

# Algorithm Based Fault Tolerant State Estimation of Power Systems

Amitabh Mishra, Lamine Mili, and Arun G. Phadke

**Abstract** - Electric power industry deregulation has transformed state estimation from an important application into a critical one. State estimations in Power systems involve very tedious computations since large systems can consist of thousands of buses and lines. In this paper, we apply Algorithm Based Fault Tolerance techniques to the Gauss-Newton iterative algorithm that solves a weighted least-squares (WLS) state estimator of a power system. These techniques efficiently detect computational errors due to transient and permanent faults in the hardware. We show that the modified state estimation algorithm has low computational over-head and is free from false alarms.

**Keywords**- Algorithm based fault tolerance, state estimation, Gaussian elimination, error detection, power systems.

## I. INTRODUCTION

Electric power industry deregulation has transformed state estimation from an important application into a critical one. State estimations in power systems involve very tedious computations since large systems can consist of thousands of buses and lines [1-3]. However, computations of that magnitude are always prone to errors caused by transient or permanent failures in hardware and also software failures to certain extents (buffer overflows, etc). Computationally efficient and accurate results are essential, since power system security monitoring and analysis hinge on them.

Algorithm Based Fault Tolerance (ABFT) is a cost effective technique to detect and correct errors caused by permanent or transient failures in the hardware. ABFT is based on encoding the data at the system level in the form of some error correcting or error detecting code and then designing algorithms to operate on encoded input data to produce encoded output data. One of the key advantage of ABFT which makes it very attractive is that it does not require any modifications to the underlying hardware to provide fault tolerance. This technique has already been applied to numerical algorithms e.g. matrix multiplications, LU factorizations etc. [4, 5]; the fast Fourier transforms [6, 7], QR factorization [8], singular value

decomposition [9], Laplace equation [14], and the multigrid method [15], [16] etc. running on array processors. Later the same set of problems were resolved using parallel algorithms [10, 11] running on general-purpose multiprocessors.

The ABFT technique is known as ABED (algorithm based error detection) when applied only for the detection of errors. The ABED technique provides fault tolerance by preserving an invariant before the start and at the end of all computations in an algorithm. It can also preserve an invariant during the computations by slightly modifying the algorithm. Lack of preservation of the invariant during the course of the algorithm signals presence of errors in the computation due to faults. Since most of these computations are performed on computers that have finite precision, sometimes invariants are not preserved because of roundoff errors in the computation of the invariant and in the algorithm, and the ABFT technique can signal presence of faults even in the absence of any actual hardware faults. This situation is known as false alarms. Invariant computations, therefore should be corrected for roundoff errors to eliminate or minimize the false alarms by using some kind of tolerance.

It has been shown that methods which compute tolerance using error analysis of the algebraic manipulations in the algorithm are superior to other methods [12]. The error analysis provides upper bounds of roundoff errors which can occur in the computation of an expression on a machine with finite precision. The invariants when modified by adding the computed value of the roundoff error accumulated in an algorithm totally solves the problem of false alarms for any data set of any magnitude or dimension.

The solution of a large set of the order of several thousand of simultaneous algebraic equations is still an important issue in numerical linear algebra. Such large systems of equations often arise in load flow studies and the state estimation in the power system as well as when a partial differential equation modeling electromagnetic field, heat conduction or fluid flow is discretized using finite differences or finite elements. Typical methods for solving system of linear algebraic equations are either direct or iterative. Gaussian elimination and all its variants fall under the category of direct methods. A direct method computes the solution to a finite precision in a finite number of operations. Operation counts for Gaussian elimination vary from  $O(N^3)$  to  $O(N^2 \log N)$  for  $N$  unknowns depending upon the properties of the coefficient matrix and how the sparsity structure of the matrix is exploited. The minimum operation count that can be achieved

Amitabh Mishra, Lamine Mili and Arun G. Phadke are with the Department of Electrical and Computer Engineering, Virginia Tech, VA 24061-111, (e-mails: [mishra@vt.edu](mailto:mishra@vt.edu), [lmili@vt.edu](mailto:lmili@vt.edu), [aphadke@vt.edu](mailto:aphadke@vt.edu)).

$O(N^2)$  is for the minimum degree algorithm, which is nearly optimal [13].

This paper presents an application of ABFT on a coefficient matrix that has been obtained as a result of state estimation algorithm of a power system that is solved using Gaussian elimination. We use error analysis to derive expressions for the upper bound on roundoff errors in the Gaussian elimination algorithm, which are then used to correct the invariants that are used in the checking step. We show that the modified Gaussian elimination algorithm has low computational over-head and is free from false alarms.

The paper is organized as follows. Section II contains a brief description of the estate estimation algorithm that we have considered for the decomposition using Gaussian elimination. Section III presents an ABFT encoded Gaussian elimination including the expressions for the row and column checksums that are used as invariants in the checking step and the derivations of expressions for the round off errors accumulated in the elements of the decomposed matrix .

## II. WLS STATE ESTIMATION

The role of static state estimator is to provide a complete, coherent, and reliable base-case power flow solution for contingency analysis, and load forecasting function, among others [1, 2]. The state estimator processes a redundant collection of measurements based on a mathematical model that relates these measurements to the nodal voltage magnitudes and phase angles, which are taken as the state variables of the system. This model is derived from Kirchhoff's and Ohm's laws and hinges on several assumptions, which include: (1) random measurement errors that are Gaussian and uncorrelated with zero mean and known diagonal covariance matrix,  $R = \text{diag}(\sigma_i^2)$ ; (2) a balanced three-phase system, (3) no time-skew between the metered values, (4) the exactness of the pi-equivalent models of the lines and transformers of their parameters, and (5) a known topology of the network. The latter is determined by a topology processor from the metered or given status of the circuit breakers of the system equipments such as lines, transformers, capacitors and inductors, FACTS devices, to name a few.

A general state estimation model is given by

$$z = h(x) + e, \quad (1)$$

where  $z$  is the measurement vector,  $e$  is the measurement error vector, and  $h(x)$  is a vector-real value function. The latter is derived from the foregoing assumptions; in particular, it depends on the given values for the line and transformer parameters and the given network topology. Many commercial software packages seeks a solution to the WLS normal equation,

$$H^T R^{-1} (z - h(x)) = 0, \quad (2)$$

through the Gauss-Newton iterative algorithm [3] expressed as

$$(H^T R^{-1} H) \Delta x^{(k)} = H^T R^{-1} \Delta z^{(k)} \quad (3)$$

where  $H = \partial h(x)/\partial x$  is the Jacobian matrix,  $\Delta z^{(k)} = z - h(x^{(k)})$  is the residual vector, and  $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$  is the increment of the state vector at the  $k$ th iteration step. The error covariance matrix of the state estimates,  $\text{Cov}(\delta_x) = \Sigma_x$ , and of the estimates associated with the measurements,  $\text{Cov}(\delta_z) = \Sigma_z$ , are respectively given by

$$\text{Cov}(\delta_x) = \Sigma_x = (H^T R^{-1} H)^{-1} \quad (4)$$

$$\text{Cov}(\delta_z) = \Sigma_z = H (H^T R^{-1} H)^{-1} H^T \quad (5)$$

The diagonal elements of  $\Sigma_x$  and  $\Sigma_z$  are the variances of the estimate errors. Under the assumptions stated earlier and under the validity of the linear approximation of the model given by (2) around the solution, these variances provide a measure of the accuracy of the estimates; the smaller they are, the more accurate the state estimator solution is.

## III. ABFT ENCODED GAUSSIAN ELIMINATION

The solution of a set of linear algebraic equations of the form  $Ax = b$  for a single right hand side  $b$  is usually obtained by performing Gaussian elimination to create an upper triangular matrix. The system is then easily solved by back substitution from the last unknown upwards. Here matrix  $A = (H^T R^{-1} H)$  given by equation (3). The basic algorithm (incorporating no error detection features) proceeds as follows: For a system of size  $m$  there are  $m-1$  iterations in the algorithm from 1 to  $m-1$ , during each of which one more column of 0's is introduced into matrix  $A$ . Iteration  $k$  of the algorithm results in the update of an  $(n-k) \times (n-k)$  sub-matrix. The update step of the algorithm can be given as

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{kj}^{(k)} \times a_{ik}^{(k)} / a_{kk}^{(k)}, \quad k < i \leq m, k < j \leq m \quad (6)$$

where  $a_{ij}^{(k)}$  represents the element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of  $A$  at the start of the  $k^{\text{th}}$  iteration. Error detection capability is introduced into the algorithm by the introduction of row and column checksums which are described below. Let us denote the  $i^{\text{th}}$  row of  $A$  by  $row_i$ . We also denote the column checksum of the  $j^{\text{th}}$  column of first  $k-1$  rows at the start of the  $k^{\text{th}}$  iteration by  $CCA_j^{(k)}$  and the column checksum of the remaining rows by  $CCB_j^{(k)}$ . We denote the row checksum of row  $i$  at the start of the  $k^{\text{th}}$  iteration by  $RC_i^{(k)}$ . At the start of the  $k^{\text{th}}$  iteration of the algorithm, assuming error-free operation of the algorithm for the first  $k-1$  steps, the following invariants are maintained:

I1 –  $RC_i^{(k)}$  for all rows;

I2 –  $CCA_j^{(k)}$  and  $CCB_j^{(k)}$  for all columns.

The algorithm computes the row checksum of all the rows and partial column checksums. At the start of the first iteration invariants I1 and I2 are correctly maintained for each row and each column. Then at the start of the  $k^{th}$  iteration we first update the tolerances for the row and column checksums for the current iteration using the expressions derived in the next section. We next perform row and column checksum checks on the  $k^{th}$  row and column of the results of the previous iteration. Since  $a_{ik}^{(k)}$  is known for all rows the column checksum computation for column  $k$  can be easily computed. Then a row checksum check on row  $k$  is performed. If both the row checksum and the column checksums pass, it means there is no error in  $k^{th}$  row and  $k^{th}$  column. The row checksums are updated in the following manner:

$$RC_i^{(k+1)} = RC_i^{(k)} - a_{ik}^{(k)} \times RC_k^{(k)} / a_{kk}^{(k)}, \quad \forall i > k \quad (7)$$

and the column checksums  $CCA$  and  $CCB$  as:

$$CCA_j^{(k+1)} = CCA_j^{(k)} + a_{kj}^{(k)}, \quad \forall j \leq k \quad (8)$$

$$CCB_j^{(k+1)} = CCB_j^{(k+1)} - a_{kj}^{(k)} \times CCB_k^{(k)} / a_{kk}^{(k)}, \quad \forall j \leq k \quad (9)$$

We then proceed to update its rows of  $(n-k) \times (n-k)$  sub-matrix using equation (6). If no errors are introduced in the  $k^{th}$  iteration, the start of  $(k+1)^{th}$  iteration results in the preservation of I1 and I2.

### A. Error Analysis

In the following, we shall provide the derivations of the error expressions for the Gaussian elimination algorithm which is described by equation (6). We denote by notation  $|err(a)|$ , the upper bound on the absolute error associated with a variable  $a$  that results due to a floating point operation on  $a$ . We can write therefore for a variable  $v$ ,

$$|\bar{v}| \leq |v| + |err(v)| \quad (10)$$

knowing that any floating point operation  $z = fl(x \oplus y)$  results in an error  $err(z) = (x + y)\delta$ , where  $|\delta| \leq \varepsilon = 2^{-t}$  for a  $t$  digit mantissa [11].

In these derivations the variables  $LE(x)$  and  $GE(x)$  denote the local error resulting from the last computation of a quantity  $x$ , and the global error or the accumulated error over all steps of computations respectively. The variable  $EB(x)$  is used to store an upper bound on the absolute value of the global error in  $x$ . So we have  $EB(x) \geq GE(x)$ . In the derivations that follow,  $O(\varepsilon^2)$  terms are dropped from the inequalities wherever they arise.

Equation (6) leads us to the algebraic expression which we need to analyze for round off errors for the Gaussian elimination and has the form:

$$z^* = fl(v^* - w^* \times x^* / y^*) \quad (11)$$

where,  $v^* = v + GE(v), \dots, z^* = z + GE(z)$ . Then we have

$$\begin{aligned} |GE(z)| &\leq |xGE(w)/y| + |wGE(x)/y| + \\ &|v|\varepsilon + 3|wx/y|\varepsilon + |wxGE(y)/y^2| + |GE(v)| \end{aligned} \quad (12)$$

**Proof:** Let's denote:

- (i)  $t_1 = fl(w^* x^*)$
- (ii)  $t_2 = fl(t_1 / y^*)$
- (iii)  $t_3 = fl(v^* - t_2)$

Let's simplify case (i) first.

$$\begin{aligned} t_1 &= fl(w^* x^*) = fl((w + GE(w))(x + GE(x))) \\ &= (w + GE(w))(x + GE(x))(1 + \delta_1) \\ &= wx + xGE(w) + wGE(x) + GE(w)GE(x) + \delta_1(wx + xGE(w) \\ &\quad + wGE(x) + GE(w)GE(x)) \\ &= wx + xGE(w) + wGE(x) + \delta_1 wx \end{aligned}$$

upon ignoring the second order terms. Similar simplification of (ii) and (iii) yields:

$$t_2 = \left(\frac{1}{y}\right)(wx + xGE(w) + wGE(x) + \delta_1 wx - \frac{wx}{y}GE(y) + \delta_2 wx)$$

and

$$\begin{aligned} z^* &= z + GE(z) = fl(v^* - t_2) = (v + GE(v) - t_2)(1 + \delta_3) \\ &= v + GE(v) - \frac{1}{y}(wx + xGE(w) + wGE(x) + \delta_1 wx - \frac{wx}{y}GE(y) \\ &\quad + \delta_2 wx) + v\delta_3 + \frac{wx\delta_3}{y} \end{aligned}$$

Since  $z = v - wx/y$ , we can further simplify the expression for  $z^*$  and write it as

$$\begin{aligned} GE(z) &= -x \frac{GE(w)}{y} - w \frac{GE(x)}{y} + v\delta_3 - \frac{wx}{y}(\delta_1 + \delta_2 + \delta_3) + \\ &\frac{wxGE(y)}{y^2} + GE(v) \end{aligned}$$

Since  $|\delta_i| < \varepsilon$ , the last expression can be simplified to

$$\begin{aligned} |GE(z)| &= \left| \frac{xGE(w)}{y} \right| + \left| \frac{wGE(x)}{y} \right| + |v|\varepsilon + \left| \frac{3wx\varepsilon}{y} \right| + \left| \frac{wxGE(y)}{y^2} \right| \\ &+ |GE(v)| \end{aligned}$$

which is identical to (12).

Using equation (12) and (6) we can show that the recurrence for the upper bound for error in an individual element holds and is given by (13) below.

$$EB(a_{ij}^{(k+1)}) \leftarrow (|a_{ij}^{(k)}| + 3|a_{ik}^{(k)}a_{kj}^{(k)} / a_{kk}^{(k)}|)\varepsilon + EB(a_{ij}^{(k)}) + |a_{ik}^{(k)}EB(a_{kj}^{(k)}) / a_{kk}^{(k)}| + |a_{kj}^{(k)}EB(a_{ik}^{(k)}) / a_{kk}^{(k)}| + |a_{ik}^{(k)}a_{kj}^{(k)}EB(a_{kk}^{(k)}) / (a_{kk}^{(k)})^2| \quad (13)$$

Equation (13) can be further simplified to

$$EB(a^{(k+1)}) \leftarrow 4|a_{kk}^{(k)}|\varepsilon + EB(a_{kk}^{(k)}) \quad (14)$$

which is arrived at by using the simplification  $|a_{kk}^{(k)}| \geq |a_{ij}^{(k)}|$  in equation (13). Using equation (14) and the fact that  $n-k$  elements are summed in the  $k^{th}$  iteration the following approximate error expression for the column checksums is obtained

$$EB(CCB1^{(k+1)}) \leftarrow (n-k)EB(a^{(k+1)}) \quad (15)$$

Similarly one can derive the error bound expression for the row checksum using the procedure described above and it's given by equation (16).

$$EB(RC_i^{(1)}) \leftarrow \left(\sum_{j=1}^n (n+1-i)|a_{ij}|\right)\varepsilon, \forall i \quad (16)$$

These error expressions are used to compute error bounds which are then added to the invariants to protect the algorithm from reporting false alarms.

#### IV. SIMULTATION RESULTS

The error coverage results for the ABFT encoded Gaussian elimination method described in the previous section are obtained by fixing the range and the size of the data sets and then varying one parameter while fixing the others as constants. The results that are reported here are the error coverage, significant error coverage for significance levels of 2 and 10 and the error acceptance level to display the robustness of algorithm. Here significance level  $\gamma$  is defined as the computation error in which the norm of deviation of the faulty result from the exact result is bounded by  $\gamma$  times the norm of deviation of fault-free results from exact results. The error acceptance level (EAL) of a test is defined as the smallest value of the significance  $\gamma$  which results in 100% error detection by the test. These comparisons are performed for transient bit-level floating addition errors, which are expected to have the lowest coverage of all classes of errors. We report results on timing overhead of this scheme as compared to the basic Gaussian elimination algorithm with no checks. We finally report the error latency for this algorithm for transient bit-level floating-point errors, since these are expected to be the hardest to detect.

#### A. Error Coverage

Tables 1 and 2 contain the error coverage results for the Gaussian elimination algorithm. Both tables contain the following columns representing the percentage computed values: The FA (False Alarm), EC (Error Coverage), SEC (2) (Significant Error Coverage – Significance Level 2) and SEC (10) (Significant Error Coverage – Significance Level – 10). The tolerance values used in these results came from the error analysis that was presented earlier. Table 1 shows the error coverage results with respect to the data range. The error coverage varies from 80-91% for a data range of 0.1 to 10000 respectively. This low variation suggests that the scheme is very robust against the data range variation.

**Table 1:** Percentage Error coverage on varying range of data for transient bit-level errors in floating point additions.

Data Range	FA	EC	SEC(2)	SEC(10)	EAL
0.1	0	80	87	97	17.05
1	0	79	90	98	8.17
10	0	84	91	99	5.37
100	0	86	93	100	3.73
1000	0	90	94	99	5.60
10000	0	91	96	100	2.38

**Table 2:** Percentage Error coverage on varying data size for transient bit level errors in floating point additions.

Matrix Size	FA	EC	SEC (2)	SEC(10)	EAL
30	0	84	91	100	3.58
60	0	86	93	100	3.73
100	0	86	97	100	2.51
125	0	84	94	99	2.50
250	0	84	90	99	2.30
500	0	84	90	99	2.25

Table 2 shows the variation of error coverage with the change in the size of the data set, other parameters being held constant. The error coverage and error acceptance levels do not show much variation with data set size either.

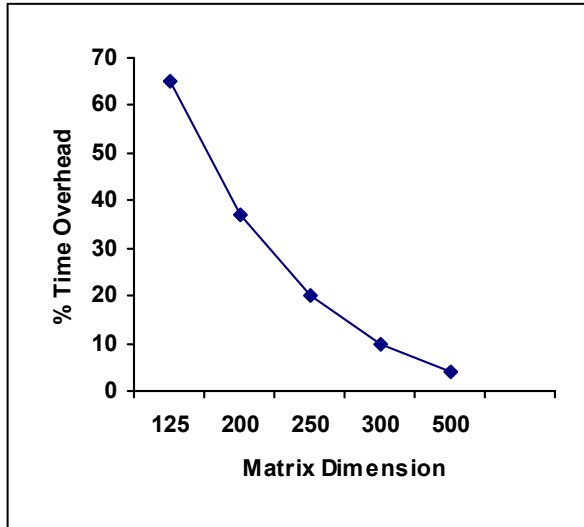
From the results contained in Tables 1 and 2 its clear that the scheme is robust which achieves high error coverage and low error acceptance levels across data sets of widely varying ranges and sizes. Significant error coverage levels are in the range of 87-96. Error acceptance levels are around 20 in the worst case, while error acceptance levels of around 10 are common. Please note that an error acceptance level of 100 would correspond to an error of a fraction of 1% in the final results for almost all applications. The scheme presented in the paper is not observed to give any false alarms as seen from column 2 of Tables 1 and 2. Note that the coverage results for memory errors as well as truly gross errors such as word level errors and control flow errors were found to be always 100% but these are not reported here due to the length of the paper considerations. These will be presented in a paper in the near future.

### B. Timing Results

For any proposed error detection algorithm, it's desired that the error detection latency be as small as possible. The error latency is defined as the time taken between the appearance of an error and its detection. One is interested in devising error detection schemes with low error latency in order to minimize data corruption.

The error latency is calculated in two ways. First, error latency is reported as the number of checks which are passed without the error being detected after the injection of the error. This is achieved by setting a flag when the error is first injected in a run and incrementing a counter every time a check is passed thereafter with the error not being detected. An error may be detected in subsequent checks even if it was not detected in the first check following its insertion due to error magnification, propagation, or due to the occurrence of additional errors. Second, error latency is also reported as a percentage of a total runtime.

From Figure 1 we note that overhead incurred in introducing checks into basic algorithm can be amortized by going to larger matrix sizes. One can see from these figures that the overhead decreases with the increase in the problem size. This is not unexpected since the check steps involve  $O(N^2)$  operations while the basic algorithm required  $O(N^3)$  operations for a system of size  $N$ . Thus, the additional overhead caused by introduction of the check steps is not of concern for large problem sizes.

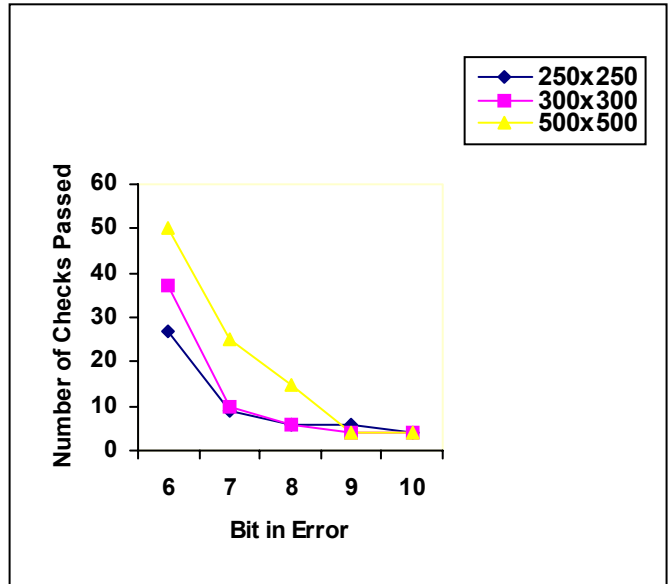


**Figure 1:** Time Overhead of the current scheme over algorithm with no checks.

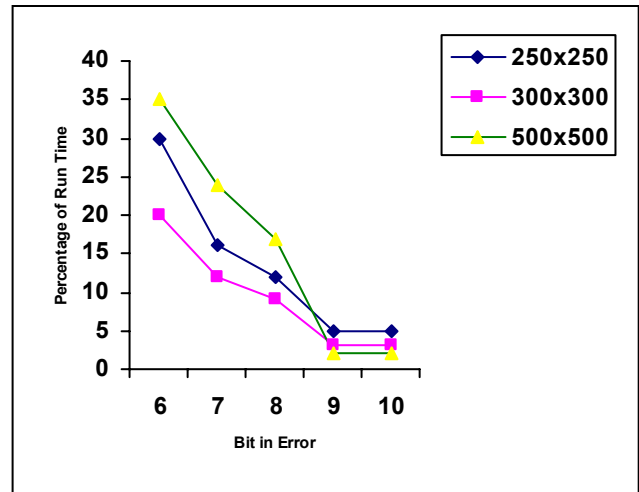
### C. Error Latency

Figures 2 and 3 indicate the error latencies for the Gaussian elimination application for transient bit-level floating-point errors for various bits in error starting with the first bit whose error was consistently detected. As the erroneous bit position becomes more and more significant, the error latency decreases until beyond a certain bit, all bit level errors are

detected in the first check following the error. For larger problem sizes, error latency for errors in more significant bits corresponds to a smaller percentage of the total run-time since the error is detected at the first check following its occurrence. Since there is a check following each iteration, and for larger problem sizes the execution time for each iteration is a smaller fraction of the total runtime, the error latency also decreases with matrix size for errors in more significant bits. But, this need not be true for errors in less significant bits since it was often observed that the latency in terms of number of checks passed following error injection tended to decrease with problem size. However, the figures indicate that errors in more significant bits were detected in the first check following error injection and the latency amounted to no more than 5% of the total runtime. Error latency studies for word level floating point and all other classes of errors injected were also conducted and in all of our simulation runs, we found that these errors were detected at the very next check following their injection.



**Figure 2:** Error latency as number of checks passed before detection



**Figure 3:** Error Latency as percentage of total runtime

## V. CONCLUSIONS

In this paper the algorithm based error detection scheme has been applied to the Gaussian elimination method for solving a WLS power system state estimator. We have derived the error expressions for the round off errors in the algebraic processes within the elimination procedure and modified the invariants for checking by taking into account the accumulated round off errors. The Gaussian elimination algorithm with checking was implemented on a SUN Enterprise server for experimental evaluation. Our results suggest that the algorithm provides error detection at low costs (less than 5% for 500 unknowns), excellent error coverage (98-100%) for floating point arithmetic in the presence of permanent bit and word errors. Moreover, the implemented algorithm is insensitive to the data range and the data size and reports no false alarms.

## VI. REFERENCES

- [1] A. Bose and K. Clements, "Real-time Modeling of Power Network," *Proceedings of the IEEE*, Vol. 75, pp. 1607-1622, Dec. 1987.
- [2] A. Monticelli. State Estimation in Electric Power Systems-A Generalized Approach. *Kluwer Academic Publishers*, 1999.
- [3] E. Hanschin, F. Schweppe, J. Kohlas, and A. Flechter, "Bad Data Analysis for Power System State Estimation," *IEEE Trans. on Power Apparatus and Systems*, Vol. PAS-94, No. 2, pp. 329-337, March/April 1975.
- [4] K. H. Huang and J. A. Abraham, "Algorithm Based Fault Tolerance for Matrix Operations", *IEEE Trans. Comput.*, Vol. C-33, pp. 518-528, June 1984 .
- [5] J. Y. Jou and J. A. Abraham, "Fault Tolerant Matrix Operations on Multiple Processor Systems using Weighted Checksum", *SPIE Proceedings*, Vol. 495, August 1984.
- [6] Y. -H. Choi and M. Malek, "A Fault Tolerant FFT Processor", *IEEE Trans. Comput.*, vol. 37, pp. 617-621, May 1988.
- [7] J. Y. Jou and J. A. Abraham, "Fault Tolerant FFT Networks", *IEEE Trans. Comput.*, vol. 37, pp.548-561, May 1988.
- [8] A. L. N. Reddy and P. Banerjee, "Algorithm-based Fault Detection for Signal Processing Applications", *IEEE Trans. Comput.*, Vol. 39, pp. 1304-1308, October 1990.
- [9] C. -Y. Chen and J. A. Abraham, "Fault-tolerant Systems for the Computation of Eigenvalues and Singular Values", *Proc. SPIE Conf.*, pp. 228-237, Aug. 1986.
- [10] V. Balasubramaniam and P. Banerjee, "Tradeoffs in the Design of Efficient Algorithm Based Error Detection Schemes for Hypercube Multiprocessors", *IEEE Trans. Softw. Eng.*, Vol. 16, pp. 183-194, Feb. 1990.
- [11] P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair, K. Roy, V. Balasubramaniam, and J. A. Abraham, "Algorithm-based Fault Tolerance on a Hypercube Multiprocessor", *IEEE Trans. Comput.*, Vol. 39, pp. 1132-1145, Sept. 1990.
- [12] A. Roy-Chowdhury and P. Banerjee, "Tolerance determination for algorithm based checks using simplified error analysis", *Proc. FTCS -93*, June 1993, Toulouse, France.
- [13] G. H. Golub and C. F. V. Loan, "Matrix Computations", *John Hopkins University Press*, Baltimore, 1987.
- [14] A. Roy-Chowdhury, N. Bellas, and P. Banerjee, "Algorithm based Error Detection Schemes for Iterative Solution of Partial Differential Equations", *IEEE Transactions on Computers*, Vol. 45, pp. 394-407, April 1996
- [15] A. Mishra and P. Banerjee, "An Algorithm based Error Detection Schemes for the Multigrid Algorithm", *FTCS-29*, Madison, pp. 12-19, 1999
- [16] A. Mishra and P. Banerjee, "An Algorithm based Error Detection Scheme for the Multigrid Method", *IEEE Transaction on Computers*, Vol. 52, No. 9, pp. 1089-1099, September 2003.