. CG solves a linear system of the form $\boldsymbol{Ax} = \boldsymbol{b}$, where the matrix $\boldsymbol{A}$ is symmetric, positive definite (SPD).

The CG algorithm is presented in Alg. 2. In this algorithm, we can identify all the vectors ($\boldsymbol{x}$, $\boldsymbol{r}$, $\boldsymbol{p}$ and $\boldsymbol{Ap}$) and scalars ($\boldsymbol{r}^\intercal\boldsymbol{r}$, α and β) that we would declare in the program for the solver. This, of course, is not the only way to select these variables.

We can draw a subgraph for each of the lines of Alg. 2. We present two examples, in Fig. 2 and Fig. 3 for Lines 3 and 7, respectively, of Alg. 2. It is not necessary to consider all the vectors of the program simultaneously. For example, $\boldsymbol{Ap}$ can be found from $\boldsymbol{p}$, so we do not place the former in our graph. We will not consider the scalars in our graphs either, for reasons explained in Section 5.3. However, if we formed the graph with nodes for all variables, we would obtain the same ESR method.

Algorithm 2 Conjugate gradient (CG) method [15, Alg. 6.18]

```
1: $\boldsymbol{x}^{(0)}$ arbitrary, $\boldsymbol{r}^{(0)} := \boldsymbol{b} - \boldsymbol{Ax}^{(0)}, \boldsymbol{p}^{(0)} := \boldsymbol{r}^{(0)}$
2: for $j = 0, 1, \ldots,$ until convergence do
3:     $\alpha^{(j)} := \boldsymbol{r}^{(j)\intercal}\boldsymbol{r}^{(j)} / \boldsymbol{p}^{(j)\intercal}\boldsymbol{Ap}^{(j)}$
4:     $\boldsymbol{x}^{(j+1)} := \boldsymbol{x}^{(j)} + \alpha^{(j)}\boldsymbol{p}^{(j)}$
5:     $\boldsymbol{r}^{(j+1)} := \boldsymbol{r}^{(j)} - \alpha^{(j)}\boldsymbol{Ap}^{(j)}$
6:     $\beta^{(j)} := \boldsymbol{r}^{(j+1)\intercal}\boldsymbol{r}^{(j+1)} / \boldsymbol{r}^{(j)\intercal}\boldsymbol{r}^{(j)}$
7:     $\boldsymbol{p}^{(j+1)} := \boldsymbol{r}^{(j+1)} + \beta^{(j)}\boldsymbol{p}^{(j)}$
8: end for
```

Observing the previously presented conventions, and following the operations specified in the algorithm, a dependency graph for the solver is drawn. Here, it is presented in Fig. 4, and the steps required to reconstruct the state are

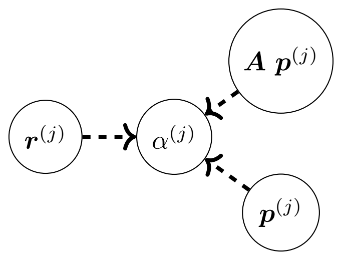

Figure 2: Subgraph for the formula $\alpha^{(j)} := \boldsymbol{r}^{(j)\intercal}\boldsymbol{r}^{(j)}/\boldsymbol{p}^{(j)\intercal}\boldsymbol{A}\boldsymbol{p}^{(j)}$. Arrows go towards the $\alpha^{(j)}$ node because that is the variable being computed. Dashed lines are used to indicate what variables cannot be computed if all of the others are known.

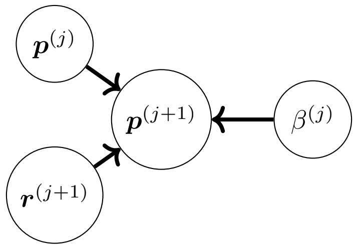

Figure 3: Subgraph for the formula $\boldsymbol{p}^{(j+1)} := \boldsymbol{r}^{(j+1)} + \beta^{(j)}\boldsymbol{p}^{(j)}$. Arrows go towards the $\boldsymbol{p}^{(j+1)}$ node because that is the variable being computed. Bold lines are used to indicate what variables can be computed if all of the others are known.

described in the figure's caption. The resulting ESR algorithm for CG is presented in Alg. 3, including the addition of the necessary steps for retrieving the required static data and entries from the surviving nodes. We do not consider the scalars nor the vector $\boldsymbol{Ap}$ in this graph. After we have found a way to recover the state, we can determine more carefully how to use the scalars. It turns out that, from the scalars, we only require $\beta^{(j-1)}$ to be able to reconstruct $\boldsymbol{r}^{(j)}$ from the equation of Line 7 in Alg. 2.

Algorithm 3 ESR reconstruction phase for the CG method on a replacement node.

```
1: Retrieve the static data $\boldsymbol{A}_{I_F,I}$, 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 $\boldsymbol{r}^{(j)}_{I_F} := \boldsymbol{p}^{(j)}_{I_F} - \beta^{(j-1)}\boldsymbol{p}^{(j-1)}_{I_F}$
5: Compute $\boldsymbol{w} := \boldsymbol{b}_{I_F} - \boldsymbol{r}^{(j)}_{I_F} - \boldsymbol{A}_{I_F,I\setminus I_F}\boldsymbol{x}^{(j)}_{I\setminus I_F}$
6: Solve $\boldsymbol{A}_{I_F,I_F}\boldsymbol{x}^{(j)}_{I_F} = \boldsymbol{w}$ for $\boldsymbol{x}^{(j)}_{I_F}$
```

4.2 Case study: The BiCGStab method

We generate an ESR reconstruction algorithm for the BiCGStab solver. This is a Krylov subspace method to solve the linear system, $\boldsymbol{Ax} = \boldsymbol{b}$, where, in contrast to CG, there are no restrictions on $\boldsymbol{A}$ beyond not being singular. BiCGStab is shown in Alg. 4.

From the listing we can produce a dependency graph, presented in Fig. 5. The steps followed to reconstruct the state are presented in the caption of this figure. Notice that the solver does not require the vector $\boldsymbol{s}^{(j)}$ to continue in the same trajectory as an undisturbed solver. In the graph of Fig. 5, it is sufficient to reconstruct $\boldsymbol{p}^{(j+1)}$, $\boldsymbol{p}^{(j)}$, $\boldsymbol{r}^{(j+1)}$ and $\boldsymbol{x}^{(j+1)}$ to compute a new $\boldsymbol{s}^{(j+1)}$. From this, we can write down the ESR reconstruction algorithm for BiCGStab, presented in Alg. 5.

4.3 Case study: The Lanczos algorithm

The Lanczos algorithm is used to find the k most dominant eigenvalues and eigenvectors of a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{|I|\times|I|}$. The algorithm finds a sequence of scalars that form a symmetric tridiagonal matrix $\boldsymbol{T}_k \in \mathbb{R}^{k\times k}$:

$$\boldsymbol{T}_k = \begin{pmatrix} \alpha^{(1)} & \beta^{(2)} & & & \\ \beta^{(2)} & \alpha^{(2)} & \beta^{(3)} & & \\ & & \ddots & & \\ & & \beta^{(k-1)} & \alpha^{(k-1)} & \beta^{(k)} \\ & & & \beta^{(k)} & \alpha^{(k)} \end{pmatrix}, \tag{1}$$

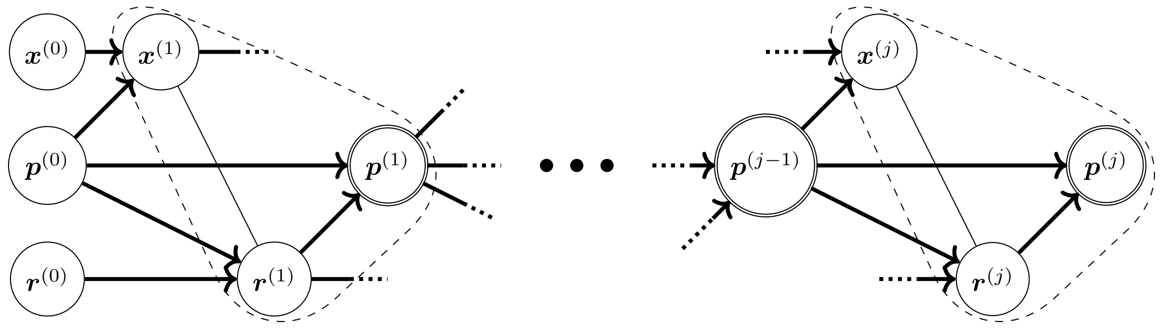

Figure 4: Dependency graph for the CG method. 1. The traversal starts in the last vector involved in the SpMV. Here, it is $\boldsymbol{p}^{(j)}$. 2. Starting the breadth-first traversal with $\boldsymbol{r}^{(j)}$, we check if $\boldsymbol{r}^{(j)}$ can be computed. Since we assume that $\boldsymbol{p}^{(j-1)}$ is also known, $\boldsymbol{r}^{(j)}$ can be computed. There are no further unknown nodes connected to $\boldsymbol{p}^{(j)}$. 3. The traversal continues from $\boldsymbol{r}^{(j)}$. From this node, both $\boldsymbol{x}^{(j)}$, through the residual relation, and $\boldsymbol{r}^{(j-1)}$ can be found. At this point, we have the complete state of the solver, surrounded by a dashed line: If we know $\boldsymbol{x}^{(j)}$, $\boldsymbol{r}^{(j)}$ and $\boldsymbol{p}^{(j)}$, the solver can continue in the same trajectory as before the node failure. $\boldsymbol{p}^{(j-1)}$ is still required to complete the reconstruction.

Algorithm 4 BiCGStab algorithm [15, Alg. 7.7]

```
1: $\boldsymbol{x}^{(0)}$ arbitrary, $\boldsymbol{r}^{(0)} := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$, $\boldsymbol{r}^{(0)*}$ arbitrary, $\boldsymbol{p}^{(0)} := \boldsymbol{r}^{(0)}$
2: for $j = 0, 1, \dots,$ until convergence do
3:     $\alpha^{(j)} := \boldsymbol{r}^{(j)\intercal}\boldsymbol{r}^{(0)*} / \boldsymbol{r}^{(0)*\intercal}\boldsymbol{A}\boldsymbol{p}^{(j)}$
4:     $\boldsymbol{s}^{(j)} := \boldsymbol{r}^{(j)} - \alpha^{(j)}\boldsymbol{A}\boldsymbol{p}^{(j)}$
5:     $\omega^{(j)} := \boldsymbol{s}^{(j)\intercal}\boldsymbol{A}\boldsymbol{s}^{(j)} / (\boldsymbol{A}\boldsymbol{s}^{(j)})^\intercal(\boldsymbol{A}\boldsymbol{s}^{(j)})$
6:     $\boldsymbol{x}^{(j+1)} := \boldsymbol{x}^{(j)} + \alpha^{(j)}\boldsymbol{p}^{(j)} + \omega^{(j)}\boldsymbol{s}^{(j)}$
7:     $\boldsymbol{r}^{(j+1)} := \boldsymbol{s}^{(j)} - \omega^{(j)}\boldsymbol{A}\boldsymbol{s}^{(j)}$
8:     $\beta^{(j)} := \frac{\alpha^{(j)}}{\omega^{(j)}}\, \boldsymbol{r}^{(j+1)\intercal}\boldsymbol{r}^{(0)*} / \boldsymbol{r}^{(j)\intercal}\boldsymbol{r}^{(0)*}$
9:     $\boldsymbol{p}^{(j+1)} := \boldsymbol{r}^{(j+1)} + \beta^{(j)}(\boldsymbol{p}^{(j)} - \omega^{(j)}\boldsymbol{A}\boldsymbol{p}^{(j)})$
10: end for
```

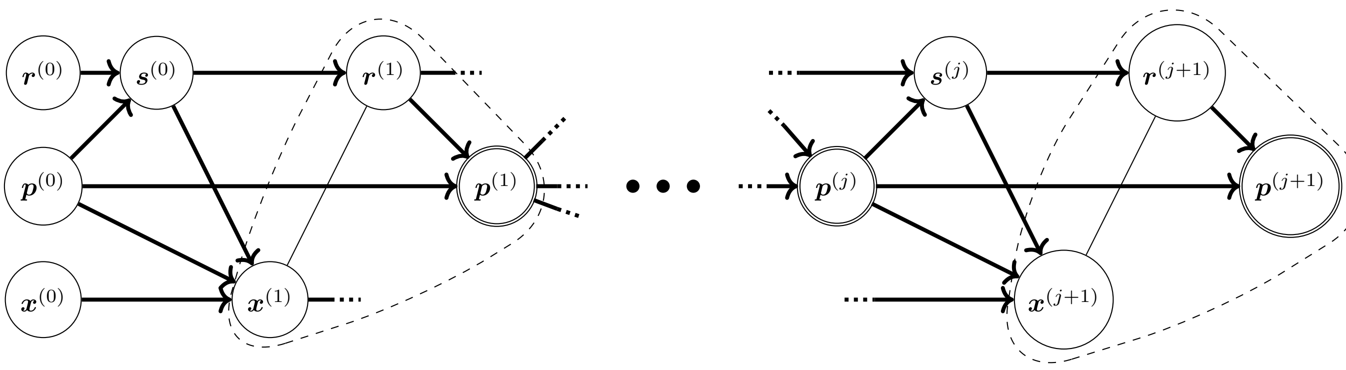

Figure 5: Dependency graph for the BiCGStab method. 1. The traversal starts in the last vector involved in the SpMV: $\boldsymbol{p}^{(j+1)}$. 2. Starting the breadth-first traversal, we check if $\boldsymbol{r}^{(j+1)}$ can be computed. Since $\boldsymbol{p}^{(j)}$ is assumed to be known, this is the case. 3. Continuing the traversal, we find that $\boldsymbol{s}^{(j)}$ and $\boldsymbol{x}^{(j+1)}$ can be computed, the latter by using the residual relation. At this point, the state has been reconstructed and the solver can continue in the same trajectory as the undisturbed solver.

Algorithm 5 ESR reconstruction phase for the BiCGStab method on a replacement node.

```
1: Retrieve the static data $\boldsymbol{A}_{I_F,I}$, $\boldsymbol{r}^{(0)*}_{I_F}$ and $\boldsymbol{b}_{I_F}$
2: Gather $\boldsymbol{r}^{(j)}_{I\setminus I_F}$, $\boldsymbol{x}^{(j)}_{I\setminus I_F}$ and $\boldsymbol{p}^{(j-1)}_{I\setminus I_F}$
3: Retrieve the redundant copies of $\beta^{(j-1)}$, $\omega^{(j-1)}$, $\boldsymbol{p}^{(j-1)}_{I_F}$, and $\boldsymbol{p}^{(j)}_{I_F}$
4: Compute $\boldsymbol{\varrho}^{(j-1)}_{I_F} := \boldsymbol{A}_{I_F,I}\boldsymbol{p}^{(j-1)}$
5: Compute $\boldsymbol{r}^{(j)}_{I_F} := \boldsymbol{p}^{(j)}_{I_F} - \beta^{(j-1)}(\boldsymbol{p}^{(j-1)}_{I_F} - \omega^{(j-1)}\boldsymbol{\varrho}^{(j-1)}_{I_F})$
6: Compute $\boldsymbol{w} := \boldsymbol{b}_{I_F} - \boldsymbol{r}^{(j)}_{I_F} - \boldsymbol{A}_{I_F,I\setminus I_F}\boldsymbol{x}^{(j)}_{I\setminus I_F}$
7: Solve $\boldsymbol{A}_{I_F,I_F}\boldsymbol{x}^{(j)}_{I_F} = \boldsymbol{w}$ for $\boldsymbol{x}^{(j)}_{I_F}$
```

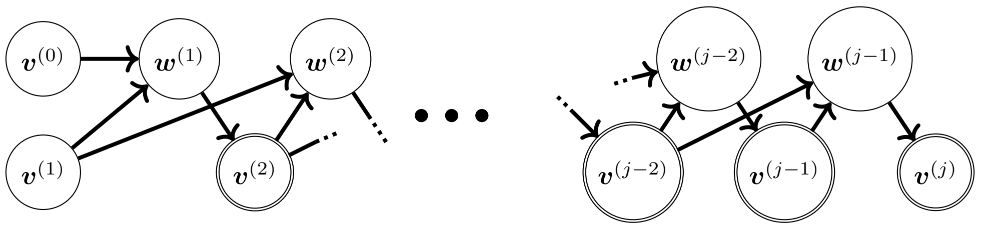

Figure 6: Dependency graph for the Lanczos method. 1. The traversal starts in the last vector involved in the SpMV. Here, it is $\boldsymbol{v}^{(j)}$. 2. Starting the breadth-first traversal with $\boldsymbol{w}^{(j-1)}$, Since we assume that $\boldsymbol{v}^{(j)}$ is known, $\boldsymbol{w}^{(j-1)}$ can be computed. If we also redundantly store $\boldsymbol{v}^{(j-1)}$, we have the complete state of the solver and can continue iterating in the same trajectory as the unaffected solver.

as well as the column vectors for an orthogonal matrix $\boldsymbol{V}_k$, such that $\boldsymbol{T}_k = \boldsymbol{V}_k^\intercal \boldsymbol{A}\boldsymbol{V}_k$. The eigenvalues of $\boldsymbol{T}_k$ approximate the dominating eigenvalues of $\boldsymbol{A}$, and if $\boldsymbol{y}$ is an eigenvector of $\boldsymbol{T}_k$, then $\boldsymbol{V}_k\boldsymbol{y}$ approximates an eigenvector of $\boldsymbol{A}$.

The Lanczos algorithm is presented in Alg. 6. From it, we build the dependency graph, ignoring the scalars and intermediate vectors ($\boldsymbol{A}\boldsymbol{v}^{(j)}$). The graph is presented in Fig. 6, and the steps followed to perform the reconstruction are presented in its caption.

Algorithm 6 Lanczos algorithm [15, Alg. 6.15]

```
1: $\boldsymbol{v}^{(1)}$ arbitrary, with $\|\boldsymbol{v}^{(1)}\|_2 = 1$, $\beta^{(1)} := 0$, $\boldsymbol{v}^{(0)} = \boldsymbol{0}$
2: for $j = 1, 2, \ldots, k$ do
3:     $\boldsymbol{w}^{(j)} := \boldsymbol{A}\boldsymbol{v}^{(j)} - \beta^{(j)}\boldsymbol{v}^{(j-1)}$
4:     $\alpha^{(j)} := \boldsymbol{w}^{(j)\intercal}\boldsymbol{v}^{(j)}$
5:     $\boldsymbol{w}^{(j)} := \boldsymbol{w}^{(j)} - \alpha^{(j)}\boldsymbol{v}^{(j)}$
6:     $\beta^{(j+1)} := \|\boldsymbol{w}^{(j)}\|_2$. If $\beta^{(j+1)} = 0$, stop.
7:     $\boldsymbol{v}^{(j+1)} := \boldsymbol{w}^{(j)}/\beta^{(j+1)}$
8: end for
```

The scalars α and β must be replicated in all nodes in order to perform the operations in Lines 3, 5 and 7 of Alg. 6. We focus on the case where $k \ll |I|$, where $|I|$ is the number of entries of the vectors and therefore also the size of the problem. Consequently, we can assume that the number of scalars remains small, and it is reasonable to maintain copies of all of them in every node.

The resulting ESR algorithm is presented in Alg. 7. After a node failure, the algorithm can recover the last $\boldsymbol{w}$ vector from the last $\boldsymbol{v}$ vector. It can then continue computing the remaining values of $\boldsymbol{T}_k$ if we retrieve the $\boldsymbol{v}$ vectors from the last two iterations. In the end, the matrix $\boldsymbol{T}_k$ is found and it can be used to approximate the eigenvalues of $\boldsymbol{A}$.

Although the Lanczos algorithm is a two-step recurrence, all of the $\boldsymbol{v}$ vectors are required for eigenvalue computations. After a node failure, the rows I_F of the matrix $\boldsymbol{V}_k$ are lost. Therefore, recovering from such a failure in a way that enables eigenvalue computations also requires provision of redundancy for all $\boldsymbol{v}$ vectors, and not only the last two. This problem is beyond the scope of this paper.

5 Handling algorithmic patterns during derivation of an ESR algorithm

When building an ESR algorithm for an iterative algorithm, certain operations are common. In this section, we explain how to deal with them.

Algorithm 7 ESR reconstruction phase for the Lanczos algorithm on a replacement node.

```
1: Retrieve the static data: $\boldsymbol{A}_{I_F,I}$
2: Gather $\boldsymbol{w}^{(j-1)}_{I\setminus I_F}$
3: Retrieve the redundant copies of $\beta^{(j)}$, $\boldsymbol{v}^{(j-1)}_{I_F}$, and $\boldsymbol{v}^{(j)}_{I_F}$
4: Compute $\boldsymbol{w}^{(j-1)}_{I_F} := \beta^{(j)}\boldsymbol{v}^{(j)}_{I_F}$
```

5.1 Reconstructing vectors involved in matrix-vector products

Some formulas in an algorithm have the form $\boldsymbol{z} := \boldsymbol{w} + \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{z}$ and $\boldsymbol{w}$ are known vectors, $\boldsymbol{A}$ is a matrix, and we try to reconstruct the vector $\boldsymbol{x}$. This is equivalent to solving the linear system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{y} := \boldsymbol{z} - \boldsymbol{w}$. As an instance of this case, in the ESR algorithm for CG, we use the residual relation $\boldsymbol{r}^{(j)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(j)}$ for the reconstruction of the iterand (Alg. 3, Line 6). After a node failure, the surviving parts of $\boldsymbol{x}$ will still be available, so it is not necessary to solve the full linear system. Instead, we can solve only for the lost entries. If I represents the ordered set of all indices of the system, and I_F is the ordered set of the indices of all entries lost after a node failure event, we can introduce an ordered set I_r, with $|I_r| = |I_F|$, and then we write this matrix-vector product as

$$\begin{pmatrix} \boldsymbol{y}_{I\setminus I_r} \\ \boldsymbol{y}_{I_r} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{I\setminus I_r, I\setminus I_F} & \boldsymbol{A}_{I\setminus I_r, I_F} \\ \boldsymbol{A}_{I_r, I\setminus I_F} & \boldsymbol{A}_{I_r, I_F} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_{I\setminus I_F} \\ \boldsymbol{x}_{I_F} \end{pmatrix}.$$

Thus, we can write an equation for the lost entries of $\boldsymbol{x}$:

$$\boldsymbol{A}_{I_r,I_F}\boldsymbol{x}_{I_F} = \boldsymbol{y}_{I_r} - \boldsymbol{A}_{I_r,I\setminus I_F}\boldsymbol{x}_{I\setminus I_F}. \tag{2}$$

5.1.1 Selection of I_r

For a SPD matrix $\boldsymbol{A}$, we can pick $I_r = I_F$ because the submatrix $\boldsymbol{A}_{I_F,I_F}$ is non-singular. This is shown in Corollary 1. This has the advantage of being local: It does not require matrix information from other nodes to define the submatrix.

Corollary 1. *If $\boldsymbol{A}$ is a symmetric, positive definite (SPD) matrix then, for any choice of index set I_j, the submatrix $\boldsymbol{A}_{I_j,I_j}$ is not singular.*

Proof. There exists a permutation matrix $\boldsymbol{P}$ that reorders the entries of a vector such that the entries corresponding to I_j are moved to the first $|I_j|$ positions.

Because $\boldsymbol{A}$ is an SPD matrix, for any vector $\boldsymbol{x} \neq 0$ we have $\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} > 0$. We define a matrix $\boldsymbol{A}' = \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\intercal$. Because of our selection of $\boldsymbol{P}$, the submatrix $\boldsymbol{A}_{I_j,I_j}$ is in the upper-left corner of $\boldsymbol{A}'$.

We define the vector $\boldsymbol{y} = \boldsymbol{P}\boldsymbol{x}$, and notice that $\boldsymbol{y}$ is non-zero if $\boldsymbol{x}$ is non-zero. Then we have $\boldsymbol{y}^\intercal\boldsymbol{A}'\boldsymbol{y} = \boldsymbol{x}^\intercal\boldsymbol{P}^\intercal\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\intercal\boldsymbol{P}\boldsymbol{x} = \boldsymbol{x}^\intercal\boldsymbol{A}\boldsymbol{x}$. Therefore, the matrix $\boldsymbol{A}'$ is SPD.

From Sylvester's criterion, it follows that the determinant of the submatrix $\boldsymbol{A}_{I_j,I_j}$ must be positive, and the matrix $\boldsymbol{A}_{I_j,I_j}$ must be non-singular.

□

For more general matrices, it might happen that $\boldsymbol{A}_{I_F,I_F}$ is non-invertible and the linear system of Eq. 2 cannot be solved. In this event, we need to pick I_r such that the submatrix is non-singular. According to Corollary 2, enough information exists in the matrix to solve the system of Eq. 2. If $I_r \neq I_F$, this requires row information from other nodes to form the subsystem, and thus requires communication.

Corollary 2. *Given an invertible matrix $\boldsymbol{A}$ and an index set I_F, there exists an index set I_r with $|I_r| = |I_F|$ such that the matrix $\boldsymbol{A}_{I_r,I_F}$ is invertible.*

Proof. Because $\boldsymbol{A}$ is invertible, all of its columns are linearly independent. Therefore, the dimension of the column space of $\boldsymbol{A}_{I,I_F}$ is $|I_F|$. By the fundamental theorem of linear algebra, the dimension of the row space of $\boldsymbol{A}_{I,I_F}$ must also be $|I_F|$, thus, there are $|I_F|$ linearly independent rows in $\boldsymbol{A}_{I,I_F}$. If the index set I_r represents this selection of rows, then the square matrix $\boldsymbol{A}_{I_r,I_F}$ has $|I_F|$ linearly independent rows, and is therefore invertible. □

The selection of I_r intends to form a matrix of the *most linearly independent* row vectors of $\boldsymbol{A}_{I,I_F}$, that is, a selection that minimizes the condition number of the resulting matrix $\boldsymbol{A}_{I_r,I_F}$. We propose to perform this selection using the *communication-avoiding rank-revealing QR factorization* (CARRQR) [6]. CARRQR minimizes communication by employing a technique called tournament pivoting. In order to identify the $|I_F|$ most linearly independent rows of $\boldsymbol{A}_{I,I_F}$, the algorithm partitions $\boldsymbol{A}^T_{I,I_F}$ into blocks of $2|I_F|$ columns. Tournament pivoting then proceeds in two steps. The first step applies a rank revealing QR factorization, e.g., QR factorization with column pivoting (QRCP), to each block to reveal and permute the $|I_F|$ most linearly independent columns to the left. The second step involves a reduction operation that combines these $|I_F|$ leftmost columns with the $|I_F|$ leftmost columns of its neighbor to form new blocks of $2|I_F|$ columns. These steps are repeated until, after a final rank revealing QR factorization, there are only $|I_F|$ columns left. The algorithm produces a permutation matrix $\boldsymbol{P}_r$ such that the most linearly independent rows are located in the upper square submatrix of $\boldsymbol{P}^T_r\boldsymbol{A}_{I,I_F}$.

Table 1: Test matrices from [5]

Matrix	Problem type	Problem size
sherman5	CFD	3312
west1505	Chemical simulation	1505

In a distributed memory environment, traditional rank revealing algorithms like, e.g., QRCP are sub-optimal with respect to the number of messages exchanged. CARRQR minimizes communication at the cost of worse bounds for a quantity involved in the characterization of a rank revealing factorization. In [6], it is shown that these bounds are usually very pessimistic, and that the algorithm is competitive in practice. CARRQR performs three times more flops than QRCP but, on a system where communication is the bottleneck, the former is expected to be significantly faster. We consider CARRQR as a first approximation to the solution of this problem, and illustrate its use to improve the condition number of a matrix subblock with experiments in Sec. 5.2.

5.1.2 Completing the linear system with rows from other nodes

It is also possible to first perform a singular value decomposition (SVD) in the replacement node, and then obtain a minimal number of rows from other nodes. Also, because of Corollary 2, there must exist a subset of indices I_c, with $I_c \subset I$ and $I_c \cap I_F = \emptyset$, that provides the additional information to solve the system. Of course, we want our selection of I_c to minimize some objective, such as the amount of communication required, which depends on the nodes that own the selected rows.

A required step would be to determine if the submatrix $\boldsymbol{A}_{I_F,I_F}$ is singular. Thus, we perform a rank-revealing QR factorization. If the subsystem has full rank, we can proceed with the solution. Otherwise, the system will have a rank $k < |I_F|$. We perform a SVD decomposition:

$$\boldsymbol{A}_{I_F,I_F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\intercal, \tag{3}$$

where $\hat{\boldsymbol{\Sigma}} \in \mathbb{R}^{k\times k}$ is the diagonal matrix of the non-zero singular values of $\boldsymbol{\Sigma}$ and $\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}} \in \mathbb{R}^{|I_F|\times k}$ are the matrices of the corresponding left and right singular vectors.

We can define another submatrix for the index set $I_G := I \setminus I_c \setminus I_F$ and decompose the corresponding linear system as in Eq. 4:

$$\begin{pmatrix} \boldsymbol{y}_{I_G} \\ \boldsymbol{y}_{I_c} \\ \boldsymbol{y}_{I_F} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{I_G,I_G} & \boldsymbol{A}_{I_G,I_c} & \boldsymbol{A}_{I_G,I_F} \\ \boldsymbol{A}_{I_c,I_G} & \boldsymbol{A}_{I_c,I_c} & \boldsymbol{A}_{I_c,I_F} \\ \boldsymbol{A}_{I_F,I_G} & \boldsymbol{A}_{I_F,I_c} & \boldsymbol{A}_{I_F,I_F} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_{I_G} \\ \boldsymbol{x}_{I_c} \\ \boldsymbol{x}_{I_F} \end{pmatrix}. \tag{4}$$

From this, we can write one more equation:

$$\boldsymbol{A}_{I_c,I_F}\boldsymbol{x}_{I_F} = \boldsymbol{y}_{I_c} - \boldsymbol{A}_{I_c,I_G}\boldsymbol{x}_{I_G} - \boldsymbol{A}_{I_c,I_c}\boldsymbol{x}_{I_c} \tag{5}$$

Eq. 5 provides $|I_c|$ additional restrictions on $\boldsymbol{x}_{I_c}$ that can be used to correct the rank deficiency of $\boldsymbol{A}_{I_F,I_F}$.

We define a vector $\boldsymbol{w}_{I_F} := \boldsymbol{y}_{I_F} - \boldsymbol{A}_{I_F,I\setminus I_F}\boldsymbol{x}_{I\setminus I_F}$, set $I_r := I_F$ and write, from Eq. 2, $\boldsymbol{A}_{I_F,I_F}\boldsymbol{x}_{I_F} = \boldsymbol{w}_{I_F}$. Then we can write

$$\hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\intercal\boldsymbol{x}_{I_F} = \boldsymbol{w}_{I_F} \to \hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\intercal\boldsymbol{x}_{I_F} = \hat{\boldsymbol{U}}^\intercal\boldsymbol{w}_{I_F}.$$

We introduce the vector $\boldsymbol{v}_{I_c} := \boldsymbol{y}_{I_c} - \boldsymbol{A}_{I_c,I_G}\boldsymbol{x}_{I_G} - \boldsymbol{A}_{I_c,I_c}\boldsymbol{x}_{I_c}$ and rewrite Eq. 5 as $\boldsymbol{A}_{I_c,I_F}\boldsymbol{x}_{I_F} = \boldsymbol{v}_{I_c}$. Then, provided that the rows of $\boldsymbol{A}_{I_c,I_F}$ are not in the span of the vectors of $\hat{\boldsymbol{V}}$, we can instead solve the system

$$\begin{pmatrix} \hat{\boldsymbol{U}}^\intercal\boldsymbol{w}_{I_F} \\ \boldsymbol{v}_{I_c} \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\intercal \\ \boldsymbol{A}_{I_c,I_F} \end{pmatrix} \boldsymbol{x}_{I_F} \tag{6}$$

and obtain the lost entries we are looking for.

The selection of columns from the surviving nodes, I_c, remains future work.

5.2 Experiments with I_r selection using tournament pivoting

We perform numerical experiments with small matrices to demonstrate how an appropriate selection of matrix rows, I_r, improves the conditioning of the subsystem if it is poorly conditioned, or creates a non-singular one if the corresponding submatrix is rank-deficient. As mentioned in Sec. 5.1.1, we employ CARRQR [6] for the selection of rows.

We work with two square matrices from the SuiteSparse Matrix Collection [5], presented in Table 1. We simulate eight nodes by uniformly dividing the rows of the matrix in eight contiguous blocks. We then examine the resulting diagonal submatrices. For the matrix `sherman5`, we pick I_F to correspond to the most poorly conditioned diagonal submatrix. For the matrix `west1505`, we pick I_F to correspond to the second submatrix, which does not have non-zero entries and is thus singular.

The resulting condition numbers are shown in Table 2.

Table 2: Condition number for submatrices

Matrix	**cond($\boldsymbol{A}_{I_F,I_F}$)**	**cond($\boldsymbol{A}_{I_r,I_F}$)**
sherman5	4540.0	469.2
west1505	∞	6.08×10^8

For the matrix `sherman5`, the reduced condition number leads to fewer iterations if we solve the subsystem with an iterative solver. For the matrix `west1505`, the subsystem is no longer singular. We can find a solution and, therefore, perform the reconstruction.

5.3 Storage and recovery of scalars

Holding scalars in memory does not represent a large overhead in comparison to holding vectors, therefore, we propose to replicate them in all nodes and to maintain the necessary copies in all nodes. Often, these scalars are computed with `AllReduce` operations (as in Line 3 in Alg. 2), making these values available in all surviving nodes in the event of a node failure.

5.4 Recomputation of matrix-vector products

If the reconstruction of a vector involves a matrix-vector product (as in Line 4 of Alg. 5) it is possible to recompute only the entries in I_F in the reconstruction node $\boldsymbol{A}_{I_F,I}\boldsymbol{x}$ instead of the full product $\boldsymbol{Ax}$. The latter would have to be performed by all nodes, while the former would result in a smaller overhead during the reconstruction.

6 Conclusions

In this paper, we propose a generic method to produce exact state reconstruction (ESR) strategies for iterative linear algebra methods, endowing them with resilience against node failures. Our approach exploits the dependency graph of an iterative algorithm to enable the reconstruction of its state from a few vectors. The method also exploits the inherent redundancy in the matrix-vector product in linear algebra algorithms formulated as a finite-term recurrence. The generic method traverses the dependency graph searching for a path to reconstruct the state of the iterative method. This path defines the resulting ESR algorithm.

We present case studies for three representative algorithms: the CG method, the BiCGStab method and the Lanczos algorithm. In the event of a node failure, the derived ESR algorithms can recover the state of the iterative algorithms, such that the linear solvers can continue iterating to convergence, like an undisturbed solver, and the eigenvalue solver can continue finding the desired eigenvalues, although computation of the corresponding eigenvectors requires further redundancy.

We describe some frequently found algorithmic patterns that appear in operations of the iterative algorithms, and present suggestions on how to efficiently solve for the desired variables. A particularly interesting problem is the solution of local linear systems during the reconstruction. Since the local diagonal submatrix can be singular or ill-conditioned, we present a technique, based on CARRQR, that produces a well-conditioned matrix from which the solution can be found. We also provide examples of its application on two matrices from real-world problems.

The complete characterization of the method in terms of the conditions that an iterative algorithm must fulfill so that an ESR algorithm can be derived from it, as well as the implementation and evaluation of the resulting resilient methods, remain future work.

Acknowledgement

This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-113.

References

[1] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. "Numerical recovery strategies for parallel resilient Krylov linear solvers". In: *Numer. Lin. Algebra. Appl.* 23.5 (2016), pp. 888–905.

[2] Jiajun Cao, Kapil Arya, Rohan Garg, Shawn Matott, Dhabaleswar K. Panda, Hari Subramoni, Jérôme Vienne, and Gene Cooperman. "System-Level Scalable Checkpoint-Restart for Petascale Computing". In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 2016, pp. 932–941.

[3] Franck Cappello, Geist Al, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. "Toward exascale resilience: 2014 update". In: *Supercomput. Front. Innov.* 1.1 (2014), pp. 4–27.

[4] Zizhong Chen. “Algorithm-based Recovery for Iterative Methods Without Checkpointing”. In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing.* 2011, pp. 73–84.

[5] Timothy A. Davis and Yifan Hu. “The University of Florida Sparse Matrix Collection”. In: *ACM Trans. Math. Software* 38.1 (2011), 1:1–1:25.

[6] James W. Demmel, Laura Grigori, Ming Gu, and Hua Xiang. “Communication Avoiding Rank Revealing QR Factorization with Column Pivoting”. In: *SIAM Journal on Matrix Analysis and Applications* 36.1 (2015), pp. 55–89.

[7] Kuang-Hua Huang and Jacob A. Abraham. “Algorithm-Based Fault Tolerance for Matrix Operations”. In: *IEEE Trans. Comput.* C-33.6 (1984), pp. 518–528.

[8] Julien Langou, Zizhong Chen, George Bosilca, and Jack Dongarra. “Recovery Patterns for Iterative Methods in a Parallel Unstable Environment”. In: *SIAM J. Sci. Comput.* 30.1 (2007), pp. 102–116.

[9] Markus Levonyak, Christina Pacher, and Wilfried N. Gansterer. “Scalable Resilience Against Node Failures for Communication-Hiding Preconditioned Conjugate Gradient and Conjugate Residual Methods”. In: *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing.* 2020, pp. 81–92.

[10] Carlos Pachajoa, Markus Levonyak, and Wilfried N. Gansterer. “Extending and Evaluating Fault-Tolerant Preconditioned Conjugate Gradient Methods”. In: *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS).* 2018, pp. 49–58.

[11] Carlos Pachajoa, Markus Levonyak, Wilfried N. Gansterer, and Jesper Larsson Träff. “How to Make the Preconditioned Conjugate Gradient Method Resilient Against Multiple Node Failures”. In: *48th International Conference on Parallel Processing - ICPP.* 2019, 67:1–67:10.

[12] Carlos Pachajoa, Christina Pacher, and Wilfried N. Gansterer. “Node-failure-resistant preconditioned conjugate gradient method without replacement nodes”. In: *2019 IEEE/ACM 9th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS).* 2019, pp. 31–40.

[13] Carlos Pachajoa, Christina Pacher, Markus Levonyak, and Wilfried N. Gansterer. “Algorithm-Based Checkpoint-Recovery For The Conjugate Gradient Method”. In: *49th International Conference on Parallel Processing - ICPP.* 2020, 14:1–14:11.

[14] Daniel A. Reed and Jack Dongarra. “Exascale Computing and Big Data”. In: *Commun. ACM* 58.7 (2015), pp. 56–68.

[15] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* 2nd Ed. 2003.