Linear Complexity Assertions for Sorting

Nirmal R. Saxena, *Member, IEEE*, and Edward J. McCluskey, *Fellow, IEEE*

***Abstract*— Correctness of the execution of sorting programs can be checked by two assertions: the order assertion and the permutation assertion. The order assertion checks if the sorted data is in ascending or descending order. The permutation assertion checks if the output data produced by sorting is a permutation of the original input data. Permutation and order assertions are sufficient for the detection of errors in the execution of sorting programs; however, in terms of execution time these assertions cost the same as sorting programs. An assertion, called the order-sum assertion, that has lower execution cost than sorting programs is derived from permutation and order assertions. The reduction in cost is achieved at the expense of incomplete checking. Some metrics are derived to quantify the effectiveness of order-sum assertion under various error models. A natural connection between the effectiveness of the order-sum assertion and the partition theory of numbers is shown. Asymptotic formulae for partition functions are derived.**

***Index Terms*— Sorting, assertions, error checking, partitions, testing, control-flow errors, data errors, watchdog checker.**

I. Introduction

ERRORS in the execution of computer programs can be classified into two categories: control-flow errors [7] and data errors [3]. *Control-flow errors are* errors that cause an execution of incorrect sequence of program or machine instructions. Temporary or permanent failures in program counter hardware, address translation buffers, and address computation units are some causes of control-flow errors. *Data errors* are errors that cause incorrect computation of final or intermediate results during program or machine instruction execution. Temporary or permanent failures in computation units like adders, in transmission units like busses, in storage units like registers and random access memory (RAM) are some causes of data errors. Fig. 1 illustrates the error checking environment and distinguishes the terms algorithm, program, and execution used in this paper. An *algorithm* is a formal procedural description. Algorithms are often described using mathematical constructs or high-level languages. A *program* is an implementation of an algorithm using high-level or machine instruction programming languages. *Execution* is an interpretation of a program and often uses hardware units like microprocessors. The error checking mechanisms described in this paper pertain to errors (as shown in Fig. 1) that happen during program implementation and program execution. The watchdog checker does error checking during or after program execution. The implementation of watchdog checker could be in software or hardware and is usually less complex than the program implementation or the execution hardware. Although the techniques described in this paper to some extent verify algorithmic correctness, the verification of algorithmic correctness is outside the scope of this paper.

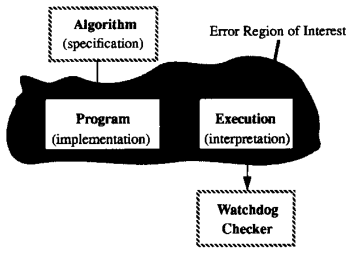


Fig. 1. Error checking environment.

The distinction between data errors and control-flow errors is important because not all data errors cause a change in the control-flow of programs. The detection mechanisms for control-flow errors often depend only on the machine instruction set architecture and the implementation of the execution hardware and not on the nature of algorithms solved by computer programs. Control-flow checking mechanisms have been proposed and studied by several researchers (for a survey see [7]). The mechanisms for detecting data errors, however, are dependent on the nature of algorithms implemented by computer programs. For example, to certify that an implementation of addition algorithm produces correct results it is important to have knowledge about some properties of the addition operation. Algorithm dependent mechanisms to detect data errors have been proposed [2], [5] for various algorithms.

In this paper an assertion is proposed to detect data errors during the execution of sorting algorithms. An *assertion* is a statement of truth about some property of the object under investigation. To derive assertions the following sorting algorithm model is assumed. The input data is an array of words denoted by $B = b_1, b_2, \cdots, b_n$. The sorting algorithm produces an array of sorted words denoted by $A = \text{sort}(B)$. Here $A = a_1, a_2, \ldots, a_n$ and $a_1 \geq a_2 \geq \cdots a_n$. The ordering relation depends on the type of data words. The magnitude of a_i and b_i is bounded by $0 \leq a_i, b_i \leq m$, where m is a positive integer and is determined by the type of data words. The analysis presented in this paper assumes that

Manuscript received January 31, 1992; revised January, 1994. Recommended by H. A. Partsch.

N. R. Saxena is with HaL Computer Systems, Campbell, CA 95008 USA; e-mail: saxena@shasta.stanford.edu.

E. J. McCluskey is with the Department of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305-4055 USA.

IEEE Log Number 9401374.

the data words are unsigned integers. Extensions to other data types are possible and this is discussed in the summary section.

The following is an *if and only if* assertion [8] for sorting algorithms.

$(\forall A, B)$ $\{A = \text{sort}(B)$ *if and only if* $\text{order}(A)$ and permute $(A, B)\}$

The order assertion, *order*(A), verifies the property that the resulting data A is sorted. The permute assertion, *permute*(A, B), verifies that the resulting data A is a permutation of the input data B. The complexity of the order assertion is $O(n)$. The complexity of the permute assertion is $O(n) \log n$. This complexity is not acceptable in a watchdog checker [11], [7] based mechanism because the implementation of permute assertion requires the same complexity as that of the sorting algorithm. The use of the order assertion has been considered earlier [2], [5], [6] and has also been implemented in flight-software [6]. However, the effectiveness of the order assertion has not been quantified. The effectiveness of the order assertion under various error models is analyzed and quantified in this paper. Some pathological cases are also demonstrated where the order assertion fails to be effective. In addition, an assertion based on the sum function [10] is derived from the permute assertion and its complexity is $O(n)$. The function sum(A) is the arithmetic sum of a_i. The following assertion is easy to prove:

$$(\forall A, B)\{\text{if permute}(A, B) \text{ then sum}(A) = \text{sum}(B)\}$$

The following assertions also easily follow:

Order Assertion: $(\forall A, B)\{\text{if } A = \text{sort}(B),$ then $\text{order}(A)\}$

Sum Assertion: $(\forall A, B)\{$ if $A = \text{sort}(B),$ then $\text{sum}(A) = \text{sum}(B)$ $\}$

Order-Sum Assertion: $(\forall A, B)\{$ if $A = \text{sort}(B),$ then $\text{order}(A)$ and $\text{sum}(A) = \text{sum}(B)$ $\}$.

These assertions can be proved by using results in [8]. The results produced by the sorting program are verified by the watchdog checker using the order and the sum assertions. It is possible that some erroneous results generated by a sorting program are not detected by order or sum assertions. The examples in Table I illustrate this.

This paper examines and quantifies the effectiveness of using order and order-sum assertions under certain error models. The effectiveness of error detection using the sum assertion can be derived from earlier reported results [12] and therefore is not presented in this paper. The implementation of the order assertion only requires the knowledge of ordering relation. In this respect the order assertion has an implementation advantage over the sum assertion because the sum assertion requires precomputed value of the sum of array B.

It is intuitively clear that combining two different assertions yields better error detection coverage than any single assertion. Section II illustrates the importance of enhancing the error detection capabilities of the order assertion by using order-sum assertion. The rest of the paper presents analysis of error detection capability of these assertions.

TABLE I
EXAMPLES OF ERROR DETECTION SCENARIOS BY ORDER AND SUM ASSERTIONS

Data B	Data A after sorting	Sum(A)=Sum(B)	Order(A)	Comments
2,1,4,3	4,3,2,1	True	True	Error free results
2,1,4,3	3,3,2,1	False	True	Error detected by sum assertion and not detected by order assertion
2,1,4,3	4,3,1,2	True	False	Error detected by order assertion and not detected by sum assertion
2,1,4,3	3,2,1,2	False	False	Error detected by order and sum assertions
2,1,4,3	4,2,2,2	True	True	Error not detected by order and sum assertions

II. ENHANCING ORDER ASSERTION BY ORDER-SUM ASSERTION

In this section, by way of example, some pathological error situations are presented where the order assertion fails to detect any error.

Scenario 1: The most primitive operations in any sorting program are the compare and exchange of data words b_i and b_j ($\forall i, j$ and $i \neq j \in \{1, 2, \cdots, n\}$). Let the initial data be $B = b_1, b_2, b_3, b_4 = 1, 1, 2, 3$. Suppose the data word b_1 in the presence of some temporary failure (bit errors in storage elements) is converted from 1 to 3 before the compare and exchange operations. If no other temporary failure occurs after the failure that corrupted b_1, then the result of the sorting program will be $A = 3, 3, 2, 1$. Clearly, this error is not detected by the order assertion. In fact, any error that corrupts the data words b_i prior to compare and exchange operations will not be detected by the order assertion. Other scenarios also can be constructed where errors occur during the compare and exchange operation and these errors are not detected by the order assertion. These scenarios provide a reasonable justification for using other assertions like the sum or the order-sum assertions.

Scenario 2: In general, the most significant bits in the data words b_i usually decide the ordering relation of the data. This is particularly true for unsigned integer data types. Given this fact, if errors corrupt the low-order or the least significant bits in the data words (during computation or transmission) it is likely that the order assertion does not detect these errors. The example in Table II illustrates this. The original error-free sorted data is 5, 3, 2, 1. Table II shows the results for the sorted array with single or multiple data word errors in their least significant bit positions (bit 0).

From Table II, the order assertion fails to detect least significant bit errors in 11 out of 15 cases; whereas, the sum assertion fails to detect 3 out of 15 cases and the order-sum assertion does not detect 2 out of 15 error cases. In this example, the order-sum assertion offers at least a factor of 5 improvement in the error detection capability of the order assertion.

TABLE II
ERROR INJECTION IN THE LEAST-SIGNIFICANT BIT OF DATA WORDS

Location of Errors	Data Words a_1,a_2,a_3,a_4	Order Assertion	Sum Assertion	Order-Sum Assertion
Error Free	5,3,2,1	True	True	True
Error in Word 1	4,3,2,1	True	False	False
Error in Word 2	5,2,2,1	True	False	False
Error in Word 3	5,3,3,1	True	False	False
Error in Word 4	5,2,2,0	True	False	False
Error in Words 1,2	4,2,2,1	True	False	False
Error in Words 1,3	4,3,3,1	True	True	True
Error in Words 1,4	4,3,2,0	True	False	False
Error in Words 2,3	5,2,3,1	False	True	False
Error in Words 2,4	5,2,2,0	True	False	False
Error in Words 3,4	5,3,3,0	True	True	True
Error in Words 1,2,3	4,2,3,1	False	False	False
Error in Words 1,2,4	4,2,2,0	True	False	False
Error in Words 1,3,4	4,3,3,0	True	False	False
Error in Words 2,3,4	5,2,3,0	False	False	False
Error in Words 1,2,3,4	4,2,3,0	False	False	False

The rest of the paper presents analytical and experimental analysis of the effectiveness of order and order-sum assertions under certain error models.

III. SINGLE-WORD ERRORS

Single-word errors in arrays A are assumed in this section. This model can be justified by considering several scenarios during the execution of sorting algorithms. For example, a temporary failure occurs in a register that stores one of the words of data during compare and exchange operations in the sorting algorithm. It is assumed that errors in a word are arbitrary and can change it to any integer value within the bounds of finite-precision. Experimental analysis of other restricted error models is presented in the sections that follow. The following theorem quantifies the effectiveness of using the order assertion under the single-word error model.

Theorem 1: If an array A has a single-word error and this error can occur in any of the n words then there are $m+a_1-a_n$ single-word errors that preserve order(A).

Proof: There are two cases to be considered:

Case 1: (Errors in a_1 or a_n). The number of possible erroneous a_1 such that order(A) is preserved is $m - a_2$. This is because a_1 can assume any value in the range $[a_2, m]$. Similarly, the number of possible erroneous a_n such that order(A) is preserved is $a_n - 1$.

Case 2: (Errors in $a_i, 2 \leq i \leq n-1$). The number of possible erroneous a_i such that order(A) is preserved is $a_{i-1} - a_{i+1}$.

Adding Case 1 and Case 2, the total number of single-word errors that preserve order(A) is given by

$$m - a_2 + a_{n-1} + \sum_{i=2}^{n-1}(a_{i-1} - a_{i+1}) = m + a_1 - a_n$$

(summation simplified by telescoping the sum).

Corollary 1: The least upper bound on the number of single-word errors that preserve order(A) is $2m$.

Proof: Clearly, the maximum value occurs when $a_1 = m$ and $a_n = 0$.

Example 1: Let $m = 9, n = 4$, and $A = 7,5,4,3. a_1 = 7$ and $a_n = 3$. There are $m + a_1 - a_n = 9 + 7 - 3 = 13$ single-word error arrays that preserve order(A). The following is an enumeration of these arrays: The words in bold font represent the words in error.

9,5,4,3	7,**7**,4,3	7,5,**5**,3	7,5,4,**4**
8,5,4,3	7,**6**,4,3	7,5,**3**,3	7,5,4,**2**
6,5,4,3	7,**4**,4,3		7,5,4,**1**
5,5,4,3			7,5,4,**0**

Given an assertion and an error model, the *aliasing fraction* is the fraction of undetected errors. There are mn possible single-word errors, therefore the aliasing fraction is $\frac{m+a_1-a_n}{mn}$. The worst case aliasing fraction is $\frac{2}{n}$ (follows from Corollary 1). If the order-sum assertion is used then the aliasing fraction for single-word errors is zero. Actually, in the single-word error model the sum assertion alone will guarantee zero aliasing fraction. This should be obvious because a single-word error in array A will cause sum(A) to be different from sum(B). *Error detection fraction* is the fraction of detected errors; and by definition, it is equal to one minus aliasing fraction.

IV. DOUBLE-WORD ERRORS

Double-word errors can occur in any two words in the data array and errors within a word are assumed equally likely. Theorems 2 and 3 enumerate the number of possible undetected double-word errors when the order and the order-sum assertions are used respectively.

Theorem 2: If the order assertion is used then the number of undetected double-word errors in array $A = a_1, a_2, \cdots, a_n$ is

$$\begin{aligned}&\frac{1}{2}(a_0^2 + a_1^2 - a_2^2 + a_n^2 - a_{n-1}^2)\\&\quad + \frac{3}{2}(a_0 + a_1 - a_n + a_2 - a_{n-1})\\&\quad - a_n(a_0 + a_1 - a_{n-2} - a_{n-1})\\&\quad - (m + 2a_1 + a_2 - a_{n-1} - 2a_n)\\&\quad + \sum_{i=1}^{n-2} a_{i+1}(a_{i-1} - a_{i+2}).\end{aligned}$$

Here $a_0 = m$.

Proof: The number of undetectable double-word errors can be partitioned into two classes:

1) errors in adjacent words,
2) errors in nonadjacent words.

First, we will count class 1) errors. Let the error locations be $j, j+1$ corresponding to errors in words a_j, a_{j+1}. Let $C(j)$ be the number of ways of choosing a'_j, a'_{j+1} such that $a_{j-1} \geq a'_j \geq a'_{j+1} \geq a_{j+2}$. Now consider length-two monotone sequence h_1, h_2, where $h_1 = a'_j - a_{j+2}$ and

$h_2 = a'_{j+1} - a_{j+2}$. Clearly, $0 \leq h_1, h_2 \leq a_{j-1} - a_{j+2}$. By definition, the number of such length-two monotone sequences is $C(j)$. From the results in Theorem 4, it easily follows that

$$C(j) = \binom{a_{j-1} - a_{j+2} + 2}{2}.$$

$C(j)$ includes the case where there are single-word errors and the case of no errors. Double-word errors are precisely those situations where a'_j and a'_{j+1} are selected such that $a'_j \neq a_j$ and $a'_{j+1} \neq a_{j+1}$.

The number of single-word errors included in $C(j)$ are $a_{j-1} - a_{j+2} + a_j - a_{j+1}$ (follows from results in Theorem 1).

The total number of double-word errors in class (1) is

$$\sum_{j=1}^{n-1} (C(j) - a_{j-1} + a_{j+2} - a_j + a_{j+1} - 1).$$

Here, $a_0 = m$ and $a_{n+1} = 0$.

Next, we will count the number of double-word errors in class 2). Now assume that errors occur in two non-adjacent locations i and j corresponding to words a_i and a_j, where $j \geq i+2$. The number of ways a_i can be in error is $a_{i-1} - a_{i+1}$. Similarly the number of ways a_j can be in error is $a_{j-1} - a_{j+1}$. Given a fixed i, j can be selected from the range $[i+2, \cdots, n]$. Therefore, the number of double-word errors involving a_i is given by

$$\begin{aligned}(a_{i-1} - a_{i+1}) &\sum_{j=i+2}^{n} (a_{j-1} - a_{j+1}) \\ &= (a_{i-1} - a_{i+1})(a_{i-1} + a_{i+2} - a_n).\end{aligned}$$

Now total number of double-word errors in class 2) is given by

$$\sum_{i=1}^{n-2} (a_{i-1} - a_{i+1})(a_{i+1} + a_{i+2} - a_n).$$

Adding class 1) and class 2) cardinalities and after some algebraic simplification we have the total number of double-word errors that preserve the order assertion as

$$\begin{aligned}&\frac{1}{2}(a_0^2 + a_1^2 - a_2^2 + a_n^2 - a_{n-1}^2) \\ &+\frac{3}{2}(a_0 + a_1 - a_n + a_2 - a_{n-1}) \\ &-a_n(a_0 + a_1 - a_{n-2} - a_{n-1}) \\ &-(m + 2a_1 + a_2 - a_{n-1} - 2a_n) \\ &+\sum_{i=1}^{n-2} a_{i+1}(a_{i-1} - a_{i+2}).\end{aligned}$$

Q.E.D.

Example 2: Let $m = 4, n = 4$, and $A = 4, 3, 2, 1$. From Theorem 2, there are 20 different double-word error arrays that are undetected by the order assertion. The following is an enumeration of these arrays: The words in bold font are the words that are error.

4,**4**,**4**,1	4,3,**3**,**3**	4,**4**,2,**2**	**3**,3,**3**,1
4,**4**,**3**,1	4,3,**3**,**2**	4,**4**,2,**0**	**3**,3,**1**,1
4,**4**,**1**,1	4,3,**3**,**0**	4,**2**,2,**2**	**3**,**2**,2,1
4,**2**,**1**,1	4,3,**1**,**0**	4,**2**,2,**0**	**2**,2,**2**,1
4,**1**,**1**,1	4,3,**0**,**0**	**3**,3,2,**2**,	
3,3,**2**,0			

It is evident from Theorem 2 that the number of undetected double-word errors has a quadratic dependence on the data.

Next, the effectiveness of the order-sum assertion under double-word errors is analyzed after the following definition.

Definition 1: Function $f(n, m, k)$ is the number of partitions of k into at most n parts such that each part is less than or equal to m.

Details on the theory of partitions can be found in [1]. Theorem 3 gives the number of undetected double-word errors when the order-sum assertion is used.

Theorem 3:

If the order-sum assertion is used then the number of undetected double-word errors is

$$\begin{aligned}&\sum_{j=1}^{n-1} (f(2, a_{j-1} - a_{j+2}, a_j + a_{j+1} - 2a_{j+2}) - 1) \\ &+\sum_{i=1}^{n-2} \left(\sum_{j=1}^{n-i-1} (\min[a_{i-1} - a_i, a_{i+j+1} - a_{i+j+2}] \right. \\ &\qquad \left. + \min[a_i - a_{i+1}, a_{i+j} - a_{i+j+1}]) \right).\end{aligned}$$

Here $a_0 = m$ and $a_{n+1} = 0$.

Proof: The number of undetectable double-word errors by the order-sum assertion can be partitioned into two classes:

1) errors in adjacent words,
2) errors in non-adjacent words.

First, we will count the number of undetectable double-word errors in class 1). Now consider errors in a_j, a_{j+1}. The number of undetectable double-word errors in this case is the number of different ways of selecting $a'_j \neq a_j$ and $a'_{j+1} \neq a_{j+1}$, such that $a_{j-1} \geq a'_j \geq a'_{j+1} \geq a_{j+2}$ and $a'_j + a'_{j+1} = a_j + a_{j+1}$. Now, let us use the length two monotone sequence h_1, h_2 defined in the proof of Theorem 2. The monotone sequence should satisfy an additional constraint which is $h_1 + h_2 = a_j + a_{j+1} - 2a_{j+2}$. The partitions of $a_j + a_{j+1} - 2a_{j+2}$ into two or less parts such that each part is not greater than $a_{j-1} - a_{j+2}$ enumerates all the length two monotone sequences satisfying the above constraints and also the case where no error occurs in a_j and a_{j+1} (single-word errors are automatically excluded in the partition function because they cannot preserve the sum assertion). Therefore the number of undetectable double-word errors in this case is given by

$$f(2, a_{j-1} - a_{j+2}, a_j + a_{j+1} - 2a_{j+2}) - 1$$

In fact, one can easily derive the following closed-form for

$f(2, \alpha, \beta)$:

$$f(2, \alpha, \beta) = \left\lfloor \frac{\beta}{2} \right\rfloor + 1, \alpha \geq \beta$$

$$f(2, \alpha, \beta) = \left\lfloor \frac{2\alpha - \beta}{2} \right\rfloor + 1, \frac{\beta}{2} \leq \alpha < \beta$$

$$f(2, \alpha, \beta) = 0, \alpha < \frac{\beta}{2}.$$

The total number of undetectable double-word errors in class 1) is given by

$$\sum_{j=1}^{n-1} (f(2, a_{j-1} - a_{j+2}, a_j + a_{j+1} - 2a_{j+2}) - 1).$$

Next, we count the number of undetectable double-word errors in class 2). Now assume that errors occur in two non-adjacent locations i and $i + j + 1$ corresponding to words a_i and a_{i+j+1}, where $j \geq 1$. Suppose error in a_i increases its integer value. The maximum possible increment is $a_{i-1} - a_i$. This has to be compensated by an error in a_{i+j+1} which decreases its integer value by a corresponding amount. The maximum possible decrement is $a_{i+j+1} - a_{i+j+2}$. The maximum increments or decrements are governed by the fact that the sequence has to be monotone even in the presence of errors. Therefore the number of possible double errors that preserve the sum, the order and increase the value of a_i is $\min\{a_{i-1} - a_i, a_{i+j+1} - a_{i+j+2}\}$. In a similar manner we can show that the number of double-word errors undetected by the order-sum assertion that decrease the value of a_i is $\min\{a_i - a_{i+1}, a_{i+j} - a_{i+j+1}\}$. Given a fixed i, index j can be selected from the range $[1, \cdots, n-i-1]$. Therefore, the number of undetected double-word errors involving a_i is given by

$$\sum_{j=1}^{n-i-1} (\min[a_{i-1} - a_i, a_{i+j+1} - a_{i+j+2}] + \min[a_i - a_{i+1}, a_{i+j} - a_{i+j+1}])$$

Adding class 1) and class 2) cardinalities, the number of undetectable double-word errors by the order-sum assertion is given by

$$\sum_{j=1}^{n-1} (f(2, a_{j-1} - a_{j+2}, a_j + a_{j+1} - 2a_{j+2}) - 1) + \sum_{i=1}^{n-2} \left(\sum_{j=1}^{n-i-1} (\min[a_{i-1} - a_i, a_{i+j+1} - a_{i+j+2}] + \min[a_i - a_{i+1}, a_{i+j} - a_{i+j+1}]) \right). \quad \text{Q.E.D.}$$

Example 3: Let $n = 4, m = 4, A = 4, 3, 2, 1$. From Theorem 3, there are six double-word errors which are missed by the order-sum assertion. Following is their enumeration. (The words in the bold font are the words in error)

$4, \mathbf{4}, \mathbf{1}, 1$
$4, 3, \mathbf{3}, \mathbf{0}$
$\mathbf{3}, 3, \mathbf{3}, 1$
$\mathbf{3}, 3, 2, \mathbf{2}$
$4, \mathbf{4}, 2, \mathbf{0}$
$4, \mathbf{2}, 2, \mathbf{2}$

TABLE III
EXPERIMENTAL VALIDATION FOR DOUBLE-WORD ERRORS

Data Arrays	**Undetected by *order***	**Estimated Undetected for *order***	**Undetected by *order-sum***	**Estimated Undetected for *order-sum***	**Improvement Factor**
Array 1	33980 (3.39%)	34199	1312 (0.13%)	1271	33980/1312 = 25.9
Array 2	36856 (3.68%)	36858	1508 (0.15%)	1506	36856/1508 = 24.4
Array 3	35175 (3.51%)	35357	1543 (0.15%)	1551	35175/1543 = 22.8
Array 4	34277 (3.42%)	33806	1250 (0.12%)	1227	34277/1250 = 27.4
Array 5	36163 (3.61%)	35906	1430 (0.14%)	1428	36163/1430 = 25.3

Table III is an experimental validation of Theorems 2 and 3 for double-word errors. The number of double-word errors injected was 10^6. The experiments were done on 5 different 10-word ($n = 10$) data arrays. The precision of words was 6 bits ($m = 63$). The estimated undetected errors was based on exact estimates provided by Theorems 2 and 3. There is close agreement (within statis- tical variance due to experimental errors) between the experimental and estimated values. About 96 percent of double-word errors are detected by the order assertion. The order-sum assertion detects greater than 99 percent of double-word errors. In practice, n is significantly smaller than m so the enhancement of the order assertion to the order-sum assertion does provide a significant reduction in the aliasing fraction. The improvement factor column in this table shows the factor reduction in the number of undetectable errors by using order-sum assertion. Using results from [12], the estimated percentage of errors not detected by sum assertion for this experiment is $1.58\%(1/m * 100, m = 63)$ which is almost ten times that of order-sum assertion (Table III). In this example, we see that the order-sum assertion gives factor of 10 improvement over the sum assertion and factor of 25 improvement (from Table III) over the order assertion.

V. ALL POSSIBLE WORD ERRORS

Analysis of double-word errors in the foregoing section demonstrates the complexity in obtaining exact estimates on the number of undetected errors for the order and the order-sum assertions. The exact estimate of undetected errors for the order-sum assertion involves a partition function of numbers. Exact analysis of more than double-word errors is difficult. In this section error detection analysis is presented for arbitrary errors in the data words.

Theorem 4: Given all possible errors in array A, if the order assertion is used then the number $S(n, m)$ of undetected errors is given by

$$S(n, m) = \binom{m + n}{n} - 1.$$

Proof: Let $C(n,m) = S(n,m) + 1$. $C(m,n)$ is the number of monotone sequences $\{a_j\}j = 1,\cdots,n$, such that $0 \le a_j \le m$. The monotone sequences can be partitioned into two disjoint classes:
1) Sequences that have $a_n = 0$.
2) Sequences that have $a_n \ge 0$.

There are $C(n-1,m)$ class (1) sequences, this is obvious because if we take any length $n-1$ mono- tone sequence and append a zero at the end we obtain a class 1) sequence. Conversely, from a class 1) sequence if we drop a_n we obtain a length $n-1$ monotone sequence. There are 4p4p$C(n,m-1)$ class 2) sequences. This fact is true because, if we take a class 2) sequence and replace each a_j by a'_j such that $a'_j = a_{j-1}$, then we a have a monotone sequence $\{a'_j\}j = 1,\cdots,n$, such that $0 \le a'_j \le m-1$. Conversely, from any monotone sequence $\{a'_j\}$ by adding one to each a'_j we obtain a class 2) sequence. The following recurrence is easily obtained:
$C(n,m) = C(n,m-1) + C(n-1,m)$ for $m,n \ge 1$ $C(1,m) = m+1, C(n,1) = n+1$ are boundary conditions.

This recurrence can be solved using any standard technique. It can be easily shown that

$$C(n,m) = \binom{m+n}{n} = \binom{m+n}{m}$$

The reader can prove this using mathematical induction. From the definition of $C(n,m)$ we have

$$S(n,m) = \binom{m+n}{n} - 1. \qquad \text{Q.E.D.}$$

Example 4: $m = 3, n = 3$ and $A = 3,2,1$. From Theorem 4, $S(3,3) = 19$. The 19 arrays that are not detected are:

3,3,3	3,3,2	3,3,1	3,3,0	3,2,2	3,2,0	3,1,1
3,1,0	3,0,0	2,2,2	2,2,1	2,2,0	2,1,1	
2,1,0	2,0,0	1,1,1	1,1,0	1,0,0	0,0,0	

The aliasing fraction in this case is given by
Aliasing fraction $= \frac{1}{(m+1)^n-1}\left[\binom{m+n}{n} - 1\right]$.
To count the number of undetectable errors by the order-sum assertion some preliminary definitions and results are useful.

Definition 2: Function $p(n,m,k)$ is the number of partitions of k into exactly n parts such that each part is less than or equal to m.

Identity (1) follows from Definitions 1 and 2:

$$f(n,m,k) = \sum_{i=1}^{n} p(i,m,k) \tag{1}$$

Theorem 5: The function $p(n,m,k)$ satisfies the following recurrence relation:

$$p(n,m,k) = \sum_{i\ge1} p(n-1, m-i+1, k-1-(i-1)n).$$

Proof: Consider all the partitions enumerated by $p(n,m,k)$. Order them by the last element in the partition. Take the partitions that end with the last element 1, delete that 1 we get $p(n-1,m,k-1)$. In general, take the partitions that end with the last element i. Delete that i and subtract $i-1$ from the rest of the $n-1$ parts, we get $p(n-1, m-i+1, k-1-(i-1)n)$. Therefore we have

$$p(n,m,k) = \sum_{i\ge1} p(n-1, m-i+1, k-1-(i-1)n).$$

Q.E.D.

Theorem 6: If the order-sum assertion is used and all possible word errors are assumed then the number of undetectable errors is $f(n,m,k) - 1$. Here $k = \text{sum}(A)$.

Proof: The number of undetectable errors by the order-sum assertion is the number of monotone sequences $\{a'_j\}j = 1,\cdots,n$, where $0 \le a'_j \le m, \sum a'_j = k$, and there is at least one j such that $a'_j \ne a_j$. Using Ferrer's graph [1], we can show a bijection from the set of all length n monotone sequences $\{a_j\}$ with $\sum a_j = k$ to the set of all partitions of k into at most n parts such that each part $\le m$. The number of such partitions is $f(n,m,k)$. This also counts the error free case. Therefore, the number of undetectable errors by the order-sum assertion is $f(n,m,k) - 1$. Q.E.D.

The aliasing fraction for the order-sum assertion is given by Aliasing fraction $= \frac{f(n,m,k)-1}{(m+1)^n-1}$
The aliasing fraction for the order assertion for all possible word errors is independent of the data words in array A; however, for the order-sum assertion the aliasing fraction is dependent on the data words of A. To estimate the worst case aliasing fraction for the order-sum assertion, an upper-bound on $f(n,m,k)$ is needed. It is well known [1] that the function $f(n,m,k)$ exhibits unimodal[1] behavior as k varies from 0 to the maximum value mn. This function has the maximum value when integer k is the ceiling or floor of $mn/2$. See [9] for a combinatorial proof of the unimodality of $f(n,m,k)$. There is no known closed form expression for $f(n,m,k)$. In Theorem 7, an asymptotic formula for $f(n,m,k)$ is derived by exploiting the unimodality of this function.

Theorem 7: The following is an asymptotic formula for the function $f(n,m,k)$:

$$f(n,m,k) = \binom{m+n}{n} \frac{\sqrt{6}}{\sqrt{\pi mn(m+n+1)}} \exp\left(\frac{-6(k-\frac{mn}{2})^2}{mn(m+n+1)}\right).$$

Proof: $f(n,m,k)$ is the coefficient of q^k [1] in the following generating function:

$$G(n,m;q) = \frac{(1-q^{m+n})(1-q^{m+n-1})\cdots(1-q^{n+1})}{(1-q^m)(1-q^{m-1})\cdots(1-q)}.$$

We estimate $f(n,m,k)$, by formulating this problem in terms of probability theory. It is well known that the coefficients

[1] Class of functions that monotonically increase to a maximum value and then monotonically decrease.

TABLE IV
EXACT AND ASYMPTOTIC ESTIMATES OF $f(n, m, k)$

m	n	k	Exact	Asymptotic
8	9	36	910	933
8	24	84	157,706	159,337
8	24	146	17,528	17,115
4	99	171	26,367	27,074
8	99	401	1,508,428,039	1,537,500,087

$f(n,m,k)$ exhibit unimodal behavior. We can fit these coefficients in a Gaussian Distribution. By removing the poles at $q = 1$ in $G(n,m;q)$ we can show that

$$G(n,m;1) = \binom{m+n}{n}$$

Let $h(n,m;q) = \frac{G(n,m;q)}{\binom{m+n}{n}}$.

Clearly, $h(n,m;1) = 1.h(n,m;q)$ behaves as a probability moment generating function. The mean μ and variance σ^2 can be obtained as follows:

$$\mu = h'(n,m;1)$$
$$\text{and } \sigma^2 = h''(n,m;1) + h'(n,m;1) - h'(n,m;1)^2$$

After some algebra, we can show that $\mu = mn/2$, and $\sigma^2 = mn(m+n+1)/12$. Using the values of mean, variance, and the probability density function, we get the following asymptotic formula:

$$f(n,m,k) = \binom{m+n}{n} \frac{\sqrt{6}}{\sqrt{\pi mn(m+n+1)}} \exp\left(\frac{-6(k-\frac{mn}{2})^2}{mn(m+n+1)}\right). \qquad \text{Q.E.D.}$$

It is clear from the asymptotic formula that $f(n,m,k)$ has the maximum value when k is close to $mn/2$. Table IV compares asymptotic estimates with exact values of $f(n,m,k)$ for some values of m,n,k. The exact estimates were obtained using Identity (1) and the results in Theorem 5. The asymptotic values closely track the exact values whenever k is in the neighborhood of $mn/2$. This is sufficient for estimating an upper bound on the aliasing fraction for the order-sum assertion. From the structure of the asymptotic formula it can be easily seen that the number of undetected errors by the order assertion is reduced by at least a factor of $\frac{\sqrt{\pi mn(m+n+1)}}{\sqrt{6}}$. For large m, n this is a significant reduction in the aliasing fraction.

VI. CONCLUSION AND FUTURE WORK

Effectiveness analysis of the order and the order-sum assertions has been presented in this paper. The order assertion was shown to exhibit poor error detection behavior in certain error situations. The use of order-sum assertion enhances the error detection capability of the order assertion and the sum assertion. This was shown both analytically and experimentally. An advantage of the order assertion is that it does not require any encoding of the data. However, the sum assertion does require the encoding of the input data B with the corresponding sum(B) value. This requires an additional $O(n)$ complexity. Throughout this paper the analysis of the order-sum assertion was based on the assumption that summation of a_i was done without loss in precision.

The following issues were not considered in this paper and could be future research problems.

1) The effectiveness of the order-sum assertion under the assumption that the sum is computed modulo some finite number needs to be analyzed.
2) The error detection analysis in this paper was based on the assumption that the data words in arrays are unsigned integers. For signed integers the results in this paper are still applicable because the sum assertion can be implemented in the watchdog checker such that it treats these integers as unsigned. Any other data type (like floating-point numbers) also has a finite bit- field representation in memory. This finite bit-field representation can be treated as long integers by the watchdog checker. For example, the checking program implementing the sum assertion can interpret double precision 64-bit floating point numbers as unsigned 64-bit integers. The permutation property of sorting implies the sum assertion as long as the addition operation is commutative. The calculation of the number of errors that preserve the order and order-sum assertion for non-integer data types is not as simple as it was for integer data types. For example, in single-word errors the number of possible erroneous $a_i, 2 \leq i \leq n-1$, such that order(A) is preserved is $a_{i-1} - a_{i+1}$. The range of values that a_i can assume is computed merely by taking the arithmetic difference between a_{i-1} and a_{i+1}. This is not true for other non-integer data types. Error detection analysis for other important data types needs to be done.
3) In addition to the sum assertion, there are other assertions (e.g., product of data words in B = product of data words in A) that are implied by the permutation property of sorting. It would be worthwhile to investigate error detection behavior of assertions (other than sum assertion) implied by the permutation property.

ACKNOWLEDGMENT

We would like to thank all of the referees for very valuable suggestions.

REFERENCES

[1] G. E. Andrews, *The Theory of Partitions, Encyclopedia of Mathematics and Its Applications*, vol. 2. Reading, MA: Addison-Wesley, 1976.
[2] P. Banerjee and J. A. Abraham, "Fault-secure algorithms for multiple processor systems," in *Proc. 11th Int. Symp. Comput. Architect.*, Ann Arbor, MI, June 1984, pp. 279–287.
[3] U. Gunneflo, J. Karlsson and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation,". in *Proc. Nineteenth Int.Symp. on Fault-Tolerant Computing*, June 1989, pp. 340–347.
[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach* Morgan Kaufman, Jan. 1990.
[5] K. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, June 1984.
[6] A. Mahmood, D. Andrews, and E. J. McCluskey, "Writing executable assertions to test flight software," Center for Reliable Comput. Tech.

Rep. 84–14, Depts. of Elect. Eng. and Comput. Sci., Stanford Univ., Stanford, CA, Nov. 1984.

[7] A. Mahmood and E. J. McCluskey, "Concurrent error detection using watchdog processors—A survey," *IEEE Trans. Comput.*, vol. 37, no. 2, pp. 160–174, Feb. 1988.

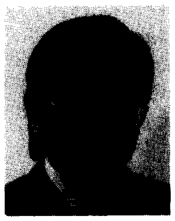
[8] Z. Manna and J. Waldinger, *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning*. Reading, MA: Addison-Wesley, 1985.

[9] K. M. O'Hara, "Unimodality of Gaussian coefficients: A constructive proof," Res. Announcement, Preprint, Univ. of Iowa., 1989.

[10] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–232, June 1975.

[11] N. R. Saxena and E. J. McCluskey, "Control-flow checking using watchdog assists and extended-precision checksums," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 554–559, Apr. 1990.

[12] ——— "Analysis of checksums, extended-precision checksums, and cyclic redundancy checks," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 969–975, July 1990.



Nirmal R. Saxena (S'83–M'90) received the B.E. degree in electronics and communication engineering from Osmania University, India, in 1982; the M.S. degree in electrical engineering from the University of Iowa in 1984; and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA in 1991. He received a gold medal for topping the undergraduate level Mathematics Olympiad conducted by the Andhra Pradesh Mathematics Teachers Association.

From 1984 to 1986 he was with Hewlett-Packard as a test developoment engineer. From 1986 to 1991, he was a member of the technical staff in the System Architecture Laboratory at Hewlett-Packard, and participated in the definition of Precision RISC Architecture. His Ph.D. at Stanford was supported by Hewlett-Packard's Honors Cooperative and Resident Fellowship Programs. He is currently on the technical staff at HaL Computer Systems, Campbell, CA 95008; is associated with Stanford University as a Consulting Assistant Professor of electrical engineering, and is a visiting faculty member of electrical engineering at Purdue University. His research interests include VLSI design and test, fault-tolerant computing, combinatorial mathematics, probability theory, and computer architecture.

Dr. Saxena is a founder of *epion associates*, which provides consulting services on integrated circuit design and test, computer system performance analysis, fault-tolerant computing, and computer architecture.

Edward J. McCluskey (S'51–M'55–SM'59–F'65) received the A.B. degree (summa cum laude, 1953) in mathematics and physics from Bowdoin College, and the B.S. (1953), M.S. (1953), and Sc.D. (1956) degrees in electrical engineering from M.I.T., Cambridge, MA.

He worked on electronic switching systems at the Bell Telephone Laboratories from 1955 to 1959. In 1959, he moved to Princeton University, where he was Professor of electrical engineering and Director of the University Computer Center. In 1966, he joined Stanford University, where he is Professor of electrical engineering and computer science, as well as Director of the Center for Reliable Computing. He was an editor of the *ACM Journal* from 1963 to 1968, an ACM National Lecturer, a member of the ACM Curriculum Committee (1967–69), and an ACM SIGARCH Director (1970–75). He became an ACM Fellow in 1994. He is President of Stanford Logical Systems Institute, which provides consulting services on fault-tolerant computing, testing and design for testability. At Stanford University, he teaches and leads research in computer engineering, emphasizing testing and fault tuolerance.

Dr. McCluskey served as the first President of the IEEE Computer Society, and as member of the AFIPS Executive Committee. He is a member of the Organizing Committees of the IEEE DFT and BIST Workshops, and General Chairiman of the IEEE-CRC BAST Workshop. He was a founding member of the editorial board of IEEE *Design and Test Magazine*In 1984, he received the IEEE Centennial Medal and the IEEE Computer Society Technical Achievement Award in Testing. In 1990, he received the EURO ASIC 90 prize for Fundamental Outstanding Contribution to Logic Synthesis. The IEEE Computer Society honored him with the 1991 Taylor L. Booth Education Award. In 1994, he was awaraded the title of Doctor Honoris Causa by the Institut National Polytechnique de Grenoble. He has published several books and book chapters. His most recent book is *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, published in 1986 by Prentice-Hall. Book chapters include: "Design for Testability in Fault-tolerant Computing," D.K. Pradham, Editor; and chapters on "Logic Design" in the *Van Nostrand Reinhold Encyclopedia of Computer Science and Engineering;* and in *Reference Data for Engineers*, edited by E. C. Jordan.