Algorithmic Techniques for Fault Detection for Sparse Linear Algebra

Joseph Sloan, Rakesh Kumar
University of Illinois Urbana-Champaign

Greg Bronevetsky, Tzanio Kolev
Lawrence Livermore National Laboratory

ABSTRACT

The growing complexity and variability of future computing systems is making it increasingly likely that individual circuits will produce erroneous results, especially when operated in a low energy modes. Previous techniques for Algorithm - Based Fault Tolerance (ABFT) [7] have been proposed for detecting errors in dense linear operations, but have high overhead in the context of sparse problems. In this paper, we propose a set of novel techniques that minimize the overhead of applying ABFT to sparse problems. The techniques are based on the insight that we can utilize sampling techniques to approximate the checks for sparse problems, as these problems are typically well-structured and sampling produces a good approximation of the problem's overall structure. Further, we propose algorithms for detecting errors in problems that have weak structure by first conditioning the problem to reduce the amount of variation within the problem, making it more amenable to approximation. These techniques are shown to yield up to $2x$ in performance improvements.

1. INTRODUCTION

This paper focuses on algorithmic techniques for low overhead fault detection for sparse linear algebra applications. Sparse linear algebra forms the core of a large class of high performance computing (HPC) applications such as linear solvers, differential equation solvers, and graph analysis problems [2, 6]. It also forms the core of a large number of emerging recognition, mining, and synthesis (RMS) applications [4]. Algorithmic approaches to fault detection eliminate the need for hardware and software overhead associated with protecting against variation induced timing errors, and allow for both power and energy to be reduced.

Some checksum-based approaches have been proposed previously for dense linear algebra [7]. For dense matrices the traditional Algorithm-Based Fault Tolerance (ABFT) [7] check is very efficient, requiring $O(n)$ time compared to $O(n^2)$ for the original multiplication. Unfortunately, these approaches cannot be used directly for sparse linear algebra problems as sparse linear algebra problems have a lower algorithmic time complexity than equivalent dense problems ($O(n)$). A direct use of the previously proposed checksum-based approaches can therefore result in high overheads for sparse linear algebra problems. (Section 4)

In this paper, we propose new algorithm approaches for low overhead checksum-based fault detection for sparse linear algebra based applications. The fault detection techniques rely on the fact that sparse applications typically have inherent structure within the data and computation itself. These structures may be exploited to improve the performance of traditional dense checks. Novel preconditioning techniques (Section 2) may also be used to improve the structure. Indeed, to the best of our knowledge, this paper is the first to address application-level fault tolerance in the context of general sparse linear algebra.

2. ALGORITHMIC FAULT TOLERANCE

Our first improvement to the traditional algorithm approach takes advantage of the fact that error checking does not always need to be exact. As such, the Approximate Random (AR) technique approximates the right-hand side of the check ($(1^T A)x$) by randomly sampling the dimensions of the problem. The sampled result is then scaled, so that the identity becomes $1^T(Ax) = (c^T A)x \cdot \frac{n}{s}$, where s out of n entries are sampled.

This check works best for matrices with fairly consistent column sums, meaning that a sample of the columns is a good approximation of the others. This is true of many real matrices. For example, Figure 1 shows the matrix *qpband* from the University of Florida Sparse Matrix Collection [3]. This matrix represents a canonical indefinite optimization problem and has a banded diagonal structure. The matrix structure is shown on the left and on the right we show the distribution of the column sums. These sums are very consistent, with a low variance of 1.6, meaning that a small number of samples will capture their distribution fairly precisely. Similar opportunities exist for the matrix *bcsstm37* in Figure 3, which represents a track ball stiffness matrix [3].

The accuracy of approximate random depends on the variance of the values in x, in addition to depending on the variance of the matrix columns. In the context of computational science, x typically corresponds to the state of a physical system. In that case it will have a fairly regular structure since different regions of the physical space will have similar states. However, in cases where the physical system is chaotic or x comes from a non-physical system, the technique will need to take the variance of x into account as the variance rises.

2.1 Approximate Clustering

When a matrix is too complex for AR to provide a good approximation, it is possible to improve accuracy by dividing the columns into clusters that have similar sums. In the Approximate Clustering (AC) approach, we do not sample uniformly but rather sample a few representatives from each cluster of columns to make sure that all column types are well-represented. Our clustering approach uses either a complete or partial pass of an agglomerative clustering algorithm depending on the properties of the data (e.g. a complete pass is used when the number of unique values is high). By selecting the subset of dimensions sampled to represent the variation of column sums in the matrix, we expect to more accurately predict the actual sum for matrices with more diverse column sums.

Figure 2 shows the matrix *msc00726*, representing a structural engineering problem from the Boeing test matrix group [3]. The distribution of the column sums for this matrix has a high variance ($> 1e3$) due to the subsets of values which are orders of magnitude away from the common range of values in the distribution. However, the histogram for this matrix shows that there is actually only a small set of unique column sums (3 and 20), which can then be used as clusters in the AC technique. Therefore, although the structure of the problem may not be well suited for random sampling, the pattern is dominant enough that we can cluster and then sample within the similar subgroups.

2.2 Identity Conditioning

Not all sparse problems have a dominant pattern, however. The matrix in Figure 4, *Oregon-1* is an example of this type of case, and represents an undirected graph based on the network included in a portion of the Internet. This problem has very high variance and an especially ill-defined distribution, which is not suited for clustering.

To solve this problem we can use the idea of pre-conditioning to transform the column sums of the original matrix A into more regular sums of an equivalent linear system. The idea is to choose the check vector c to be the solution of the identity equation $c^T A = 1^T$. If the identity equation is solved exactly we eliminate the effect of A on the predicted check entirely, and instead can simply replace it with $\sum x$, since $(c^T A)x = 1^T x = \sum x$

The vector c can be computed approximately by solving the least squares optimization problem $min\|A^T c - 1\|$. When executed until the residual reaches < 100 (typically 1-2 iterations), the algorithm provides a good approximation of the identity equation [1]. Inexact solutions to $c^T A = 1^T$ are valuable because they reduce the error of approximating $(c^T A)x$ with $\sum x$ by reducing the effect of matrix A on the accuracy of the check. This also makes it possible to use conditioning to improve the accuracy of sampling and clustering because the natural variance of A is dampened in the product $c^T A$, making a fixed sampling or clustering of c more representative than when it is done with 1.

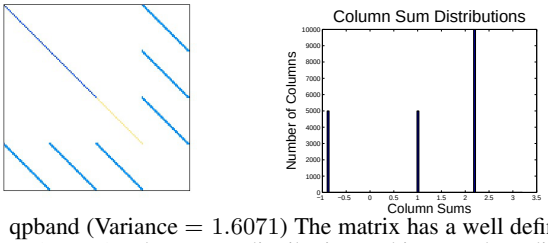

Figure 1: qpband (Variance $= 1.6071$) The matrix has a well defined and low variance($< 1e3$) column sum distribution and is a good candidate for both Approximate Random and Approximate Clustering.

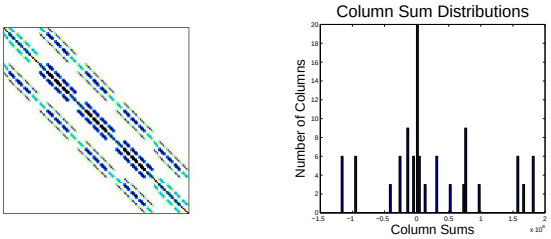

Figure 2: msc00726 (Variance $= 9.4724e14$) The matrix has high variance($> 1e3$) column sums. This matrix is a good candidate for clustering given the finite sets of unique values shown above.

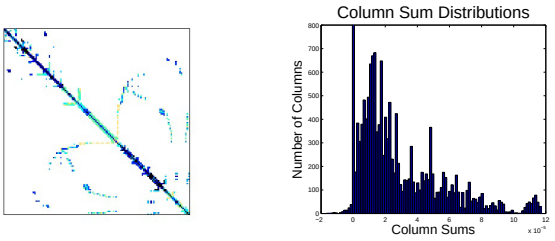

Figure 3: bcsstm37 (Variance$= = 6.1668e - 10$). The matrix has a well defined column distribution with low variance($< 1e3$) and is particularly well suited for Approximate Random Technique

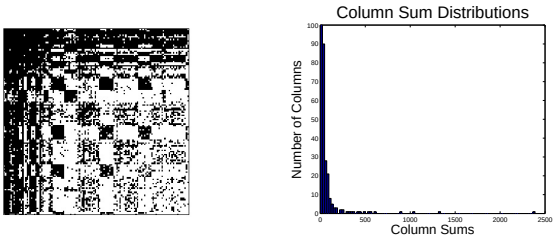

Figure 4: Oregon-1 (Variance $= 1065.5$). The matrix has column sums that are less well defined and have high variance($> 1e3$). Conditioning is a good candidate for this particular problem.

2.3 Null Conditioning

We can also eliminate the effect of A and x entirely by finding a vector in the null space of the problem. We call this *null conditioning*. We first find the the smallest singular value using singular value decomposition (SVD). If this value is sufficiently small (below $1e-6$ in our experiments), we use the associated vector as a null vector of the matrix in the check. The check only then requires verifying that the dot product of the output of the MV product is equal to zero: $A^T v = 0$ with $v \neq 0$. Therefore, the check is completed by verifying $v^T y = 0$, where $y = Ax$. We note that this check can be used for dense algebra problems as well. For square symmetric matrices, which are common in practice, SVD reduces to the eigenvalue problem. Our particular SVD implementation does not need to compute all the singular values, but instead only computes the smallest singular value and the associated vector within a relaxed accuracy target.

3. METHODOLOGY

There are various types of faults that can occur in computations. Our evaluation focuses on transient faults where the behavior of transistors and circuits is perturbed by various process and environmental variations. These faults corrupt the outputs of the circuits and may affect the results of numerical computations. Silent Data corruptions (SDCs) are typically the most difficult types of faults to detect, as the program will complete with incorrect results with no indication that the result is suboptimal or, in the worst case, unacceptable. Many emerging workloads are becoming increasingly data-centric [4], making the detection of faults in data increasingly important. Moreover, errors manifesting in control, such as memory corruption, deviations of control flow or memory access errors can typically be caught by using simple low overhead techniques [8, 9]. Section 4 evaluates the effect of computation faults within linear algebra operations, such as the MV product ($y = Ax$).

Additionally, along with considering alternative fault scenarios, we also consider different application contexts and inputs by using a set of real world matrices and vectors for publicly-available matrix libraries from the University of Florida Sparse Matrix Collection and Matrix Market [3, 5]. The set of matrices were chosen from different classes and sizes to understand the performance and accuracy of the proposed detectors across various structures present in real problems.

4. RESULTS AND ANALYSIS

This section discusses experimental results used to evaluate both the accuracy and performance of the approaches proposed in Section 2.

4.1 Performance Overhead

The traditional algorithmic techniques for fault tolerance are based on dense checks that result in high overheads when applied to sparse problems. Figure 5 shows the mean overhead for the dense check for MV when applied to a set of 50 real sparse matrices. The average overhead was around 30% but was as high as $80 - 90\%$ for matrices *Bcsstm39* and *t3dl_e*. The technique's overhead depends strongly on the size, sparsity, and locality of the problem. The sparsity of the matrix influences both the total number of operations and memory efficiency due to the increase in indirect memory accesses. The overhead rises as the problem becomes more sparse since this means that the original MV does less work while the cost of the check remains the same. Matrices *Bcsstm39* and *t3dl_e*, for which the overhead were the highest, are also the most sparse , with an average of about one nonzero entry per row.

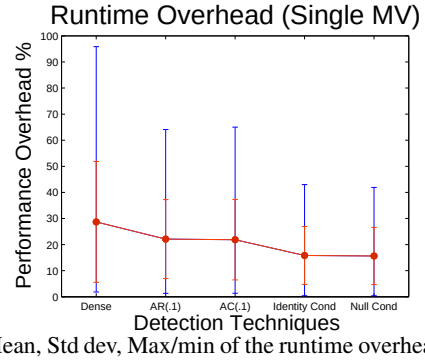

Figure 5: Mean, Std dev, Max/min of the runtime overhead of the check over single MV products for various sparse problems. Depending on sparsity, locality, and size of the problem, overheads can be high due to the additional computations and loss of memory locality.

The approximate random check (described in Section 2) exploits the structure of sparse problems by sampling the matrix column sums and x to capture the important features of the problem and predict the corresponding checksum. Figures 5 and 6 present the runtime overhead of approximate random over a set of 50 sparse problems. The overheads of approximate random correspond to a sampling rate of 10% (denoted $RS(.1)$). The data shows that the runtime overhead of the approximate random algorithm is lower than the dense check and has no additional setup cost.

The clustered sampling check is focused on problems that do not contain a well behaved and low variance distribution, but still have

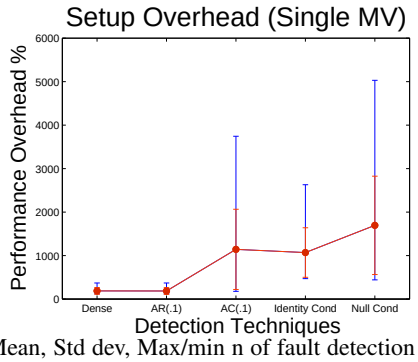

Figure 6: Mean, Std dev, Max/min n of fault detection setup overhead over single MV products of various sparse problems. Matrix reuse, which is common within iterative linear algebraic applications, allows for these overheads to be amortized over the entire application's execution.

structure within segments of the problem, as is commonly seen in sparse problems. It performs additional preprocessing to improve the accuracy of sampling. Figures 5 and 6 show that this check has the same runtime cost as random sampling when using the same 10% sampling rate (denoted $CS(.1)$) and has a higher setup cost. By trading off runtime overheads for this higher setup cost (roughly $100x$), the total overhead is actually still significantly reduced. This is because common linear solvers are iterative in nature and typically reuse the same matrix more than 10000 times over the application's execution. This amount of reuse more than amortizes the cost of setup for the sparse algorithmic detection techniques. (Section 4.3 discusses further).

In the case that the matrix does not have a clear local or global structure, identity conditioning can be used to create a more well defined distribution by smoothing the distribution of $c^T A$ to improve detection accuracy. The effectiveness of identity conditioning is related to the accuracy with which the identity equation $c^T A = 1^T$ is solved. For many problems we find that to detect moderate-magnitude errors it is only necessary to run the least squares algorithm for $1-3$ iterations.

Finally, null conditioning uses a null vector of the linear system or its lowest-frequency component to completely smooth the details of the matrix structure and input vector distribution to a single scalar value of zero. The smallest singular value and associated vector can also be found with relaxed accuracy, although the size of this value depends on the problem. Figures 5 and 6 show that although null conditioning has the highest setup cost, it also has the lowest runtime cost because there is no need to compute $(c^T A)x$ at all, since it is known to be very close to 0.

These results show that the proposed techniques can vary widely in their performance properties, and are also very dependent on the characteristics of the given sparse problem. This provides developers with a significant amount of power to choose the algorithm that minimizes overheads in their particular use-case, both in terms of the matrix structure as well as the reuse pattern of their algorithm.

4.2 Accuracy

This section reports the accuracy with which all our algorithms detect errors of different magnitudes. The accuracy of each fault detection technique, is evaluated by performing fault injection experiments. Each algorithm's ability to detect errors is reported in terms of false positive rate (FPR) and true positive rate (TPR). The TPR is defined as the percentage of experiments that correctly detect a fault when a fault is present. The FPR is defined as the percentage of experiments that incorrectly detect a fault when no faults are present. An ideal detector has both high TPR and low FPR for all meaningful errors arising in the system.

Each algorithm compares the computed check to a threshold, which then directly impacts the TPRs and FPRs seen. Figure 9 shows the results of fault injection experiments where the MV product was executed 1000 times on each matrix within the set of 50 matrices. In each run we used a random x and injected an error of magnitude ranging from $1e-20$ to $1e20$. For a given error magnitude, Figure 9 shows the average FPR associated with the threshold τ that produces a TPR of 95%.

The dense check has very good accuracy, showing no false positives at 95% TPR on errors as small as $\epsilon = 1$. In contrast, the sampling used by the approximate random check has reduced accuracy. It reaches 15% FPR for $\epsilon = 1e3$ and 0% FPR for $\epsilon = 1e12$ when using 10% sampling. The impact of the lower accuracy depends on the resilience needs of the algorithm that utilizes the MV operations. Further, it is possible to trade off accuracy and performance by using different sampling rates. At a sampling rate of 1% the runtime overhead is reduced by 50%, with an associated loss in accuracy.

Clustered sampling improves the accuracy of random sampling for all the matrices, with a 1-2 order of magnitude improvement in the smallest magnitude of error which is detected with small a small FPR ($< 1\%$) and high TPR. In general,the benefits of clustered sampling over random sampling were the highest for matrices that have high variance.

The Identity and Null conditioning techniques both can significantly reduce the smallest detectable error magnitude by an additional 1-2 orders of magnitude. Identity conditioning works particularly well with problems which have relatively low condition numbers, as these problems are able to approximately solve the identity equation much more easily. For example, identity conditioning worked exceedingly well for *fv, shallow_water,* and *qpband*. In the case of *qpband*, the smallest magnitude error that is detected with high probability is reduced from $1e3$ to $1e-7$, compared to the AR technique.

On average, the null conditioning technique performed worse then the other techniques, as the majority of matrices used in our experiments contained structurally full ranks (meaning the upper bound on their rank is full). Therefore, the null-space of these types of problems, with full rank, were limited to primarily zero vectors, which are not well suited for detection. We do note, however, that null conditioning is very useful for the subset of matrices that have a smallest singular values (SSV) less than about $1e-6$. For example, null conditioning is able to detect errors which are $1-2$ orders of magnitude smaller than those detected by random sampling, for the matrix *Dubcova1* (SSV= .00481) .

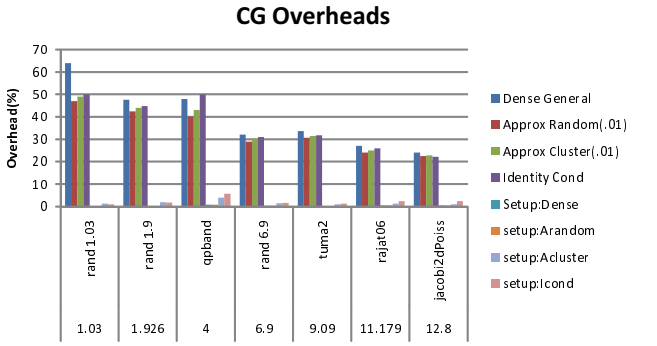

Figure 7: Average overhead for the proposed techniques for CG. The matrices are organized according to sparsity (sparsity values are listed below the graph)

The proposed sparse techniques are shown to have significant average runtime performance improvements over the traditional dense check in the context of a single MV product, while maintaining similar high detection accuracy under moderate and large error magnitudes.

4.3 Discussion: Linear Solver Applications

In this section, the performance of the fault detection techniques are considered within the context of one common linear solver application, the Conjugate Gradient (CG) algorithm. Our proposed approaches are applied to CG, in order to evaluate the performance benefits in the context of a more complex linear algebra application. With sparse matrices, the vector and dot product operations within CG can account for nearly half of the operations used each iteration. We therefore also protect these operations by using similar dense checks.

Overhead for the setup plus the total overall overhead are shown in the Figure 7. These overheads are averaged over 100 runs. Each

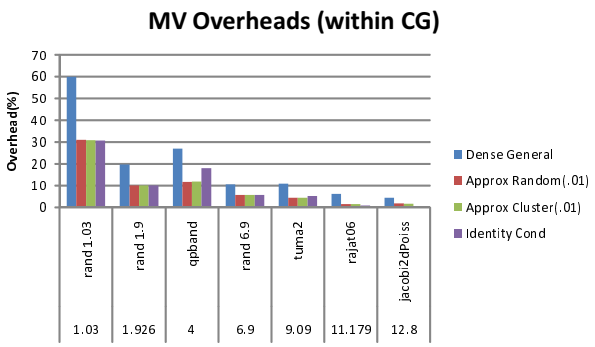

Figure 8: Average overheads when considering only the MV products in CG. The matrices are organized according to sparsity (sparsity values are listed below the graph)

matrix on the bottom is labeled with its corresponding sparsity factor (s =average # of non-zeros per row). The matrices are ordered in according to the the degree of sparsity. Moving from left to right, the matrices are less sparse.

Figure 7 illustrates how common linear algebra application typically exhibit large amounts of reuse. The setup overhead of computing the column sums is negligible ($< 1\%$), because of this large amount of reuse. The overhead of the dense check, in the context of CG, is also shown to result in large performance overheads ($30-50\%$) when using sparse matrices.

The reduction in total overhead is smaller (5 to10%) compared to an application which only uses MV products (Power series, Eigen solver), because much of the time is also spent on linear vector operations and nonlinear dot products. Extending the sparse techniques for approximating the dot product operation is the subject of future work. The reduction in overhead can be nearly cut in half when only the time spent computing MV products is considered (Figure 8).

5. CONCLUSIONS

The ability to detect and correct application faults for future error prone and energy constrained computing systems is of critical importance. Our paper focuses on low overhead fault detection for sparse linear algebra algorithms which represents the core of a large class of emerging applications.

In this paper, we showed that previous checksum-based approaches for linear algebra fault detection in the context of dense problems may have high overheads for sparse problems (30% to 90%). We presented approaches that reduce the overhead of fault detection for sparse linear algebra-based problems by up to $2x$ by exploiting structure in the problems. Proposed approaches worked for both problems that are relatively well distributed and of low variance (through random and clustering based sampling) as well as for problems that have less well defined characteristics (through identity and null conditioning).

6. REFERENCES

[1] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: a portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, Supercomputing '90, pages 2–11, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[2] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28:135–151, 2001.

[3] Timothy A. Davis. University of florida sparse matrix collection. *NA Digest*, 92, 1994.

[4] Asanovic et. al. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[5] Ronald F. Boisvert et. al. Matrix market: A web resource for test matrix collections. In *The Quality of Numerical Software: Assessment and Enhancement*, pages 125–137. Chapman and Hall, 1997.

[6] Robert Falgout and Ulrike Yang. hypre : A library of high performance preconditioners. In *Computational Science, ICCS 2002*, volume 2331 of *Lecture Notes in Computer Science*, pages 632–641. Springer Berlin, Heidelberg, 2002.

[7] Kuang-Hua Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, C-33(6):518 –528, 1984.

[8] Man lap Li, Pradeep Ramach, Swarup K. Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Swat: An error resilient system, 2008.

[9] Nahmsuk Oh, Philip P. Shirvani, and Edward J. McCluskey. Control-flow checking by software signatures. *IEEE Transactions on Reliability*, 51:111–122, 2002.

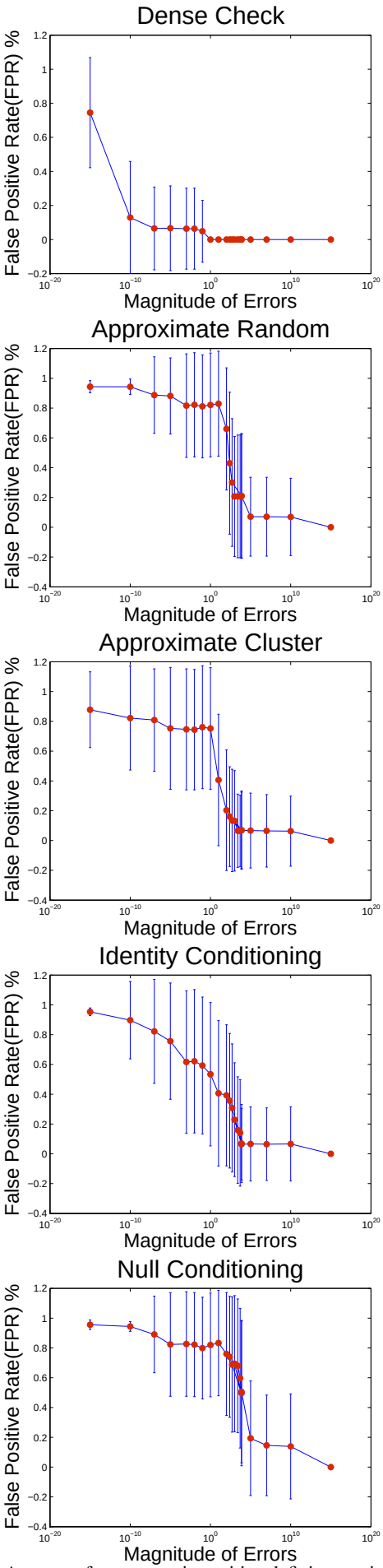

Figure 9: Accuracy for symmetric positive definite matrices. Both the mean and variance of the measurements are shown. Note: The x-axis is log scale from $1e-15$ to $1e15$.