SIAM J. SCI. COMPUT.
Vol. 39, No. 1, pp. C48–C64

ERASURE CODING FOR FAULT-OBLIVIOUS LINEAR SYSTEM SOLVERS*

YAO ZHU†, DAVID F. GLEICH†, AND ANANTH GRAMA†

Abstract. Dealing with faults is an important problem as parallel and distributed systems scale to millions of processing cores. Traditional methods for dealing with faults include checkpoint-restart, active replicas, and deterministic replay. Each of these techniques has associated resource overheads and constraints. In this paper, we propose an alternate approach to dealing with faults based on input augmentation. This approach, which is an algorithmic analog of erasure-coded storage, applies a minimally modified algorithm on the augmented input to produce an augmented output. The execution of such an algorithm proceeds completely oblivious to faults in the system. In the event of one or more faults, the real solution is recovered using a rapid reconstruction method from the augmented output. We demonstrate this approach on the problem of solving sparse linear systems using a conjugate gradient solver, where we present input augmentation and output recovery techniques. Through simulations, we show that our approach can be made oblivious to a large number of faults with low computational overhead. Specifically, we demonstrate cases where a single fault can be corrected with less than 10% overhead in time, and even in extreme cases (fault rates of 20%), our approach is able to compute a solution with reasonable overhead.

Key words. erasure coding, linear system solvers, fault oblivious computation, algorithm based fault tolerance

AMS subject classifications. 68M15, 65F10

DOI. 10.1137/15M1041511

1. Introduction.

The next generation of parallel and distributed systems are projected to scale to millions of processing cores and beyond. In this regime, hardware and software faults present major challenges for scalable execution of programs. We consider solving n $n \times n$ nonsingular linear system

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \tag{1.1}$$

in a computing environment with faults.

Our contribution in this paper is the design of a new type of method for solving (1.1), which we call an erasure-coded algorithm. At the core of our concept is an algorithm that allows us to recover the correct solution to the problem even if some of the computational units involved in the solution process fail during the execution of the algorithm. We use an abstraction of a computational unit failing wherein specific components of a vector become *fixed* or *stuck* and can no longer be changed. This emulates the behavior of a distributed system where a node dropped out of the computation. The outcome of our effort is a fault-oblivious solution procedure. More specifically, given system (1.1), we design an augmented system:

$$\tilde{\boldsymbol{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{1.2}$$

*Submitted to the journal's Software and High-Performance Computing section September 28, 2015; accepted for publication (in revised form) September 12, 2016; published electronically February 22, 2017.

http://www.siam.org/journals/sisc/39-1/M104151.html

Funding: This work was supported in part by DOE award DE-SC0014543 and ARO Award 1111691-CNS.

†Department of Computer Science, Purdue University, West Lafayette, IN 47907 (yaozhu@purdue.edu, dgleich@purdue.edu, ayg@purdue.edu).

together with a solution strategy, such that using the solution strategy to solve (1.2) in an environment with faults would have the following properties:

1. *Deterministic finite termination.* The solution process terminates in finite steps. When it terminates, it indicates one and only one of the two cases: (i) it fails to solve (1.2) to a specified precision; (ii) an approximate solution $\tilde{\mathbf{x}}$ to (1.2) within the specified precision has been found.
2. *Recoverability of the intended solution.* In case (ii) above, we are able to recover the intended solution $\mathbf{x}$ to (1.1) from $\tilde{\mathbf{x}}$ through an inexpensive computation.

Our specific design relies on a linear coding of the input system (1.1). For this reason, we refer to (1.2) as the *encoded system*, and the input system (1.1) as the *raw system*. There are three major components of our approach, which we discuss in the next three sections.

- The system abstraction—section 3. We define the types of faults and their semantics that our coded linear solver can handle.
- The encoding scheme—section 4. We add a set of rows and columns to the matrix to render the new linear system singular, but consistent. It remains this way for up to k componentwise faults in the solution vector $\tilde{\mathbf{x}}$.
- The solution process and recovery scheme—section 5. Our solution process relies on a conjugate gradient algorithm. We show that when this algorithm terminates, it does so at a consistent solution to the encoded system. We then describe how to recover the true solution $\mathbf{x}$ in light of the encoding.

These three components are highly interrelated, and our final design integrates the properties of all of them. Thus, we may leave particular details unspecified until future sections when relevant choices become obvious.

For simplicity, in this manuscript, we restrict ourselves to the case where $\boldsymbol{A}$ is symmetric positive definite (SPD) and validate the central concept in a synthetic environment with simulated faults. There are a number of important practical and implementational issues with this technique that need to be addressed in the future before it can offer a ready-to-deploy solution. Nevertheless, our initial results provide a promising proof of concept. We report results on the efficiency of our encoding in section 6. Our main findings are that the encoded system takes minimal additional work to solve in the presence of a fault. In the presence of a substantial number of faults (20% of components failing), it takes 8× the number of iterations of a no-fault linear solver. To achieve this, we use a new linear system that takes approximately 40% more iterations to solve if no faults occur. This demonstrates the potential for substantial savings, compared to a checkpoint-restart, deterministic replay, or active replica fault tolerant system.

2. Review of existing literature. There are a variety of existing techniques for fault tolerance that yield the three properties we need. These techniques can be broadly classified as system supported or algorithmic.

2.1. System-supported fault tolerance. System-supported fault tolerance schemes include checkpoint-restart, active replicas, and deterministic replay. Checkpoint-restart schemes involve the overhead of both consistent checkpointing and I/O (e.g., Bougeret et al. (2011)). These schemes are particularly difficult on ultrascale platforms, where I/O capacity and bandwidth are both at a premium relative to the compute capability and system memory. Furthermore, the asynchronous nature of many highly scalable algorithms makes it costly to identify consistent checkpoints and to perform associated rollbacks. Variants of these schemes include in-memory checkpointing, use of persistent storage (flash memory), and application-specified checkpoints.

Active replicas, commonly used in mission-critical real-time applications, execute multiple replicas of each task (Schneider, 1990). Failures are detected and replicas are managed to support real-time constraints. While the runtime characteristics of these schemes can be controlled (e.g., through worst-case runtime estimation), the resource overhead of such schemes are high, since tolerating a k-process failure among p processes requires $(k+1)p$ active processes. This cost is significant—as an example, tolerating 10 faults in an ensemble of 1000 cores (a 1% error rate) requires 11,000 processes!

Yet other systems such as MapReduce (Dean and Ghemawat, 2004) use the concept of deterministic replay. In this model, computation proceeds in steps with checkpoints at the end of each step. Processes are monitored within the steps, and in the event of a failure, the computation associated with the failed process is replayed at an alternate node. While this model has been successfully applied to a large set of wide-area distributed applications, it has the drawbacks of staged execution and increased job makespan, particularly when the number of faults is large. Furthermore, checkpointing to persistent storage (typically a distributed file system) can add significant overhead, particularly when the steps are small.

2.2. Algorithm-based fault tolerance. There are a number of possibilities for algorithm-based fault tolerance, depending on the features offered by the architecture and the types of errors handled.

Selective reliability. Perhaps the simplest model is selective reliability, where algorithms are programmed to be tolerant to faults in certain regions of the computations. Bridges et al. (2012) used this idea for an iterative inner-outer Krylov solver for $\boldsymbol{A}\mathbf{x} = \mathbf{b}$. The outer recurrence in a Krylov solver was assumed to be reliable, while an inner preconditioned iteration could produce faults, including soft errors. For soft errors, this setting then involves the framework of flexible Krylov subspace methods, like flexible GMRES (Saad, 1993, Eshof and Sleijpen, 2004, Simoncini and Szyld, 2003a,b). In general, these schemes require deep algorithmic and analytical insight to quantify fault tolerance properties and associated overheads. For instance, in the flexible GMRES framework, the magnitude of perturbations needs to be controlled to maintain convergence, although the perturbations can happen anywhere in the result of a numerical operation. Recent work on these problems focuses on new methods to generate realistic and challenging soft errors (Elliott, Hoemmen, and Mueller, 2015) to stress the methods.

Algorithmic checksumming and erasure coding. The result most closely related to ours is due to Huang and Abraham (Huang and Abraham, 1984). They consider a similar idea for detecting soft errors in dense matrix computations. To do so, they add a checksum row or column to a matrix to identify a single error in a dense matrix computation. For instance, for the computation of $\boldsymbol{C} = \boldsymbol{AB}$, let

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} \\ \mathbf{e}^T\boldsymbol{A} \end{bmatrix}, \qquad \tilde{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{B}\mathbf{e} \end{bmatrix}, \qquad \tilde{\boldsymbol{C}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{AB} & \boldsymbol{AB}\mathbf{e} \\ \mathbf{e}^T\boldsymbol{AB} & \mathbf{e}^T\boldsymbol{AB}\mathbf{e} \end{bmatrix}.$$

(Here $\mathbf{e}$ is the vector of all ones.) Single entry soft errors can be identified by searching for discrepancies with the checksum rows and columns in $\tilde{\boldsymbol{C}}$. This result motivated a line of research (Chen et al., 2005, Chen and Dongarra, 2005, 2008, Bosilca et al., 2009, Chen, 2009, 2011) on generalizing the concept and dealing with fail-stop failures. For instance, Chen et al. (Chen et al., 2005, Chen and Dongarra, 2005) consider the use of real-valued matrices to construct error codes for fault tolerance in matrix computations. This concept is extended to the computation of parallel matrix multiplication (Chen

and Dongarra, 2008), which motivates the development of new coding matrices for multiple errors (Chen, 2009). Some requirements on the coding matrices were noted by Bosilca et al. (2009). These ideas are subsequently studied in the context of iterative methods (Chen, 2011), where the focus is on efficiently emulating a checkpoint-restart system, where the efficiently computed checksums are used instead of the checkpoints (if the same information isn't available due to the matrix structure).

Comparison with our work. In comparison with the selective reliability work, our erasure-coded approach only requires the encoding and final decoding to be reliable, and these involve a small amount of work. Our current setting is designed to handle fail-stop failures, and we plan to investigate soft errors in the future. In comparison with the algorithm checksum work, our ideas explore the use of coding schemes in iterative algorithms in an entirely different manner, where we clearly establish a general framework for these ideas in iterative methods. This involves concepts related to Kruskal rank, as mentioned in previous work on algorithmic fault tolerance (Bosilca et al., 2009). Prior work in fault tolerant iterative methods using coding involves some similar ideas (Chen, 2011), but in the context of algorithms that restore the state on a per iteration basis instead of solution recovery after the entire computation, as in our case.

3. Assumptions of the system abstraction. We begin our technical description by stating our assumptions regarding the underlying execution environment, so that we can give precise algorithms in subsequent sections. We view the execution of the algorithm in two stages—the setup phase and the execution phase. The setup phase consists of the input augmentation step of the algorithm. During this phase, we assume that a small amount of reliable work can be done. In the presence of faults, this can be achieved using more expensive fault tolerance techniques such as replicated execution or deterministic replay. Since this step is a very small fraction of the overall computation (less than 1% for typical systems), the overhead is not significant. This is equivalent to the selected reliability model of Bridges et al. (2012).

The execution phase of the algorithm corresponds to the solve over the augmented system. In this phase, we assume an ensemble of message passing processes executing the solver. We also assume fail-stop failures, i.e., in the event of a fault, a process halts. No further messages are received from this process by any of the other processes. All of the other processes involved are able to reliably detect this halt as well. We do not assume any distribution of temporal faults, only that the total number of faults is bounded.

Indeed there are other fault models as well, ranging from transient (soft) faults to Byzantine behavior. Soft faults manifest themselves in the form of erroneous data. These data, when incorporated into data at other processes, can lead to cascading error in programs. In principle, our proposed method can be extended to these other fault classes using existing fault detection schemes. For instance, messages signed with a checksum allow us to detect on-the-wire errors. Asserts in the program, corresponding to predicates whose violation signifies an error can be used to detect soft errors in processes. If these asserts are further extended to algorithmic invariants, additional errors can be detected (Elliott, Hoemmen, and Mueller, 2014). When a soft error is detected at a process, the process is killed, once again resulting in a fail-stop failure. Asserts work similarly when Byzantine failures are detected. Thus, a combination of tolerance to fail-stop failures with fault detection techniques allows us to deal with a broad set of faults, although we focus on simple fail-stop cases here and do not consider the more general setting.

Furthermore, we note that there are many system-level software issues associated with the proposed fault-oblivious paradigm and this fault model. For instance, in many APIs (such as MPI), a single process failure can cause the entire program to crash. In yet other scenarios, a crashed process can cause group communication operations (reductions, broadcasts, etc.) to block. These kinds of program behaviors would not allow our proposed scheme to work. We assume program behavior in which faulty processes simply drop out of the ensemble while the rest continue. We leave the (non-trivial) design of these fault-oblivious APIs to future work and focus on the feasibility and superior performance of the erasure-coded computation scheme for linear solvers in this manuscript. For an example of some details for related approaches, see Wilke et al. (2015), Gamell et al. (2015) where they use key-value semantics overlayed on the MPI protocol and a whole-scale replacement to MPI to enable fault tolerant APIs.

4. Encoding scheme. We begin by formalizing some notation. Let $\mathbf{x}^*$ be the true solution of the raw linear system $\boldsymbol{A}\mathbf{x} = \mathbf{b}$. Let $k \leq n$ be the number of allowed faults during its execution. Let $\boldsymbol{E} \in \mathbb{R}^{n\times k}$ be an encoding matrix that we'll specify completely, shortly. We design the augmented matrix $\tilde{\boldsymbol{A}} \in \mathbb{R}^{(n+k)\times(n+k)}$ as follows:

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{AE} \\ \boldsymbol{E}^T\boldsymbol{A} & \boldsymbol{E}^T\boldsymbol{AE} \end{bmatrix}. \tag{4.1}$$

We choose the encoding of $\mathbf{x}^*$ to be an embedding into $\mathbb{R}^{n+k}$, i.e.,

$$\tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix}. \tag{4.2}$$

Accordingly, the encoding of $\mathbf{b}$ is given by

$$\tilde{\mathbf{b}} = \tilde{\boldsymbol{A}}\tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{b} \\ \boldsymbol{E}^T\mathbf{b} \end{bmatrix}. \tag{4.3}$$

4.1. Basic properties. We now establish a few properties of these systems in terms of their rank, a characterization of the solutions, and the semidefiniteness of $\tilde{\boldsymbol{A}}$.

PROPOSITION 4.1. *A null space basis of* $\tilde{\boldsymbol{A}}$ *is* $\left[\begin{smallmatrix} \boldsymbol{E} \\ -\boldsymbol{I}_k \end{smallmatrix}\right]$.

Proof. From the design of $\tilde{\boldsymbol{A}}$ in (4.1) and $\boldsymbol{A}$ being SPD, we have $\operatorname{rank}(\tilde{\boldsymbol{A}}) = n$. Thus the null space has dimension k. Then, by inspection, $\tilde{\boldsymbol{A}}\left[\begin{smallmatrix} \boldsymbol{E} \\ -\boldsymbol{I}_k \end{smallmatrix}\right] = 0$ and $\left[\begin{smallmatrix} \boldsymbol{E} \\ -\boldsymbol{I}_k \end{smallmatrix}\right]$ has column rank k. ∎

As a corollary of the above proposition, we have the following proposition regarding the nonambiguity of the solution encoding (4.2).

PROPOSITION 4.2. *Let* $\left[\begin{smallmatrix} \mathbf{y} \\ \mathbf{z} \end{smallmatrix}\right]$ *be any solution of* (1.2) *where* $\mathbf{y} \in \mathbb{R}^n$. *Once* $\mathbf{z} \in \mathbb{R}^k$ *is specified, then the components of* $\mathbf{y}$ *are uniquely determined. Moreover, if* $\mathbf{z} = 0$, *then* $\mathbf{y} = \mathbf{x}^*$.

Proof. Note that $\left[\begin{smallmatrix} \mathbf{x}^* \\ 0 \end{smallmatrix}\right]$ is a solution to (1.2). Thus, any solution to (1.2) can be written as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{E} \\ -\boldsymbol{I}_k \end{bmatrix} \mathbf{a}$$

for a unique $\mathbf{a} \in \mathbb{R}^k$. Due to the nonzero structure, we have $\mathbf{a} = -\mathbf{z}$. Hence, $\mathbf{y}$ is uniquely determined as $\mathbf{x}^* - \boldsymbol{E}\mathbf{z}$. The final statement follows from $\mathbf{z} = 0$. ∎

This theorem will be important because if we have a solution $\left[\begin{smallmatrix}\mathbf{y}\\ \mathbf{z}\end{smallmatrix}\right]$, then to recover $\mathbf{x}^*$ we need to compute $\mathbf{y} + \boldsymbol{E}\mathbf{z}$ only. As we establish the remainder of our framework, this will give a straightforward recovery algorithm.

We now prove that $\tilde{\boldsymbol{A}}$, as given in (4.1), is symmetric positive semidefinite (SPSD).

PROPOSITION 4.3. *If $\boldsymbol{A}$ is SPD, then $\tilde{\boldsymbol{A}}$ as defined in* (4.1) *is SPSD.*

Proof. Let the Cholesky factorization of $\boldsymbol{A}$ be $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$. Again, by inspection, we have

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{E}^T\boldsymbol{A}\boldsymbol{L}^{-T} \end{bmatrix} \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{E}^T\boldsymbol{A}\boldsymbol{L}^{-T} \end{bmatrix}^T. \qquad \square$$

4.2. Solution degeneracies and faults. The matrix $\tilde{\boldsymbol{A}}$ has rank n, despite having $n + k$ rows and columns. We now show how a specific use of this degeneracy allows us to have fault tolerant solutions to (1.2). Let $\tilde{\mathbf{x}} \in \mathbb{R}^{(n+k)}$ be the encoded solution. For the sake of clarity in presentation, we assume that faults *only* occur within the components of $\tilde{\mathbf{x}}_{1:n}$, i.e., the redundant components $\tilde{\mathbf{x}}_{(n+1):(n+k)}$ introduced by the encoding cannot be faulty. Please note that this is not a limitation of our scheme, if faults occur in the components of the solution corresponding to the encoding, it suffices to set these components to zero and proceed. (This is what we assume occurs on failure in section 5.) In the case that the encoded components fail, then, it simply corresponds to reduced redundancy so that if $m < k$ components fail in the encoding, we can only tolerate $k - m$ failures in the actual components of the solution.

We let the set of faulty indices be $\mathcal{F} \subset [n]$. We constrain the cardinality $|\mathcal{F}| \leq k$. Let $\mathcal{C}$ be the set of nonfaulty (or correct) components. Without loss of generality, we consider the system (1.2) in three components corresponding to the correct, faulty, and redundant components of the solution. This is equivalent to a permutation, after which we have that the solution is

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix} \begin{array}{l} \text{correct} = \tilde{\mathbf{x}}_{\mathcal{C}}, \\ \text{faulty} = \tilde{\mathbf{x}}_{\mathcal{F}}, \\ \text{redundant}. \end{array} \tag{4.4}$$

The overall permuted system is:

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{Z}_1 \\ \boldsymbol{A}_{12}^T & \boldsymbol{A}_{22} & \boldsymbol{Z}_2 \\ \boldsymbol{Z}_1^T & \boldsymbol{Z}_2^T & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \boldsymbol{E}^T\mathbf{b} \end{bmatrix} \text{ where } \begin{cases} \boldsymbol{Z}_1 &= \boldsymbol{A}_{11}\boldsymbol{E}_1 + \boldsymbol{A}_{12}\boldsymbol{E}_2, \\ \boldsymbol{Z}_2 &= \boldsymbol{A}_{12}^T\boldsymbol{E}_1 + \boldsymbol{A}_{22}\boldsymbol{E}_2, \\ \boldsymbol{R} &= \boldsymbol{E}^T\boldsymbol{A}\boldsymbol{E}. \end{cases}$$

As our solver progresses, the components in $\mathbf{f}$ become "stuck" at some intermediate values as faults occur. We describe the semantics of these faults more formally in the next section (section 5). The goal of this section is to show that we can recover solutions even when setting $\mathbf{f}$ to some arbitrary value.

For our erasure-coded solver, we need a condition on the matrix $\boldsymbol{E}$ such that

1. there is always a solution to (1.2) for any $\mathbf{f}$ as long as $|\mathcal{F}| \leq k$ (Proposition 4.5);
2. given any solution computed with faulty components ($|\mathcal{F}| \leq k$), we can extract a solution to (1.2) (Proposition 4.6).

The condition on the matrix $\boldsymbol{E}$ that is essential to these results is the Kruskal rank. Recall the definition.

DEFINITION 4.4 (Kruskal rank (Kruskal, 1977)). *The Kruskal rank, or k-rank, of a matrix is the largest number k such that every subset of k columns is linearly independent.*

Notice that the condition for a matrix to be of Kruskal rank k is much stronger than being rank k. The following proofs require that the Kruskal rank of $\boldsymbol{E}^T$ is k in order to handle up to k faults. Some intuition for this requirement is that we need the matrix $\boldsymbol{E}$ to encode redundancies to any possible faults with a number up to k. Recovering the solution will require us to invert a matrix for the components where the solution was faulty and, hence, we need all possible subsets of k rows of $\boldsymbol{E}$ to be invertible—giving us the Kruskal rank condition. This condition was also noted by Bosilca et al. (2009) in the context of coding for matrix multiplication. We now present these two results formally.

PROPOSITION 4.5. *Let $\boldsymbol{E}^T \in \mathbb{R}^{k\times n}$ have Kruskal rank k and let $\mathcal{F}$ be an arbitrary subset of $[n]$ with $|\mathcal{F}| \le k$. Then there exists a solution to* (1.2) *with $\tilde{\mathbf{x}}_{\mathcal{F}} = \mathbf{f}$ for any $\mathbf{f}$. When $|\mathcal{F}| = k$, such a solution is unique.*

Proof. Note that any solution of (1.2) has the form

$$\begin{bmatrix}\mathbf{x}^*\\ 0\end{bmatrix} + \begin{bmatrix}\boldsymbol{E}\\ -\boldsymbol{I}\end{bmatrix}\mathbf{a}$$

for some $\mathbf{a} \in \mathbb{R}^k$. Let us permute this solution as in (4.4):

$$\begin{bmatrix}\mathbf{c}\\ \mathbf{f}\\ \mathbf{r}\end{bmatrix} = \begin{bmatrix}\mathbf{x}_1^*\\ \mathbf{x}_2^*\\ 0\end{bmatrix} + \begin{bmatrix}\boldsymbol{E}_1\\ \boldsymbol{E}_2\\ -\boldsymbol{I}\end{bmatrix}\mathbf{a}.$$

It suffices to show that there exists $\mathbf{a}$ such that $\mathbf{f} = \mathbf{x}_2^* + \boldsymbol{E}_2\mathbf{a}$. Because the rows of $\boldsymbol{E}_2$ correspond to the faulty components, this is a set of $|\mathcal{F}|$ columns from $\boldsymbol{E}^T$. These columns are linearly independent by the Kruskal rank condition. Thus, there exists a solution to this underdetermined linear system. If $|\mathcal{F}| = k$, then the system is square and nonsingular, so the vector $\mathbf{a}$ is unique. ∎

According to our fault model, as faults occur during an iterative process, the components of $\mathbf{f}$ become stuck, because the processes responsible for updating them have dropped out of the computation. Thus, the actual system that we solve is what we call a purified system consisting of only nonfaulty components:

$$\begin{bmatrix}\boldsymbol{A}_{11} & \boldsymbol{Z}_1\\ \boldsymbol{Z}_1^T & \boldsymbol{R}\end{bmatrix}\begin{bmatrix}\mathbf{c}\\ \mathbf{r}\end{bmatrix} = \begin{bmatrix}\mathbf{b}_1\\ \boldsymbol{E}^T\mathbf{b}\end{bmatrix} - \begin{bmatrix}\boldsymbol{A}_{12}\\ \boldsymbol{Z}_2^T\end{bmatrix}\mathbf{f}. \tag{4.5}$$

By Proposition 4.5, if the encoding matrix $\boldsymbol{E}^T$ has Kruskal rank k, there exists a solution to the purified subsystem (4.5) from a solution to (1.2) with $\mathbf{x}_{\mathcal{F}} = \mathbf{f}$ fixed. We now ask the reverse question. Suppose $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{r}\end{smallmatrix}\right]$ is any solution to (4.5), will $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{f}\\ \mathbf{r}\end{smallmatrix}\right]$ be a solution to (1.2) (under the permutation)? The following proposition shows the answer is yes. The reason we need this proof is that there are many possible solutions to the purified subsystem. We need to establish that all solutions to (4.5) with $\mathbf{f}$ fixed will lead us to a full solution to (1.2).

PROPOSITION 4.6. *Let $\boldsymbol{E}^T \in \mathbb{R}^{k\times n}$ have Kruskal rank k. Let $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{r}\end{smallmatrix}\right]$ be any solution to the purified system* (4.5) *with $\mathbf{f}$ fixed and where $|\mathcal{F}| \le k$. Then $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{f}\\ \mathbf{r}\end{smallmatrix}\right]$ is a solution to* (1.2).

Proof. This proof is equivalent to checking whether the following equation is satisfied by the purified solution:

$$\begin{bmatrix}\boldsymbol{A}_{12}^T & \boldsymbol{Z}_2\end{bmatrix}\begin{bmatrix}\mathbf{c}\\ \mathbf{r}\end{bmatrix} = \mathbf{b}_2 - \boldsymbol{A}_{22}\mathbf{f}. \tag{4.6}$$

We establish this fact algebraically from the solution of the purified system. First note that $\boldsymbol{E}_2^T$ is a k-by-$|\mathcal{F}|$ matrix with full column rank, and thus it has a left inverse $(\boldsymbol{E}_2^T)^\dagger = (\boldsymbol{E}_2\boldsymbol{E}_2^T)^{-1}\boldsymbol{E}_2$. Now consider the two equations in the purified system:

$$\boldsymbol{A}_{11}\mathbf{c} + \boldsymbol{Z}_1\mathbf{r} = \mathbf{b}_1 - \boldsymbol{A}_{12}\mathbf{f}, \tag{4.7}$$

$$\boldsymbol{Z}_1^T\mathbf{c} + \boldsymbol{R}\mathbf{r} = \boldsymbol{E}^T\mathbf{b} - \boldsymbol{Z}_2^T\mathbf{f}. \tag{4.8}$$

The result of (4.8) $-\ \boldsymbol{E}_1^T$(4.7) is

$$\boldsymbol{E}_2^T\boldsymbol{A}_{12}^T\mathbf{c} + \boldsymbol{E}_2^T\boldsymbol{Z}_2\mathbf{r} = \boldsymbol{E}_2^T\mathbf{b}_2 - \boldsymbol{E}_2^T\boldsymbol{A}_{22}\mathbf{f}.$$

To complete the proof, we premultiply this equation by left inverse $(\boldsymbol{E}_2^T)^\dagger$. ∎

5. The solution process and recovery scheme. Proposition 4.3 establishes $\tilde{\boldsymbol{A}}$ being SPSD when $\boldsymbol{A}$ is SPD. Thus we can apply the conjugate gradient (CG) method to solve a singular but consistent linear system (1.2) (Ipsen and Meyer, 1998). Specifically, we use the following two-term recurrence form of CG (Meurant, 2006).

Algorithm 1 Fault oblivious CG with a two-term recurrence. When we notice a fault, we set $\beta_t = 0$ at that iteration.

1: Let $\mathbf{x}_0$ be the initial guess and $\mathbf{r}_0 = \mathbf{b} - \boldsymbol{A}\mathbf{x}_0$, $\beta_0 = 0$.
2: **for** $t = 0, 1, \ldots$ until convergence **do**
3: $\quad\beta_t = (\mathbf{r}_t, \mathbf{r}_t)/(\mathbf{r}_{t-1}, \mathbf{r}_{t-1})$
4: $\quad\mathbf{p}_t = \mathbf{r}_t + \beta_t\mathbf{p}_{t-1}$
5: $\quad\mathbf{q}_t = \boldsymbol{A}\mathbf{p}_t$
6: $\quad\alpha_t = (\mathbf{r}_t, \mathbf{r}_t)/(\mathbf{q}_t, \mathbf{p}_t)$
7: $\quad\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t\mathbf{p}_t$
8: $\quad\mathbf{r}_{t+1} = \mathbf{r}_t - \alpha_t\mathbf{q}_t$
9: **end for**

We consider the setting when Algorithm 1 is executed in a distributed environment. For the encoded system (1.2), this means the encoded matrix $\tilde{\boldsymbol{A}}$ and the encoded vectors are distributed among multiple processes by rows. Let the index set associated with process i be $\mathcal{I}_i$, then $[n+k] = \bigcup_i \mathcal{I}_i$. According to our fault model, the operations of Algorithm 1 affected by faults in a distributed environment are the aggregation operations—inner products and the matrix-vector multiplication $\boldsymbol{A}\mathbf{p}_t$. Thus our erasure-coded CG can be defined by specifying the semantics of these two aggregation operations under faults. At the tth iteration of erasure-coded CG, let the set of failed processes be $\mathcal{P}_t$. Then the set of faulty indices is $\mathcal{F}_t = \bigcup_{i\in\mathcal{P}_t}\mathcal{I}_i$. We assume that each viable process can detect the breakdown of any process that should be sending it data on their local variables, which we call neighbor processes. Based on this assumption, we specify the semantics of the two aggregation operations as follows:

- Inner products $(\mathbf{r}_t, \mathbf{r}_t)$ and $(\mathbf{q}_t, \mathbf{p}_t)$. The viable processes carry out the all-reduce operation by skipping the faulty components $\mathcal{F}_t$ in the vectors:

$$\begin{aligned}(\mathbf{r}_t, \mathbf{r}_t) &= \left((\mathbf{r}_t)_{[n+k]\setminus\mathcal{F}_t}, (\mathbf{r}_t)_{[n+k]\setminus\mathcal{F}_t}\right),\\ (\mathbf{q}_t, \mathbf{p}_t) &= \left((\mathbf{q}_t)_{[n+k]\setminus\mathcal{F}_t}, (\mathbf{p}_t)_{[n+k]\setminus\mathcal{F}_t}\right).\end{aligned} \tag{5.1}$$

  Furthermore, we require no reuse of aggregation operation results. This means that when computing α_t, $(\mathbf{r}_t, \mathbf{r}_t)$ is recomputed simultaneously with $(\mathbf{q}_t, \mathbf{p}_t)$. Similarly when computing β_t, $(\mathbf{r}_{t-1}, \mathbf{r}_{t-1})$ is recomputed simultaneously with $(\mathbf{r}_t, \mathbf{r}_t)$. For this purpose, we have to maintain both $\mathbf{r}_t$ and $\mathbf{r}_{t-1}$.

- Matrix-vector multiplication $\mathbf{q}_t = \boldsymbol{A}\mathbf{p}_t$. A viable process carries out its local aggregation operation for computing $\boldsymbol{A}_{\mathcal{I}_i,:}\mathbf{p}_t$ by skipping the faulty components $\mathcal{F}_t$ in $\mathbf{p}_t$:

$$\boldsymbol{A}_{\mathcal{I}_i,:}\mathbf{p}_t = \boldsymbol{A}_{\mathcal{I}_i,[n+k]\setminus\mathcal{F}_t}(\mathbf{p}_t)_{[n+k]\setminus\mathcal{F}_t}.$$

Given the above semantics on aggregation operations, the erasure-coded CG on $\tilde{\mathbf{x}}$ can be effectively considered as an iterative solution process on the subsystem defined on $\tilde{\mathbf{x}}_{[n+k]\setminus\mathcal{F}_t}$, as given in (4.5). Note that the right-hand side of the purified subsystem (4.5) depends on $\mathbf{f}$, the snapshot value of $\tilde{\mathbf{x}}_{\mathcal{F}_t}$. For this reason, we require each viable process to cache the freshest snapshot values it have received from its neighbor processes.

Another technical issue we need to consider when faults happen is the update to the search direction $\mathbf{p}_t$. In fault-free CG with exact arithmetic, we have $(\mathbf{r}_t, \mathbf{p}_{t-1}) = 0$. However, given the semantics of inner product, as defined in (5.1), the orthogonality of $\mathbf{r}_t$ and $\mathbf{p}_{t-1}$ will generally not hold. When we observe a fault, we truncate the update $\mathbf{p}_t = \mathbf{r}_t + \beta_t\mathbf{p}_{t-1}$ to be

$$\mathbf{p}_t = \mathbf{r}_t.$$

This corresponds to a reset of the Krylov process.

We now consider the recovery of the solution to the raw system (1.1). Suppose the erasure-coded CG converges on the encoded system (1.2) after T iterations. Let the encoding matrix $\boldsymbol{E}^T \in \mathbb{R}^{k\times n}$ have Kruskal rank k. Let $\mathcal{F} \in [n]$ be the set of all faulty indices upon convergence such that $|\mathcal{F}| \le k$. In erasure-coded CG, the snapshot value $\mathbf{f}$ of the faulty components $\tilde{\mathbf{x}}_{\mathcal{F}}$ are cached on the viable processes. Because of the semantics of aggregation operations, the erasure-coded CG solves the purified subsystem defined in (4.5). Let the returned approximate solution be $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{r}\end{smallmatrix}\right]$. According to Proposition 4.6, then

$$\tilde{\mathbf{x}} = \begin{bmatrix}\mathbf{c}\\ \mathbf{f}\\ \mathbf{r}\end{bmatrix}$$

is a solution to the encoded system (1.2). Then, by Proposition 4.1, we can recover the intended solution to the raw system (1.1) through the equation

$$\begin{bmatrix}\mathbf{x}^*\\ 0\end{bmatrix} = \tilde{\mathbf{x}} + \begin{bmatrix}\boldsymbol{E}\\ -\boldsymbol{I}_k\end{bmatrix}\mathbf{r}, \tag{5.2}$$

and by Proposition 4.2, the recovery equation (5.2) is nonambiguous.

6. Experimental results. In this section, we report on experimental results with varying degrees of faults and the associated overhead. The main purpose of these experiments is to demonstrate the feasibility of the concept of an erasure-coded linear system solver. We also empirically investigate the relationship between the condition number of the encoded system and the raw system. Through these experiments, we also identify critical research problems to be solved in order to realize the idea of an erasure-coded linear solver in a distributed setting. We do not intend these experiments to be exhaustive and comment on a few limitations in the future work section. In the interest of making our experiments reproducible, we provide the original experimental codes as well as our numerical results, at the website: https://github.com/dgleich/erasure-coded-cg.

TABLE 1
Test matrices.

Matrix	n	nnz	Type
`Ltridiag500`	500	1498	1D model problem
`mhdb416`	416	2312	electromagnetics problem
`nos3`	960	15,844	structural problem

6.1. Experiment design. In our experiments, we use three SPD test matrices, with their sizes (n), number of nonzeros (nnz), and types shown in Table 1. `Ltridiag500` is a 500×500 one-dimensional (1D) model matrix with the stencil $[\ -1 \quad 2 \quad -1\ ]$. `mhdb416` (Kooper et al., 1995) and `nos3` (Duff, Grimes, and Lewis, 1989) are from the University of Florida Sparse Matrix Collection (Davis and Hu, 2011). We implemented a MATLAB code to simulate the erasure-coded CG described in section 5. In our current simulation, we inject fault components only at the end of a CG iteration. Furthermore, we assume that all the faults happen at the same time. This is a limited simulation, but it suffices to demonstrate the feasibility of the fundamental idea; we are planning more realistic experiments in true parallel settings in the future. For a given number of tolerable faults, k, we design the $n \times k$ encoding matrix $\boldsymbol{E}$ as a random Gaussian matrix scaled by $\frac{1}{\sqrt{n}}$, i.e.,

$$\boldsymbol{E} = \frac{1}{\sqrt{n}}\bar{\boldsymbol{E}}, \qquad \bar{\boldsymbol{E}}_{ij} \sim \mathcal{N}(0,1), \qquad i = 1, \ldots, n,\ j = 1, \ldots, k.$$

This matrix has Kruskal rank k, with high probability. This choice has also been proposed previously in the context of real-valued erasure codes (Chen and Dongarra, 2005, Chen, 2009). This choice does increase the number of nonzeros in the matrices, which results in an increase in the per iteration time. At the moment, however, we are focused on the theoretical aspects and bounding the number of iterations. In future efforts aimed at deploying these concepts in a parallel setting, we plan to use *sparse* encoding matrices that will not result in a significant increase in nonzeros (see the discussion in the future work section).

6.2. Experimental results. For each test matrix in Table 1, we run an erasure-coded CG simulation code for $k = 0, 1, 20\%n$ number of faults. The right-hand side $\mathbf{b}$ of the raw system (1.1) is produced from random solution vectors $\mathbf{x}$ in $(0,1)^n$. We set the number of maximum CG iterations to be $10n$, and monitor the convergence using the stopping criterion $\|\mathbf{r}_t\|_2 \leq 10^{-10}$. For each value of k, the k fault components are randomly selected from $[n]$. These fault components are injected simultaneously at the end of one randomly selected CG iteration that is no larger than $0.25n$. We refer to this CG iteration as the fault point. In Figures 1–3, we plot the residual norm versus CG iteration for the three test matrices; and from left to right for $k = 0, 1, 20\%n$. The fault point is marked by the red cross.

To evaluate the quality of the recovered solution $\mathbf{x}^*$, we compute the relative residual achieved by $\mathbf{x}^*$ on the raw system (1.1):

$$\frac{\|\mathbf{b} - \boldsymbol{A}\mathbf{x}^*\|_2}{\|\mathbf{b}\|_2}.$$

We summarize the number of iterations and relative residuals on the raw system for three test matrices in Tables 2–4. We observe that on `Ltridiag500` and `nos3`, when

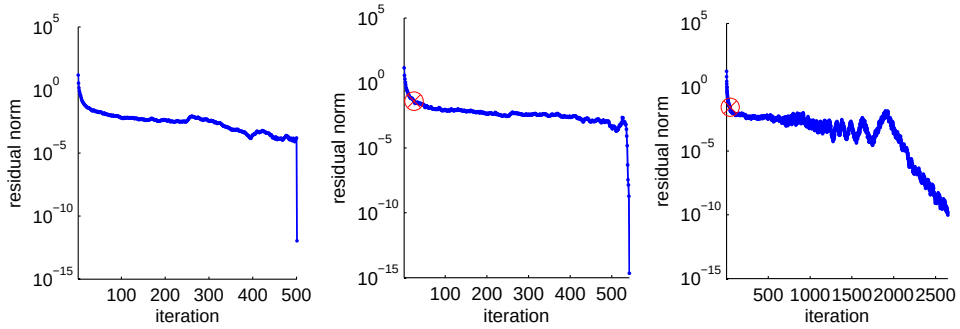

FIG. 1. `Ltridiag500` *convergence. From left to right* $k = 0, 1, 20\%$. *The fault point is marked by the red cross.*

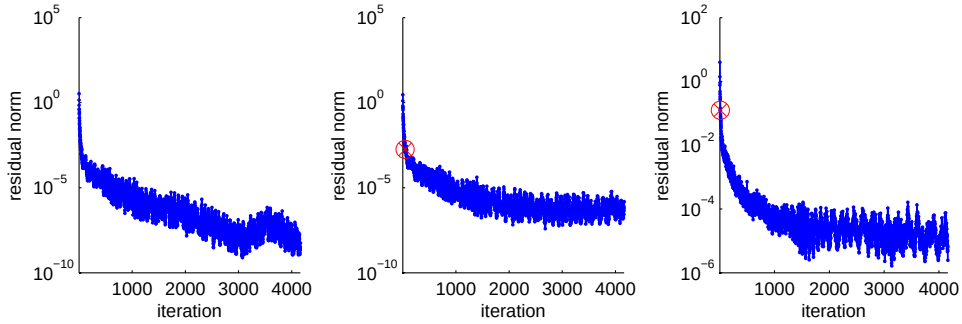

FIG. 2. `mhdb416` *residual norm. From left to right* $k = 0, 1, 20\%$. *The fault point is marked by the red cross.*

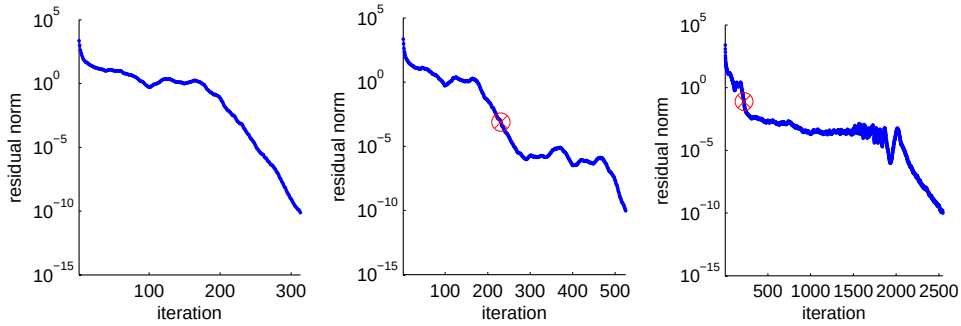

FIG. 3. `nos3` *residual norm. From left to right* $k = 0, 1, 20\%$. *The fault point is marked by the red cross.*

there is only 1 fault, the erasure-coded CG can recover a solution with almost the same quality as if there is no fault, with a comparable number of iterations. When there are 20% fault components, the erasure-coded CG can still recover an approximate solution with good quality, although the number of iterations increases substantially. In contrast to `Ltridiag500` and `nos3`, on `mhdb416`, the erasure-coded CG reaches the maximum number of iterations (i.e., $10n$) for all three values of k. Comparing the solution quality of $k = 0$ to those of $k = 1, 20\%n$ in Table 3 indicates more iterations are required for the latter two cases.

TABLE 2
Results on `Ltridiag500`.

k	Iteration number	Relative residual on raw system
0	500	1.39×10^{-14}
1	540	3.76×10^{-15}
20%	2640	3.72×10^{-11}

TABLE 3
Results on `mhdb416`.

k	Iteration number	Relative residual on raw system
0	4160	1.19×10^{-9}
1	4160	1.47×10^{-5}
20%	4160	2.09×10^{-6}

TABLE 4
Results on `nos3`.

k	Iteration number	Relative residual on raw system
0	312	3.51×10^{-14}
1	524	4.09×10^{-14}
20%	2581	1.91×10^{-13}

7. Effects on conditioning. The previous experiments show that the number of iterations can vary with the size of the encoding block. We now empirically investigate one aspect of this property. For this study, recall that the runtime of the CG solver can be bounded by the spectral condition number of the matrix (Meurant, 2006, Theorem 2.30), as well as the dimension of the matrix. Both measures increase as a result of our encoding procedure. In the next set of experiments, we show the impact of encoding on the condition numbers to understand this effect.

For these experiments, we are only concerned with the condition number of the *matrix*, and not the condition number associated with the *problem*. The matrix-condition number provides an upper bound on the condition number of the problem of solving a linear system. In this case, the linear system that results from the encoding procedure has additional structure that may enable a more refined conditioning analysis, but we leave this to future work.

Given a square matrix $\boldsymbol{A}$, its spectral condition number is based on the largest and smallest nonzero singular values, $\sigma_{\max}$ and $\sigma_{\min}$, of $\boldsymbol{A}$:

$$\kappa(\boldsymbol{A}) = \sigma_{\max}/\sigma_{\min}.$$

In the following results, κ will denote the condition number of the raw system and κ' will denote the condition number of $\tilde{\boldsymbol{A}}$. We are primarily concerned with the ratio κ'/κ, because this reflects the impact of the erasure coding on the spectral condition number. In all of our tests, this ratio was positive, showing that our procedure increases the condition number of the matrix. Note that if $\kappa'/\kappa = \sigma$ then, based on (Meurant, 2006, Theorem 2.30), we expect CG to take approximately $\sqrt{\sigma}$ additional iterations for the encoded problem. (This holds when κ is large and we use $\log((\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)) = \log(1-2/(\sqrt{\kappa}+1)) \approx \log(1-2/\sqrt{\kappa}) = -2/\sqrt{\kappa}$ to estimate the number of iterations.)

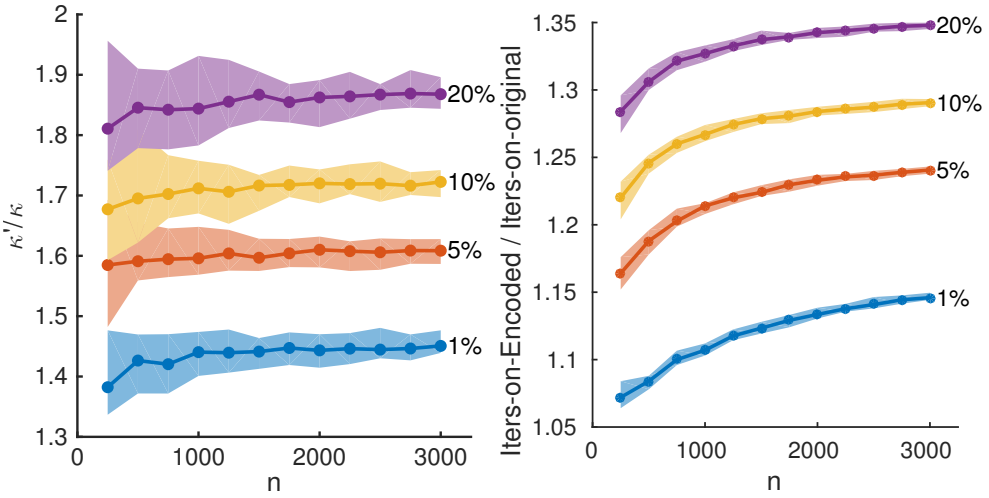

FIG. 4. *(At left) The ratio between the condition number of the encoded matrix κ' and the original system κ remains bounded as the matrix size (n) increases and when we add encoding matrices to tolerate k% of entries possibly failing. (At right) The actual number of iterations taken by the CG method shows the effect of the change in condition number and this also remains bounded. The matrix used in this study is an n-by-n tridiagonal Laplacian matrix with coefficients $-1, 2, -1$. The thick line shows the median over 25 trials and the shaded region shows the maximum and minimum results.*

In the first experiment, we study the change in condition number as the matrix size grows for the tridiagonal matrix with stencil $\begin{bmatrix}-1 & 2 & -1\end{bmatrix}$, and as the percentage of faults varies from 1% of the matrix size to 20% of the matrix size. The results are shown in Figure 4 (left). Because we use random encoding matrices, we show a distribution of results over 25 trials, and focus on the maximum, minimum, and median results. These results show that κ'/κ appears to converge to a constant as the matrix size increases and only depends on the fraction of redundancies. With $k = 5\%$ of n, for instance, κ'/κ appears to converge to 1.6. This means that we expect CG to take $\sqrt{1.6} \approx 1.27$ times more iterations with this number of columns.

In the right portion of Figure 4, we test this hypothesis and find that CG takes *slightly fewer* than the expected number of iterations in the regime we studied. With $k = 5\%$ of n, again, we see it takes about 1.23 times more iterations. Our studies, at the moment, are on matrices with dimension up to 3000, and the iteration counts do appear to be slightly increasing. We expect them to converge to the expected change in iteration count for larger matrices.

Finally, we verify our findings over a larger space of matrices. To do so, we select all matrices from the University of Florida collection from $n = 500$ to $n = 10{,}000$ that are symmetric and positive definite. We eliminate any matrices whose condition numbers are larger than $\kappa = 5 \times 10^{15}/n$, and also those where $\kappa < 50$. This results in a selection of 60 matrices. For each matrix, we compute the condition number ratio κ'/κ, with $k = 20\%$ of n. The results in Figure 5 show that as the raw condition number κ increases, the change to the condition number κ' appears to remain bounded by a small constant. When κ is small, the change can be larger. The worst increase is for `Norris/fv3`, which is a finite element mesh on a two-dimensional (2D) plane that is well conditioned. For matrices with large condition numbers, the worst result is for `HB/nos2`, which is again from a finite element problem on a 2D plane. However, this corresponds to a smaller perturbation than for `Norris/fv3`.

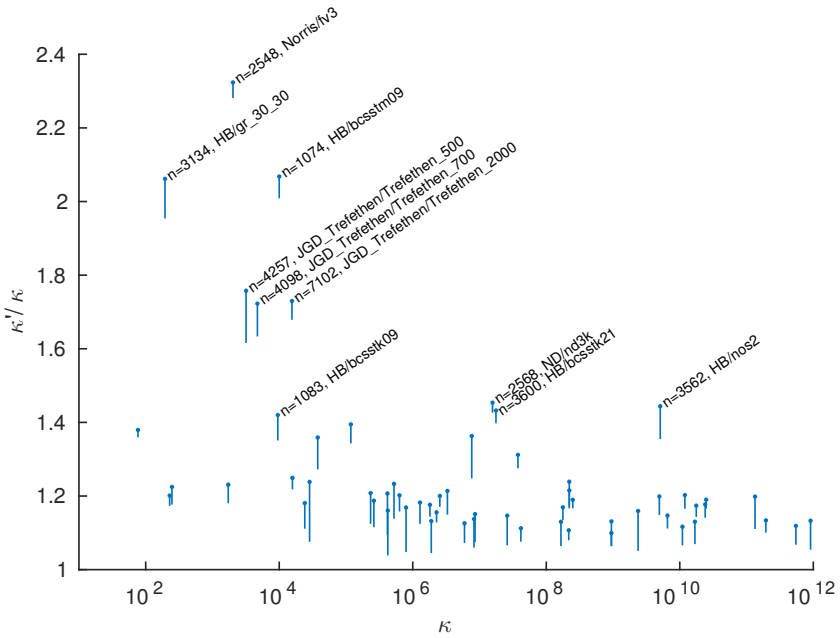

FIG. 5. *For a large set of positive definite matrices from the University of Florida collection, the finding that the change in relative change condition number is bounded holds. Each dot and line shows the maximum condition number ratio and the range as a function of the original condition number when we add an encoding matrix for* 20% *of entries failing. These results show that matrices with* small *condition number initially tend to exhibit the largest increases.*

TABLE 5

This table shows statistics for the worst *change in conditioning that arose in* 25 *trials on the UF dataset. So the result in the* 1% *row of* 1.082 *shows that for* 85% *of the* 60 *matrices, the largest change observed over* 25 *trials was less than* 1.082. *The* 95%*-tile and max results tend to measure matrices with small condition numbers as illustrated in Figure* 5.

Redundancies	κ'/κ			
	Median	80%-tile	95%-tile	Max
1%	1.024	1.082	1.423	1.807
5%	1.070	1.185	1.572	2.001
10%	1.118	1.259	1.713	2.141
20%	1.199	1.371	1.910	2.323

Overall, these experiments point towards a modest increase in iteration counts resulting from our encoding procedure. The median result over the 60 matrices of the worst increase over 25 trials is $\kappa'/\kappa = 1.199$, which results in about 10% additional iterations to encode for 20% redundancies. For 1% redundancies, the median κ'/κ is 1.024, giving an additional 1% iterations. Additional results are presented in Table 5.

8. Future work. This paper introduces erasure-coded linear system solvers, and provides a proof of concept of its performance and effectiveness. To further develop the theory and techniques for practically realizing this idea, we plan to explore the following theoretical and practical directions.

- *Parallel experiments with faults.* Our current simulation models the execution of a parallel CG algorithm with a fixed number of faults in MATLAB. We plan to design and test in a realistic environment, on real parallel systems, with either simulated or real-world faults. In particular, an interesting question is what happens to our method with a fault rate instead of a fixed number of faults. This will likely introduce some interesting relationships between the size of the system, the number of steps for convergence, and the fault rate of the distributed system.
- *Improved encoding matrices.* Our current design of the encoding matrix $\boldsymbol{E}$ uses the random Gaussian matrix, which is dense. This results in both an increase to the number of iterations (as mentioned in section 7) as well as an increase in per iteration time due to the density. To enhance both encoding and recovery efficiency, we plan to adopt structured and sparse random projections (Clarkson et al., 2013, Foucart and Rauhut, 2013, Meng and Mahoney, 2013), which allow fast matrix-vector multiplications.
- *Improved convergence theory.* Our current erasure-coded CG adopts the truncation or restart strategy when new faults occur. Our preliminary experiments indicate that such a strategy may result in slow and wavy convergence when there are large numbers of faults. We plan to adapt the flexible CG technique (Notay, 2000) to our erasure-coded CG solver in order to improve its convergence and to investigate the impact of preconditioning.
- *Formal conditioning theory.* Finally, there are a number of theoretical questions about the change in condition numbers that arise as a result of the investigation in section 7. First, we conjecture that for matrices with a large enough condition number, using a random Gaussian encoding matrix will enable us to bound the change in condition number of the encoded system at a value that is close to 2. Second, we plan to study the conditioning more carefully to see if the structure of the purified system or the structure of the encoded system give rise to a more refined conditioning analysis. This will include understanding how the conditioning changes *when* a fault occurs and how we expect the behavior of CG to proceed in this case. Third, we will also need to extend these results to the new, sparse, encoding matrices discussed above.
- *Whole application study.* This paper presents a concept which, at the moment, applies only to the linear solve step of an application pipeline. A key study in motivating the use of this idea in practice is a demonstration of improved performance in a whole-scale application setting for a representative scientific problems in computational fluid mechanics, structrual analysis, or other possible application domains. For instance, Gamell et al. (2015) consider fault tolerance in the context of the S3D combustion simulation. The whole application example would provide a setting to fairly compare our ideas against checkpointing, and to allow us to tune for both settings.

Acknowledgments. We thank Rob Schreiber for looking over an initial draft of this work and suggesting some important related work and also Ahmed Sameh for discussing the ideas with us. We also thank the anonymous referees for suggesting additional literature and the focus on conditioning.

REFERENCES

G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, *Algorithm-based fault tolerance applied to high performance computing*, J. Parallel Distrib. Comput., 69 (2009), pp. 410–416, https://doi.org/10.1016/j.jpdc.2008.12.002.

M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, *Checkpointing strategies for parallel jobs*, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, New York, 2011, 33, https://doi.org/10.1145/2063384.2063428.

P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen, *Fault-Tolerant Linear Solvers via Selective Reliability*, preprint, arXiv:1206.1390, 2012.

Z. Chen, *Optimal real number codes for fault tolerant matrix operations*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, 2009, 29, https://doi.org/10.1145/1654059.1654089.

Z. Chen, *Algorithm-based recovery for iterative methods without checkpointing*, in Proceedings of the 20th International Symposium on High Performance Distributed Computing, ACM, New York, 2011, pp. 73–84, https://doi.org/10.1145/1996130.1996142.

Z. Chen and J. Dongarra, *Numerically stable real number codes based on random matrices*, Computational Science, Lecture Notes in Comput. Sci. 3514, Springer, Berlin, 2005, pp. 115–122, https://doi.org/10.1007/11428831_15.

Z. Chen and J. Dongarra, *Algorithm-based fault tolerance for fail-stop failures*, IEEE Trans. Parallel Distrib. Systems, 19 (2008), pp. 1628–1641, https://doi.org/10.1109/TPDS.2008.58.

Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, *Fault tolerant high performance computing by a coding approach*, in Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, New York, 2005, pp. 213–223, https://doi.org/10.1145/1065944.1065973.

K. L. Clarkson, P. Drineas, M. Magdon-Ismail, M. W. Mahoney, X. Meng, and D. P. Woodruff, *The fast Cauchy transform and faster robust linear regression*, in SODA, Curran, Red Hook, NY, 2013, pp. 466–477.

T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1.

J. Dean and S. Ghemawat, *MapReduce: Simplified data processing on large clusters*, in Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI2004), USENIX, Berkeley, CA, 2004, pp. 137–150.

I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14, https://doi.org/10.1145/62038.62043.

J. Elliott, M. Hoemmen, and F. Mueller, *Resilience in Numerical Methods: A Position on Fault Models and Methodologies*, preprint, arXiv, cs.MS, pp. 1401.3013, 2014.

J. Elliott, M. Hoemmen, and F. Mueller, *A numerical soft fault model for iterative linear solvers*, in Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, 2015, pp. 271–274, https://doi.org/10.1145/2749246.2749254.

J. van den Eshof and G. L. G. Sleijpen, *Inexact Krylov subspace methods for linear systems*, SIAM J. Matrix Anal. Appl., 26 (2004), pp. 125–153.

S. Foucart and H. Rauhut, *A Mathematical Introduction to Compressive Sensing*, Birkhäuser, Basel, 2013.

M. Gamell, K. Teranishi, M. A. Heroux, J. Mayo, H. Kolla, J. Chen, and M. Parashar, *Local recovery and failure masking for stencil-based applications at extreme scales*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, New York, 2015, 70, https://doi.org/10.1145/2807591.2807672.

K.-H. Huang and J. A. Abraham, *Algorithm-based fault tolerance for matrix operations*, IEEE Trans. Comput., C-33 (1984), pp. 518–528, https://doi.org/10.1109/TC.1984.1676475.

I. C. F. Ipsen and C. D. Meyer, *The idea behind Krylov methods*, Amer. Math. Monthly, 105 (1998), pp. 889–899, https://doi.org/10.2307/2589281.

M. Kooper, H. van der Vorst, S. Poedts, and J. Goedbloed, *Application of the implicitly updated Arnoldi method with a complex shift-and-invert strategy in MHD*, J. Comput. Phys., 118 (1995), pp. 320–328, https://doi.org/10.1006/jcph.1995.1102.

J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95–138, https://doi.org/10.1016/0024-3795(77)90069-6.

X. Meng and M. W. Mahoney, *Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression*, in Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, ACM, New York, 2013, pp. 91–100.

G. Meurant, *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*, Software, Environ. Tools, SIAM, Philadelphia, 2006.

Y. Notay, *Flexible conjugate gradients*, SIAM J. Sci. Comput., 22 (2000), pp. 1444–1460.

Y. Saad, *A flexible inner-outer preconditioned GMRES altorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.

F. B. Schneider, *Implementing fault-tolerant services using the state machine approach: A tutorial*, ACM Comput. Surv., 22 (1990), pp. 299–319, https://doi.org/10.1145/98163.98167.

V. Simoncini and D. B. Szyld, *Flexible inner-outer Krylov subspace methods*, SIAM J. Numer. Anal., 40 (2003a), pp. 2219–2239.

V. Simoncini and D. B. Szyld, *Theory of inexact Krylov subspace methods and applications to scientific computing*, SIAM J. Sci. Comput., 25 (2003b), pp. 454–477.

J. J. Wilke, K. Teranishi, J. C. Bennett, H. Kolla, D. S. Hollman, and N. Slattengren, *Evolving the message passing programming model via a fault-tolerant, object-oriented transport layer*, in Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale, ACM, New York, 2015, pp. 41–46, https://doi.org/10.1145/2751504.2751513.