April 1993

UILU-ENG-93-2214
CRHC-93-08

Center for Reliable and High-Performance Computing

PERIODIC APPLICATION OF CONCURRENT ERROR DETECTION IN PROCESSOR ARRAY ARCHITECTURES

Paul Peichuan Chen

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2214 CRHC-93-08	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1308 W. Main St. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Moffett Field, CA
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 7a 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Periodic Application of Concurrent Error Detection in Processor Array Architectures

12. PERSONAL AUTHOR(S)

CHEN Paul Peichuan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1993 April 22	110

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	modularity, high parallelism, VLSI/WSI, rea-time signal processing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Processor arrays can provide an attractive architecture for some applications. Featuring modularity, regular interconnection and high parallelism, such arrays are well-suited for VLSI/WSI implementations, and applications with high computational requirements, such as real-time signal processing.

Preserving the integrity of results can be of paramount importance for certain applications. In these cases, fault tolerance should be used to ensure reliable delivery of a system's service. One aspect of fault tolerance is the detection of errors caused by faults. Concurrent error detection (CED) techniques offer the advantage that transient and intermittent faults may be detected with greater probability than with off-line diagnostic tests. Applying time-redundant CED techniques can reduce hardware redundancy costs. However, most time-redundant CED techniques degrade a system's performance.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

PERIODIC APPLICATION OF CONCURRENT ERROR DETECTION IN PROCESSOR ARRAY ARCHITECTURES

BY

PAUL PEICHUAN CHEN

B.S., Stanford University, 1984
M.S., University of Illinois at Urbana-Champaign, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois

PERIODIC APPLICATION OF CONCURRENT ERROR DETECTION IN PROCESSOR ARRAY ARCHITECTURES

Paul Peichuan Chen, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1993
Prof. W. Kent Fuchs, Advisor

Processor arrays can provide an attractive architecture for some applications. Featuring modularity, regular interconnection and high parallelism, such arrays are well-suited for VLSI/WSI implementations, and applications with high computational requirements, such as real-time signal processing.

Preserving the integrity of results can be of paramount importance for certain applications. In these cases, fault tolerance should be used to ensure reliable delivery of a system's service. One aspect of fault tolerance is the detection of errors caused by faults. Concurrent error detection (CED) techniques offer the advantage that transient and intermittent faults may be detected with greater probability than with off-line diagnostic tests. Applying time-redundant CED techniques can reduce hardware redundancy costs. However, most time-redundant CED techniques degrade a system's performance.

Periodic Application of Concurrent Error Detection (PACED) is a technique introduced in this thesis to reduce the performance costs incurred through the use of time-redundant CED in processor array architectures. To check computations periodically instead of continuously, PACED varies the application of such CED techniques to a processor array in both time and space. The purpose of PACED is to provide probabilistic detection of transient, intermittent, and permanent failures in processor arrays while reducing the overhead of performing detection.

Since CED is not performed continuously when PACED is used, undetected errors may occur prior to an error indication. Therefore, upon error detection, not only the current outputs of the array but both recent and subsequent outputs may also be erroneous. This thesis investigates the confidence to place on system outputs when PACED is applied, deriving formulae to predict the amount of output to suspect as possibly erroneous for single processors, linear unidirectional and two-dimensional mesh-connected processor arrays. The error coverage afforded by PACED in these architectures is also studied. Finally, the performance impact of using PACED in each array type is studied using both an array simulation model that gives estimates of application completion times with low computational cost and results of experiments using an Intel iPSC/2 hypercube to simulate a 16-node unidirectional linear array and a 4×4 two-dimensional mesh array.

ACKNOWLEDGMENTS

I give sincere thanks and appreciation to my advisor Professor W. Kent Fuchs for his guidance and support during the course of my graduate studies. I also thank Professors Prithviraj Banerjee, Ravishanker K. Iyer and Michael C. Loui for serving on my committee.

I thank Antoine Mourad for his invaluable assistance with the confidence analyses presented in Sections 3.2 and 4.3 and the error coverage analysis of Section 3.3. I also thank Robert Dimpsey, Kumar Goswami, Inhwan Lee, and Dong Tang for their help with the curve fitting performed in Section 3.1.

I thank my parents for their support and understanding these past years.

Thanks to all of my friends in and out of the Center for Reliable and High-Performance Computing and special thanks to the following people. I wouldn't have done it without them — Dan Bailey, Kate Baumgartner, Jeff Baxter, John Bentrup, Randy Brouwer, Robert Dimpsey, Pat Duba, John Fu, Kumar Goswami, John Holm, Sabrina Hwu, Bob Janssens, Andrew Jeter, Ralph Kling, Suzanne Kuo, Marc Levitt, Jim Li, Robert and Dorothy Long, Matt Lowrie, Vicki McDaniel, Maria Mendez, Antoine Mourad, Robert Mueller-Thuns, Tom Niermann, Michael Peercy, Stony Peng, Paul Ryan, Joe Scanlon, Dale Schouten, Jude Shavlik, Jonathan Simonson, Craig Stunkel, Kurt Thearling, Paul Tobin, Nancy Warter, Vickie Willis, and Kun-Lung Wu.

I gratefully acknowledge the support provided by the Office of Naval Research (Contract No. N00014-89-K-0070).

Finally, thanks to Bach, Beethoven, Brahms, Brubeck, Costello, Gould, Jarrett, Haydn, Mozart, and Shostakovich for the soundtrack, and a very special thank-you to Melody for making it fun when it wasn't.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

Processor arrays can provide an attractive architecture for some applications. Featuring modularity, regular interconnection, and high parallelism, such arrays are well-suited for VLSI/WSI implementations and applications with high computational requirements, such as real-time signal processing.

Preserving the integrity of results can be of paramount importance for certain applications. In these cases, fault tolerance features should be used to handle component failures that could upset reliable delivery of a system's service. One aspect of fault tolerance is the detection of errors caused by faults. Techniques for error detection may be classified as either off-line, in which diagnostic tests are applied to the system, or concurrent, in which normal system operations are checked for errors. Concurrent error detection (CED) techniques offer the advantage that transient and intermittent faults may be detected with greater probability than with off-line methods.

This thesis considers the application of CED techniques to processor array architectures. To minimize the overhead caused by fault tolerance, both hardware and time redundancies should be minimized. Applying time-redundant CED techniques can reduce the hardware costs. Table 1.1 lists some examples of such techniques, which are described below.

TABLE 1.1.
EXAMPLE TIME-REDUNDANT CED TECHNIQUES.

Alternating logic	[1, 2]
Recomputing with shifted operands (RESO)	[3]
Comparison with concurrent redundant computation (CCRC)	[4]
Recomputing by alternate path	[5]
Data redundancy	[6]
Triple time redundancy	[7, 8]
Algorithm-based fault tolerance	[9, 10]
Saturation	[11]
Spare capacity	[12]

Time-redundant CED techniques have been used to detect faults in digital circuits. For example, in *alternating logic*, both the true and complemented values of a circuit's inputs are applied serially to produce two versions of the output [1, 2]. The two error-free versions are complementary for self-dual functions. Although all faults which manifest themselves as single stuck-at faults can be detected, this technique could require hardware modification to create self-dual functions from non-self-dual ones, as well as extra flip-flops for the sequential parts of the circuit. *Recomputing with shifted operands* (RESO) [3] also achieves error detection by comparing two results. Each computation is followed by a similar one which uses bit-shifted versions of the operands; the bit-shifted result is then shifted back and compared with the original result. RESO can detect all errors in ripple-carry and carry-lookahead adders due to one faulty bit slice, and all errors in array multipliers and array dividers due to one faulty cell.

At a higher architectural level, the method called *comparison with concurrent redundant computation* (CCRC) [4] compares the results of two identical computations performed concurrently on different processors. Similar to CCRC are the *recomputing by alternate path* method,

designed specifically for use in an FFT processor array [5], and the *data redundancy* technique, which uses idle processors to perform duplicate computations [6]. All three techniques can detect any errors caused by faults confined to a single processor.

Many CED techniques have been applied to processor arrays (see Table 1.2). Some examples include: alternating logic in divider arrays [14], RESO in linear logic arrays [15] and matrix-multiply arrays [16], CCRC in divider and bidirectional systolic arrays [4], data redundancy in both linear and mesh matrix-multiply arrays [6], *triple time redundancy* in linear systolic arrays [8], and *algorithm-based fault tolerance* in FFT arrays [9] and matrix operations arrays [10]. In triple time redundancy, adjacent triples of processors perform triple modular redundancy (TMR), a standard error-masking technique. Triple time redundancy can detect up to $\lceil n/3 \rceil$ faulty cells before reconfiguration is necessary, where n is the size of the array. However, two extra processing elements (PEs) are required, as well as additional interconnect and switches throughout the array. An earlier version of triple time redundancy used even more

TABLE 1.2. OVERHEADS OF TIME-REDUNDANT CED IN PROCESSOR ARRAYS.

technique	CED overhead	
Algorithm-based fault tolerance	< 50%	[13]
Alternating logic	≥ 100%	[14]
RESO	≥ 100%	[15, 16]
CCRC	≥ 100%	[4]
Data redundancy	≥ 100%	[6]
Triple time redundancy	≥ 200%	[8]
Γ-processes	≥ 200%	[17]
Overlapping H-processes	≥ 300%	[18]

hardware: approximately $3n/2$ cells were required, as well as increased complexity of the PEs and the interconnect [7].

Algorithm-based fault tolerance is a technique which, by modifying an algorithm to operate on specially encoded data, can provide both error detection and location. Though not as generally applicable as other CED techniques, extremely low performance cost can be realized since the fault-tolerance scheme is tailored to the specific application. Error coverages ranging from 85% to 100% have been reported with less than 10% performance degradation [13].

The *Γ-processes* [17] and *overlapping H-processes* [18] techniques employ different patterns of neighboring PEs within mesh-connected two-dimensional processor arrays to perform redundant computations. The Γ-processes technique can detect errors caused by faults confined to one of every three PEs, but requires an extra row and an extra column of PEs. Designed for algorithms whose main PE computation is of the form $(a \cdot b) * (c \cdot d)$ (where $\cdot$ and $*$ represent general binary operators), overlapping H-processes can detect any errors from one PE of any 4×4 subarray of an array.

The problem with most time-redundant CED techniques is that their use may degrade a system's performance. If PE utilization is less than 100% in an array, then such techniques may possibly be applied with very little performance cost. Data redundancy and CCRC rely on idle cycles at PEs within an array, caused by a 50% PE utilization inherent to the algorithm, in which to launch redundant computations. Though the completion time of a single problem is unaffected, the array loses its ability to interleave problems: its throughput is cut 50%. These techniques would incur an overhead of 100% if used in algorithms in which the PEs of the array were used continuously. The application of RESO to band matrix multiplication also relies on

idle cycles. Of three designs proposed [16], two had PE utilizations under 50% (33% and 50%), enabling application of RESO in the idle cycles. The third design's utilization was 100% until RESO was added: the data rate was halved to create artificial idle cycles between computations. Without resorting to such measures, RESO can incur a time overhead of 100% or more, since shifting of operands is required in addition to the replicated computation. When alternating logic is applied to divider arrays, at least 100% overhead results since the complemented versions of the inputs are applied interleaved with the actual inputs [14]. Both triple time redundancy and Γ-processes require every PE to perform three times as much work, which causes an overhead of at least 200%, ignoring the overhead due to the increased message traffic. Overlapping H-processes can reduce the throughput of a mesh array by 75% — a time overhead greater than 300%.

Periodic Application of Concurrent Error Detection (PACED) is a technique introduced in this thesis to reduce the performance degradation incurred through the use of time-redundant CED in processor array architectures. To check computations periodically instead of continuously, PACED varies the application of time-redundant CED techniques to a processor array in both time and space. The purpose of PACED is to provide probabilistic detection of transient, intermittent, and permanent failures in processor arrays, while reducing the overhead of performing detection. Error recovery is not provided by PACED. Other techniques, such as rollback or forward recovery, are necessary to handle recovery from detected errors.

Since CED is not performed continuously when PACED is used, undetected errors may occur prior to an error indication. Therefore, when an error is detected, not only the current outputs of the array but both recent and subsequent outputs may also be erroneous. This thesis first

investigates the confidence to place on a single processor's outputs when PACED is applied, deriving formulae to predict the amount of output to suspect as possibly erroneous. In linear processor arrays, checking patterns are created when constituent PEs perform PACED at different times; optimal scheduling of these patterns to minimize the error detection latency has been studied [19]. By use of these checking patterns, if errors can be propagated by PEs, then the amount of output to suspect upon error detection as possibly erroneous can be limited. It is then shown that high confidence in most linear array outputs can be achieved using CED applied relatively infrequently. Similar patterns of checking are then studied in two-dimensional mesh-connected processor arrays, to determine which outputs from the array to suspect as possibly erroneous upon error detection. The error coverages afforded by PACED in the single processor, linear array, and two-dimensional mesh array are also studied.

Finally, the performance impact of using PACED in each array type is studied using both an array simulation model that gives estimates of application completion times with low computational cost and results of experiments using an Intel iPSC/2 hypercube to simulate a 16-node unidirectional linear array and a 4×4 two-dimensional mesh array.

This thesis focuses on the use of PACED in processor array architectures. The idea of periodic checking has previously been applied to multicomputer systems: *saturation* [11] and *spare capacity* [12] use idle processors in large-grain parallel architectures to perform redundant copies of other processor's processes. Error detection is achieved by voting at each processor on the process results. The performance is affected only by the increased message traffic, which can be negligible when certain specific protocols are used [11].

Other architectures may profit from the application of PACED, for example, fine-grained parallel architectures that use very long instruction words (VLIW) to address multiple functional units (FUs). In the FUs of the CRAY-1 scalar unit, idle cycles have reduced the performance cost of using RESO to check computations to the range 0.2% to 17.3% for the Livermore Fortran kernels [20]. A similar result was obtained through simulations with the IMPACT VLIW machine model [21]. From those simulations, it was found that idle cycles in a 4-FU architecture running a set of integer Unix utilities limited the performance penalties from almost nil (0.2% for *wc*) to quite significant (*grep*: 161%) [22]. In both of these studies, however, checking was performed for every checkable computation, and the performance costs were dependent upon the chance coincidences of redundant computations with idle functional units. A form of PACED in which compile-time information is used to schedule redundant computations only during idle slots could reduce the performance costs. This technique has been employed with good results for control-flow checking on the Multiflow TRACE 14/300 [23]. Since 100% of the checking operations used otherwise idle resources, there was no estimated performance penalty (neglecting increased memory traffic), and greater than 99% of all control-flow errors were detected in the benchmarks tested. A compiler-assisted PACED scheme to provide data integrity could meet with similar success.

The contributions of this thesis are as follows. A method is introduced to reduce the performance costs of using time-redundant CED through periodic application. An analysis is provided to determine, upon error detection at a single processor using PACED, the confidence to place on that processor's outputs. Similar analyses are performed for the linear unidirectional and two-dimensional mesh-connected processor array architectures, assuming that errors can be

propagated through the array. The error coverage afforded by PACED in each architecture is also studied. A PACED checking-pattern simulator and analyzer are described that facilitate choosing PACED parameter values in the two-dimensional array to minimize the error detection latency and the amount of suspected output at error detection time. A performance simulation model is described that estimates the performance costs of PACED applied to the linear and two-dimensional arrays; results of experiments using the simulation model are also given. Finally, empirical data collected from experiments using the Intel iPSC/2 hypercube are provided that show PACED in linear and two-dimensional arrays can reduce the performance degradation incurred through the use of CED.

The organization of this thesis is as follows. Chapter 2 outlines the PACED technique. In Chapter 3, the confidence and error coverage analyses of a single processor using PACED are described, and similar analyses of PACED applied to linear unidirectional arrays and two-dimensional mesh arrays are presented in Chapters 4 and 5, respectively. Those chapters also discuss the performance of the array architectures using PACED. Finally, Chapter 6 summarizes and presents conclusions.

CHAPTER 2.

THE PACED TECHNIQUE

This thesis considers processor array architectures in which the constituent processing elements (PEs) are regularly interconnected and each PE communicates only with its local neighbors. The computational activity at each PE, called a *computation cycle*, consists of receiving input, performing a task with or without applying CED, and sending output. A *task* is a fine-grained set of data manipulations, such as a multiply-accumulate operation. "Fine-grained" means that many such tasks are required to complete a problem execution.

When PACED is applied to one PE of an array, it can be parameterized as follows. Let M be the period of CED application and let N be the duration of CED application, where $0 \le N \le M$. The parameters M and N govern the time distribution of CED at the processor: in any period of M computation cycles, N tasks are checked and $M - N$ tasks are unchecked. As a mathematical abstraction to facilitate analysis of the PACED technique, let the *checking sequence*, $CS_{M,N}$, be an array of M values as follows:

$$CS_{M,N}[r] = 1, \text{ for } 0 \le r \le N - 1,$$

$$CS_{M,N}[r] = 0, \text{ for } N \le r \le M - 1.$$

EXAMPLE 2.1: The checking sequence for $M = 13$ and $N = 5$ is

$$CS_{13,5} = (1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0). \qquad \square$$

Each value in the checking sequence represents the checking activity at a PE during one computation cycle. The entire sequence represents one M-computation cycle (M-cycle) *period.* The N checked computation cycles are represented by the N consecutive "1"s in $CS_{M,N}$. The $M - N$ unchecked computation cycles are represented by the $M - N$ subsequent "0"s in $CS_{M,N}$. The checking activity at a PE over time may be represented as a cyclic reading of the checking sequence array. Note that the definition of the checking sequence gives but one possible way to perform N checks in M cycles; there are a maximum of $\binom{M}{N}$ different ways to perform N-out-of-M checking (some combinations are simply shifts of other patterns). In the remainder of this thesis, only checking sequences as defined above are considered; a value from a $CS_{M,N}$ array will represent the checking activity at a particular PE at a particular computation cycle. Figure 2.1 shows a portion of the activity at a processor using PACED with $M = 5$ and $N = 2$. When N/M is small, less performance degradation can usually be expected, but small N/M also reduces the probability of error detection.

When PACED is applied to the constituent PEs of a processor array, M and N may in general vary at each PE in the array. A third parameter, the PE checking offset O, determines the

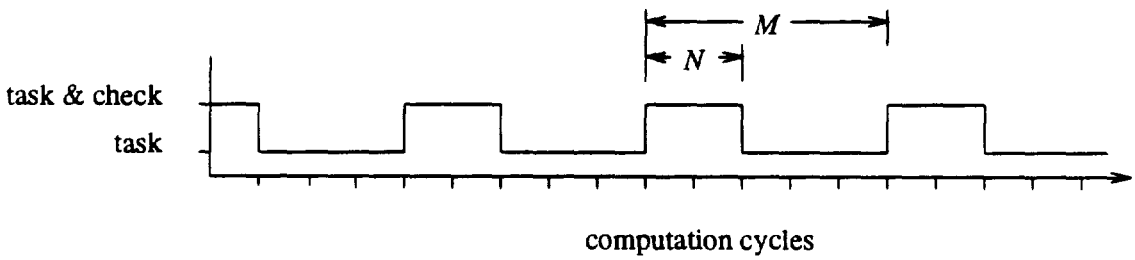


Figure 2.1. PACED parameters M and N.

initialization of each PE's first M-cycle period; O is an offset into the checking sequence $CS_{M,N}$. By varying O at each PE in an array, checking is performed at different times at different PEs. Snapshots of the checking activity in the array then reveal patterns of checking.

EXAMPLE 2.2: Given a $U \times V$ two-dimensional processor array using PACED, let the slope of the checking pattern be given by *RISE/RUN* and let the checking pattern be set by $O_{i,j} = (M_{i,j} + i + j - (U - 1 - i)RUN - (V - 1 - j)RISE) \bmod M_{i,j}$ at each $PE_{i,j}$.[1] Figure 2.2 shows snapshots of a 10×10 mesh-connected array using PACED with $M_{i,j} = 12$, $N_{i,j} = 4$, and *RISE/RUN* = 1/3, where each snapshot shows the checking activity in the array during one computation cycle. This checking pattern sets up waves of checking which advance upstream through the array, "catching," in effect, errors propagating downstream. □

The parameters described in this chapter are but one possible way to define PACED. As noted above, there are a maximum of $\binom{M}{N}$ different ways to perform N-out-of-M checking; this thesis only considers N consecutive checked cycles followed by $M - N$ unchecked cycles. A variation of PACED could be designed for arrays running algorithms with inherent idle cycles, so that CED would only be performed during PE idle times. Although the arrival of the idle cycles may be periodic, instead of a strict N-out-of-M schedule they may follow a more complicated pattern involving several different N and M values that change value in a periodic manner.

[1] Here and in the remainder of this thesis, the binary mod function is assumed to return a positive integer. To ensure this condition, multiples of the modulus can be added until the result is nonnegative while still less than the modulus.

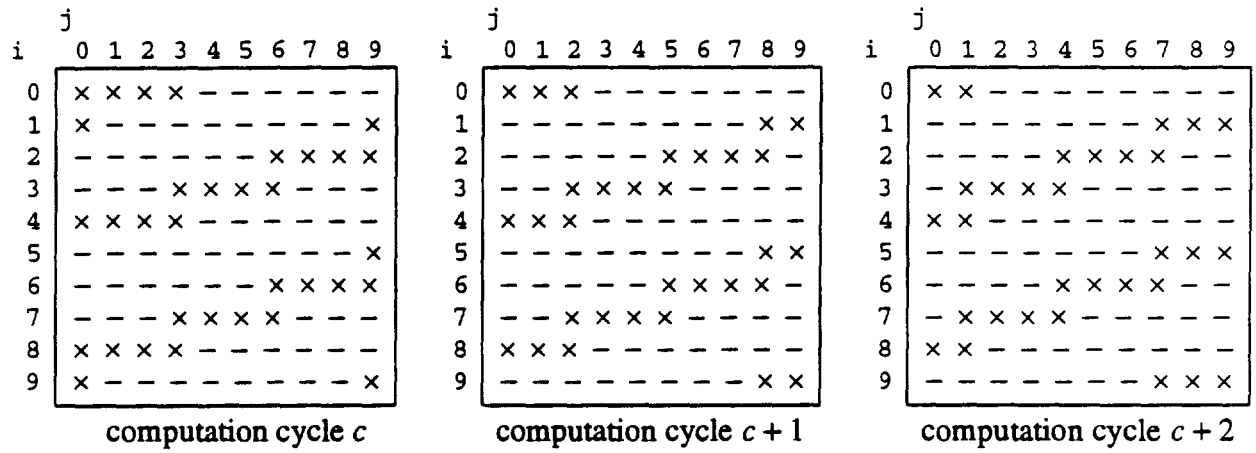


Figure 2.2. PACED in a 10×10 mesh-connected array.

In another variation, M and N values could be dynamic, assuming different values according to the particular application under execution, time of day, system workload fluctuations, or even the presence of detected errors. For example, normal PACED could prevail until an error is detected, to which the system might respond by increasing N or setting $N = M$ for some predetermined length of time. If no other errors are detected in this interval, then normal PACED would be resumed. This scheme could give assurance that an error arrival process has become inactive.

Finally, another variation might allow each PE to perform CED at its discretion, based on conditions such as individual workload or input data, thereby having no fixed values of M and N at all. This method of applying PACED could have potentially greater savings in performance costs than the type of PACED considered in this thesis, especially if CED can be scheduled to occur during idle cycles. If errors produced at PEs that are not checking can be propagated through the array, error coverage could still be very high. The use of these PACED variations, though not considered in this thesis, certainly merit further investigation.

CHAPTER 3.

PACED IN A SINGLE PROCESSOR

In this chapter, an analysis of PACED applied to a single processor will determine the confidence to place on that processor's outputs upon error detection. Because a processor using PACED does not perform CED continuously and because it is possible that the CED method employed does not have *perfect detection* (cannot detect all possible errors), there is a probability that some outputs produced prior and subsequent to an error indication may be erroneous. In some applications, e.g., image edge detection and image smoothing, a small number of errors may be tolerable. In other applications, however, high confidence in array outputs may be desired. For these cases, when an error is detected, it is important to know what confidence to place on outputs: which outputs to trust, and with what probability, and which outputs to suspect as possibly incorrect. Following the confidence analysis, the error coverage that can be expected when using PACED in a single processor will be investigated.

3.1. Error Arrival Model

Faults are generally characterized as one of three types: transient, intermittent, or permanent. Much work has been done in modeling the behavior of intermittent faults [24-27]. Because the primary interest of this study lies in the correctness of outputs, this thesis concentrates on errors; no assumptions are made concerning either the types or the distributions of faults that cause the errors. It has been shown that errors often arrive in clusters or

bursts [28, 29], perhaps caused by "incomplete fixes" in which repairs after an error detection insufficiently address the cause of the error, or error propagation, which can cause additional errors to appear after an initial detection. Thus, it is assumed that errors arrive in clusters (of one or more errors), that error clusters follow a Poisson arrival process with a constant mean arrival rate, and that the errors within clusters themselves follow a Poisson distribution. The following examples demonstrate that errors may arrive either clustered or singly. These examples confirm that the Poisson distribution serves as a good approximation to the error arrival process.

EXAMPLE 3.1: A Poisson arrival process was fitted to actual error arrivals measured on one machine of a "VAXcluster" distributed system. The system was composed of seven machines and four mass storage controllers, interconnected by the Computer Interconnect (CI) bus. The data were collected by the VAX/VMS operating system during normal operation of the machine "Earth," from 8 December 1987 to 14 August 1988 [29].

The SAS procedure NLIN (nonlinear regression) [30] was used to fit a two-phase hyperexponential function to the data for the machine "Earth" because a single exponential could not be found to fit the data well. The density of the fitted distribution $\hat{f}(t)$ is

$$\hat{f}(t) = 0.88(0.829\ e^{-0.829t}) + 0.12(0.012\ e^{-0.012t}),$$

where t is measured in minutes. Figure 3.1 shows $\Delta\ \hat{f}(t)$ superimposed upon the histogram of the time-between-error (TBE) data, where the bin size $\Delta = 5$ min. Note that the ordinate axis is shown on a log scale as the values quickly become very small. The sample mean and sample standard deviation for the data are also given in the figure. The fit was tested using the chi-square test and could not be rejected at the 0.28 significance level, with $r^2 = 0.99997$. The error

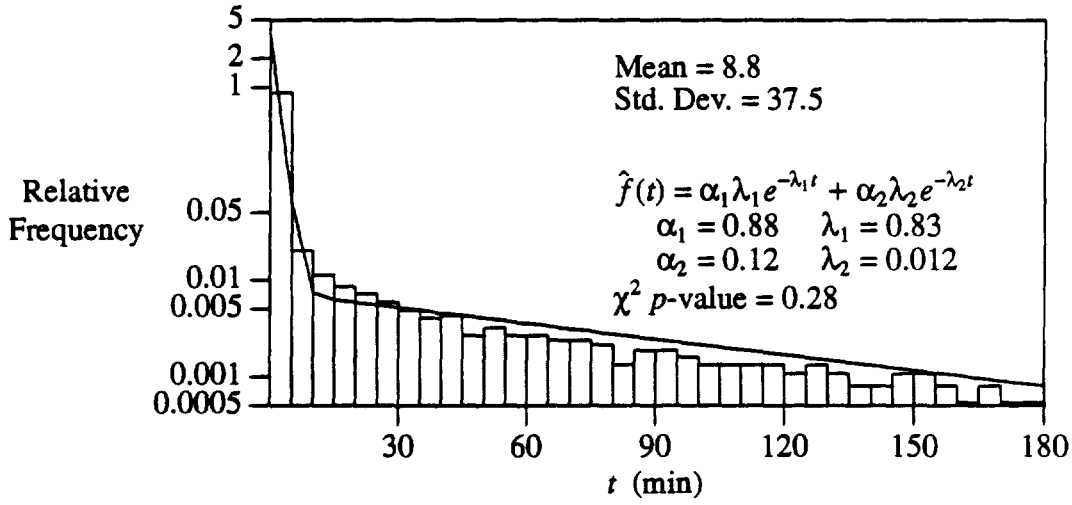


Figure 3.1. TBE histogram and fitted pdf for VAX "Earth."

arrival process is thus approximated by two homogeneous Poisson processes. Approximately 88% of the errors arrive in clusters with interarrival time 1.21 min [1/(0.829 error/min)] while approximately 12% of the arrivals signal new clusters with interarrival time 80.5 min [1/(0.012 error/min)]. □

Two-phase hyperexponential density distributions have also been used to model software error interarrivals on the VAXcluster taken as a whole and on a Tandem Cyclone multiprocessor [31]. In the following distributions, t is measured in days.

$$\hat{f}_{\text{VAX}}(t) = 0.67(0.20e^{-0.20t}) + 0.33(2.75e^{-2.75t})$$

$$\hat{f}_{\text{Cyclone}}(t) = 0.87(0.10e^{-0.10t}) + 0.13(2.78e^{-2.78t})$$

These distributions also can be interpreted as modeling errors that arrive in clusters. The VAX system has an intracluster rate of 2.75 error/day with new clusters arriving at a rate of 0.20/day; the Cyclone has an intracluster rate of 2.78 error/day and a cluster arrival rate of 0.10/day.

EXAMPLE 3.2: Single event upsets (SEUs) in spacecraft electronics have been studied extensively to develop techniques to estimate the rate at which such errors might occur [32]. Table 3.1 summarizes some observed SEU rates from various spacecraft. The wide range in SEU rates can be attributed to both the dependence of the error rate on the orbital environment and the sensitivity of the circuitry to the ionizing particles [33]. Again using NLIN, a single exponential was fit to the data from Pioneer, collected 10 January 1979 to 16 August 1990. The first row of Table 3.1 shows the mean arrival rate of the data. The density of the fitted distribution $\hat{f}(t)$ is

$$\hat{f}(t) = 0.039e^{-0.039t},$$

where t is measured in days. Figure 3.2 shows the histogram of the data overlaid with $\Delta\ \hat{f}(t)$, using a bin size Δ of 15.8 days. The figure shows the sample mean and sample standard deviation for the data as well. Using the chi-square test, the fit could not be rejected at the 0.11

TABLE 3.1.
OBSERVED SEU RATES.

satellite	observed SEU rate, errors/bit/day
Pioneer	3.3×10^{-2} [34]
Hughes Leasat	1.26×10^{-4} [32]
Hughes Leasat	2.44×10^{-4} [32]
Hughes Leasat	2.71×10^{-4} [32]
Pioneer	$< 2 \times 10^{-5}$ [35]
Unspecified	4.1×10^{-6} [36]
Unspecified	7.5×10^{-6} [37]
Unspecified	2.07×10^{-7} [38]
Voyager	$< 2.6 \times 10^{-9}$ [35]

significance level with $r^2 = 0.9958$. This exponential distribution models the arrival of clusters of errors which are composed of only single errors and that have mean interarrival time of 25.6 days [1/(0.039 error/day)]. □

3.2. Confidence Analysis

Suppose that a processor, perhaps a constituent of a processor array, is using PACED. When an error is detected, the current outputs of the processor should be suspected as being possibly erroneous. Use of PACED implies that checking may not have been performed continuously; this casts some doubt on both the recent and future outputs of the processor.

In the following sections, formulae are derived for determining how much output to suspect as possibly erroneous when an error is detected at a processor employing PACED. Given the evanescent nature of transient and intermittent faults, it is assumed that the error arrival

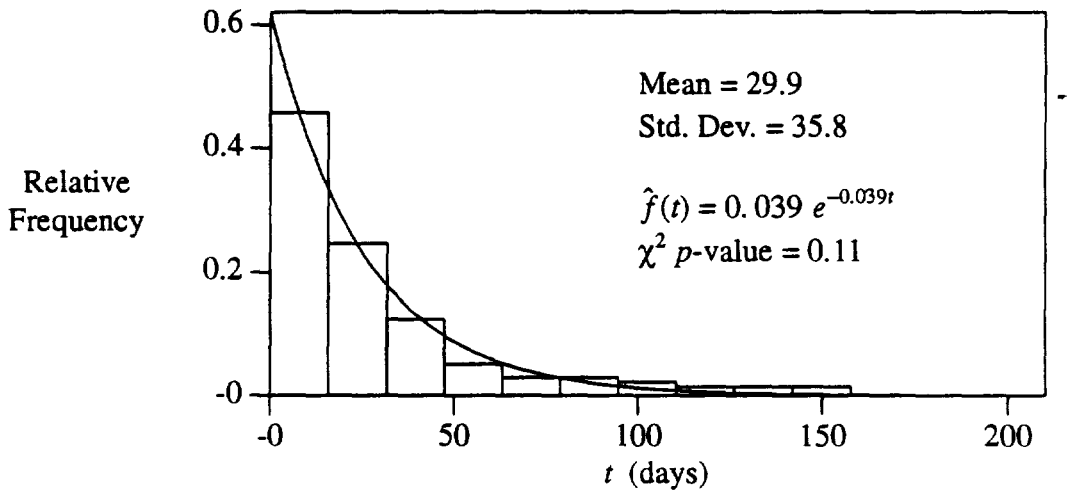


Figure 3.2. TBE histogram and fitted pdf for Pioneer.

process dies after a certain time; however, while active, the process is assumed to behave as a Poisson process. Assuming an error arrival distribution like that discussed in Section 3.1 in which errors arrive in clusters, the intracluster arrival rate is used as the parameter of the Poisson distribution (0.829 error/min from Example 3.1). By ensuring that the time to perform M cycles is small compared to the intercluster arrival time (80.5 min in Example 3.1), it can be further assumed that the detection of an error is independent of whether any other error is detected.

With these assumptions, it is first shown that detected error arrivals also follow a Poisson process. Let E_t represent the number of error arrivals in a time interval of length t. Since it is assumed that error arrivals follow a Poisson process, then their interarrival times are exponentially distributed. Let D_t represent the number of detected error arrivals in a time interval of length t. If the detected error interarrival time is exponentially distributed, this implies that detected error arrivals also follow a Poisson process. This lemma introduces the variable q, the *detection probability*, which is the probability that when a particular CED technique is applied (e.g., RESO), it will detect an error if one exists.

LEMMA 3.1: In a single processor using PACED with $1 \le N \le M$ and where the CED technique has detection probability $q \le 1$, detected error arrivals are exponentially distributed.

PROOF:

$$\Pr\{D_t = k\} = \sum_{n=k}^{\infty} \Pr\{E_t = n\} \binom{n}{k} \left(q \frac{N}{M}\right)^k \left(1 - q \frac{N}{M}\right)^{n-k}$$

$$= \left(\frac{q \frac{N}{M}}{1 - q \frac{N}{M}}\right)^k \sum_{n=k}^{\infty} \frac{(\lambda t)^n}{n!} e^{-\lambda t} \frac{n!}{k!(n-k)!} \left(1 - q \frac{N}{M}\right)^n$$

$$= \left(\frac{q\frac{N}{M}}{1-q\frac{N}{M}} \right)^k \frac{e^{-\lambda t}}{k!} \sum_{n=k}^{\infty} \frac{(\lambda t)^n \left(1-q\frac{N}{M}\right)^n}{(n-k)!}$$

$$= \frac{\left(q\frac{N}{M}\right)^k}{k!} e^{-\lambda t}(\lambda t)^k e^{\lambda t(1-q\frac{N}{M})}$$

$$= \frac{\left(\lambda q\frac{N}{M}t\right)^k}{k!} e^{-\lambda q\frac{N}{M}t}$$

This is a Poisson distribution, with modified error arrival rate $\lambda' = \lambda qN/M$. □

When an error is detected at a processor running PACED, some of the previously produced unchecked outputs may be erroneous. Also, some of the previously produced checked outputs may be erroneous, if the CED technique employed does not have perfect detection (i.e., $q < 1$). In addition, future outputs from the processor should also be suspected, since the detected error signals that a fault process is active and may be producing errors.

The next two subsections determine, when an error is detected, the intervals of time during which outputs should be suspected as possibly erroneous, given the desired level of confidence to place on unsuspected outputs. For each detected error, two time intervals in which to suspect outputs are found: one interval prior, and one subsequent, to the detected error. The lengths of these intervals are determined using two different criteria. 1) Fault-Active Intervals: Suspect all output produced in time intervals in which the fault was probably active. 2) Undetected-Errors Intervals: Suspect all output produced in time intervals that start from the time of the current detected error and extend backward to include, with a desired probability, the first undetected error, and forward to include, with a desired probability, the last undetected error. The length of

the time intervals found using the first criterion is called K and is derived in the following subsection; that found using the second criterion is called L and is derived in Section 3.2.2.

3.2.1. Fault-active intervals

Let K be the length of a time interval such that the probability that a detected error arrives within K is greater than some desired value C, where C, the *confidence*, is set arbitrarily close to 1. When an error is detected, if no other errors were detected within a time interval of length K prior to the detection, outputs produced earlier than K units of time before the time of the detected error can be trusted with confidence C (i.e., are correct with probability C): had the fault that caused the detected error been active K units of time previously, another detected error should have been observed, with probability C. With no other detected errors observed, the fault was probably (with probability C) inactive and outputs produced before that time are correct (can be trusted) with probability C. All outputs produced within K time units of the time of the detected error should be suspected as possibly erroneous: the outputs may be used, but the user should be aware that some of this suspected output may be incorrect.

In addition, outputs produced in the time interval of length K after the time of the detected error should also be suspected. If no other errors are detected in a time interval of length K after the time of the detection, then outputs produced later than K units of time after the time of the detected error can be trusted (are correct) with confidence C. Figure 3.3 illustrates the time intervals of length K. Theorem 3.1 gives an expression for K in terms of C, the PACED parameters, and the parameters of the detected-error arrival process.

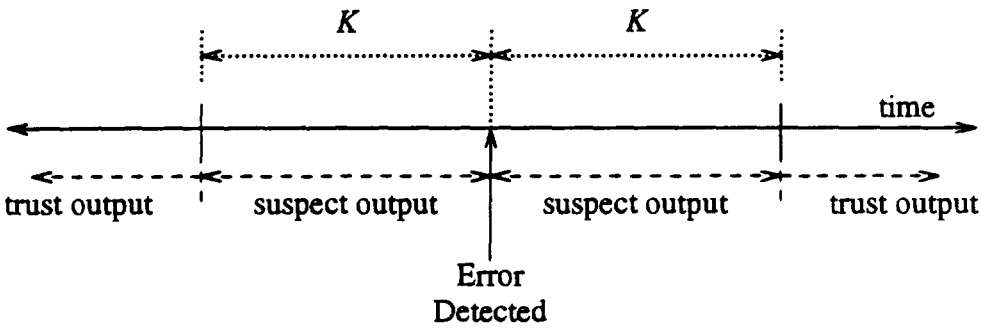


Figure 3.3. Outputs to suspect in fault-active intervals of length K.

THEOREM 3.1: Let a processor use PACED where $1 \leq N \leq M$ and the CED technique has detection probability $q \leq 1$. Upon error detection, outputs produced prior to a time interval of length K before the time of the detected error, or after an interval of length K subsequent to the time of the detected error, can be trusted with confidence C. The length K satisfies

$$K \geq -\frac{M}{N}\frac{1}{\lambda q}\ln(1-C) \ .$$

Outputs produced within K time units before or after the time of the detected error should be suspected as possibly erroneous, since the fault was active with probability C in those intervals.

PROOF: Let D represent the detected error interarrival time. Since detected errors follow a Poisson process with parameter $\lambda qN/M$, $\Pr\{D > t\} = e^{-\lambda q\frac{N}{M}t}$ and $\Pr\{D \leq t\} = 1 - e^{-\lambda q\frac{N}{M}t}$.

Let K be the length of a time interval such that $\Pr\{D \leq K\} \geq C$. Then,

$$1 - e^{-\lambda q\frac{N}{M}K} \geq C$$

$$K \geq -\frac{M}{N}\frac{1}{\lambda q}\ln(1-C) \ . \qquad \square$$

This expression for K can be used to predict, at error detection time, how much of both the most recent outputs and the subsequent future outputs to suspect as possibly erroneous. Conversely, given the length of time between the initial detected error and any preceding (or subsequent) detection, the level of confidence to place on outputs produced before (or after) that time interval may be determined, using Theorem 3.1.

For multiple detections, time intervals of length K are simply taken about each detection with no special significance attached to overlaps. Hence, if a second error detection occurs within K of a first, then the following outputs should be suspected: those produced within K before the first detection, those produced between the two detections and those produced within K after the second detection.

EXAMPLE 3.3: Let $q = 1$, $N/M = 0.5$, and $C = 0.99$. Using $\lambda = 0.829$ error/min from Example 3.1, if an error is detected, then by Theorem 3.1, outputs generated earlier than 11.1 min prior to the detected error, or later than 11.1 min after the detected error, can be trusted with a confidence of 0.99, provided no other errors were detected in those time intervals. All outputs produced less than 11.1 min before or less than 11.1 min after the detected error should be suspected as possibly erroneous. □

Figure 3.4(a) shows how the confidence is affected by the error arrival rate λ and the time interval length K, given a constant CED detection probability $q = 1$ and a constant amount of checking $N/M = 0.5$. The confidence varies from 0 to 1 on the z (vertical) axis; K varies on the x-axis (increasing to the right) and λ varies on the y-axis (increasing into the page). Figure 3.4(b) shows a zoom of Figure 3.4(a), focusing on confidences greater than 0.95 (the grid on the

N=5, M=10, q=1

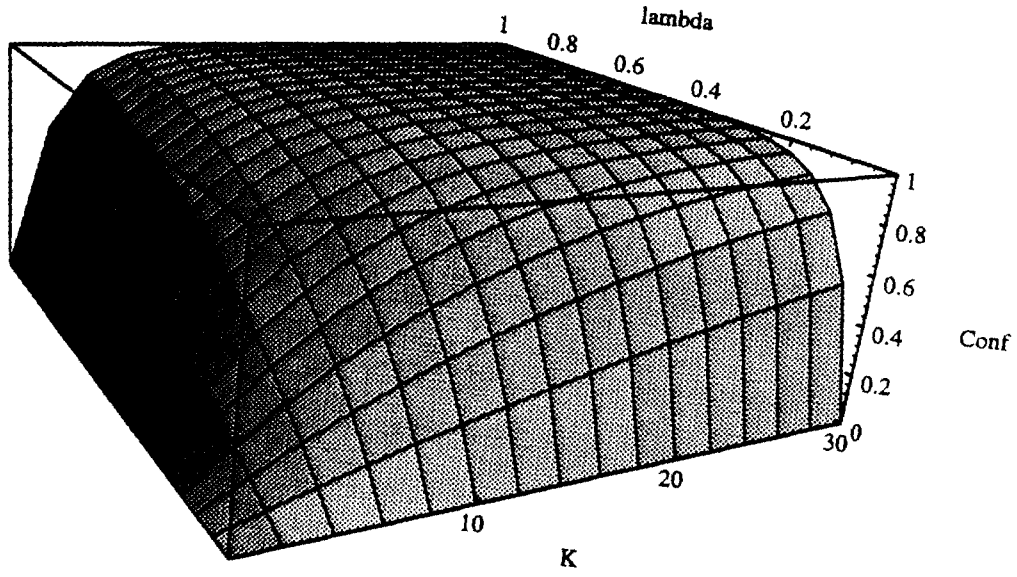


Figure 3.4(a). Fault-active intervals, C vs. λ ($0 \leq C \leq 1$).

N=5, M=10, q=1

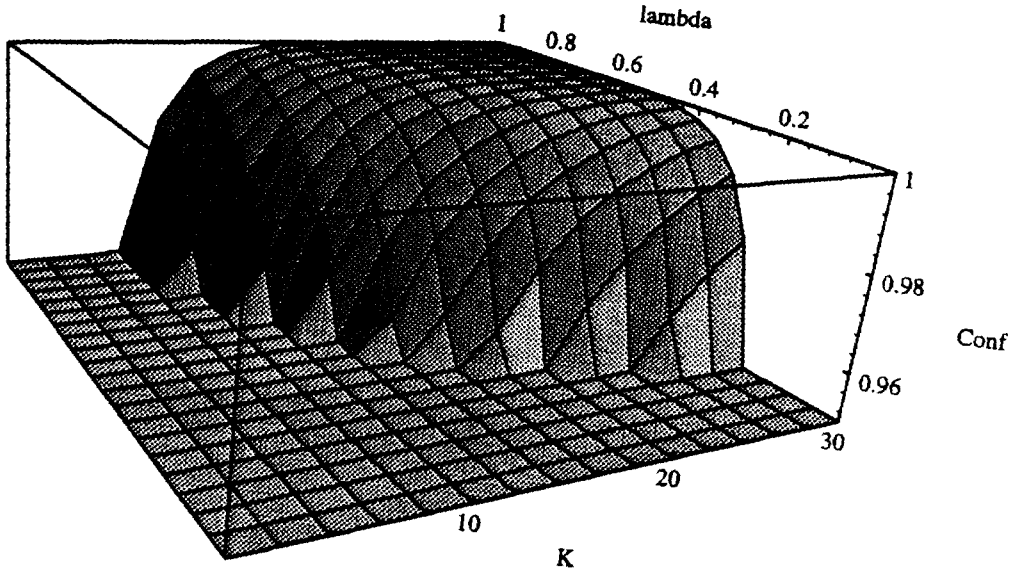


Figure 3.4(b). Fault-active intervals, C vs. λ ($C \geq 0.95$).

floor of the plot is not part of the function). As can be seen, K has to be larger if the error arrival rate λ is smaller, to achieve a given level of confidence. The assumption that errors arrive in clusters allows small values of K to reach high confidence levels: when $\lambda \geq 0.5$ error/min, confidences greater than 0.95 can be achieved with $K > 12$ min when the checking ratio N/M is just 0.5.

Figure 3.5 shows how the confidence is affected by varying amounts of checking, given a constant error arrival rate $\lambda = 0.829$ error/min and a constant CED detection probability $q = 1$. As on the previous plot, confidence is shown on the z-axis; here, however, K increases into the page (though still on the x-axis) and N increases to the left on the y-axis. Given a time interval K of just 12 min, a checking ratio value N/M greater than 0.3 will suffice to give greater than 0.95 confidence in outputs. Figure 3.5(b), a zoom of Figure 3.5(a) for $C \geq 0.95$, shows this clearly. This is an encouraging result as it allows designers utilizing PACED in a processor to use less than continuous checking (the goal of PACED, after all) and still achieve high confidence in outputs produced near a detected error.

Figure 3.6 shows how the confidence is affected by the CED detection probability q, given a constant error arrival rate $\lambda = 0.829$ error/min and a constant checking ratio $N/M = 1$. The axes are similar to Figure 3.5 except q replaces N on the y-axis. If $K > 12$ min is used, confidences over 0.95 can be achieved for any $q \geq 0.3$ (Figure 3.6(b)). This, too, is an encouraging result as it may be difficult in practice to estimate q accurately. This result shows that the precise value of q is not critical, for large enough K.

The similarity of Figures 3.5(a) and (b) to Figures 3.6(a) and (b), respectively, is not coincidental. From Theorem 3.1, $C \leq 1 - e^{-\lambda q \frac{N}{M} K}$. Holding q constant at 1 while varying N/M from

λ=0.829 error/min, M=10, q=1

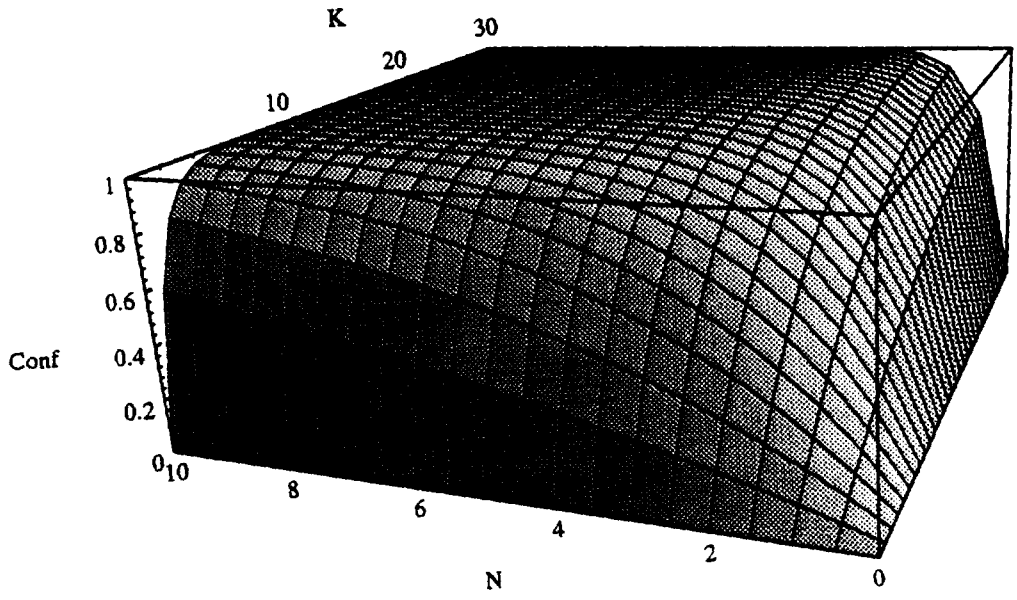


Figure 3.5(a). Fault-active intervals, C vs. N ($0 \leq C \leq 1$).

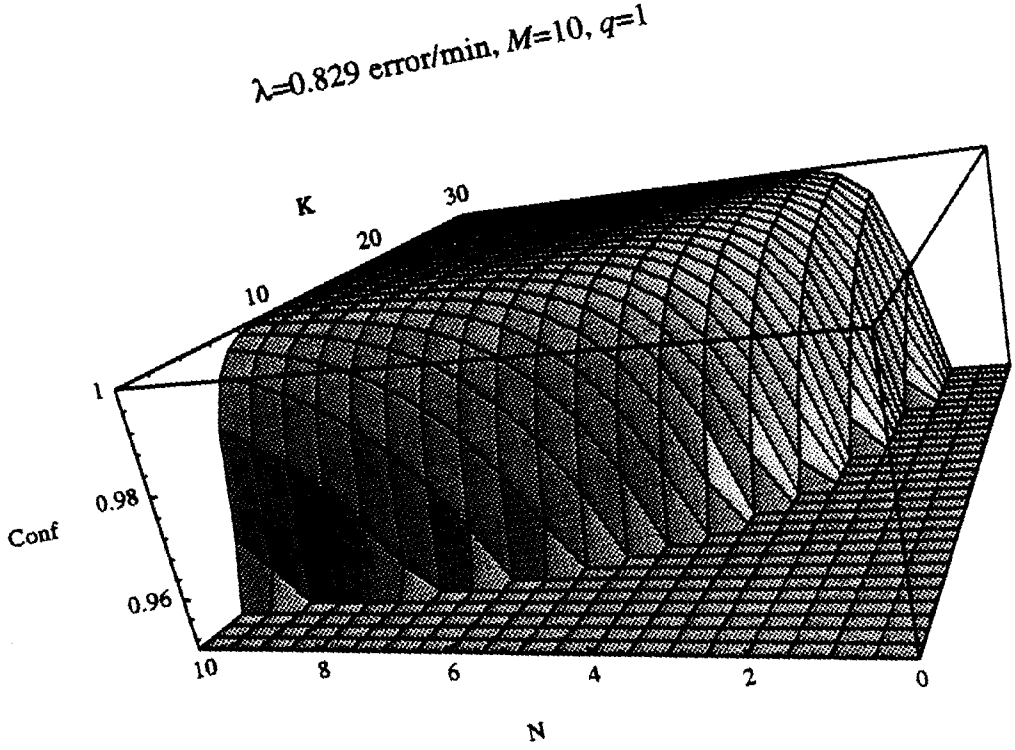


Figure 3.5(b). Fault-active intervals, C vs. N ($C \geq 0.95$).

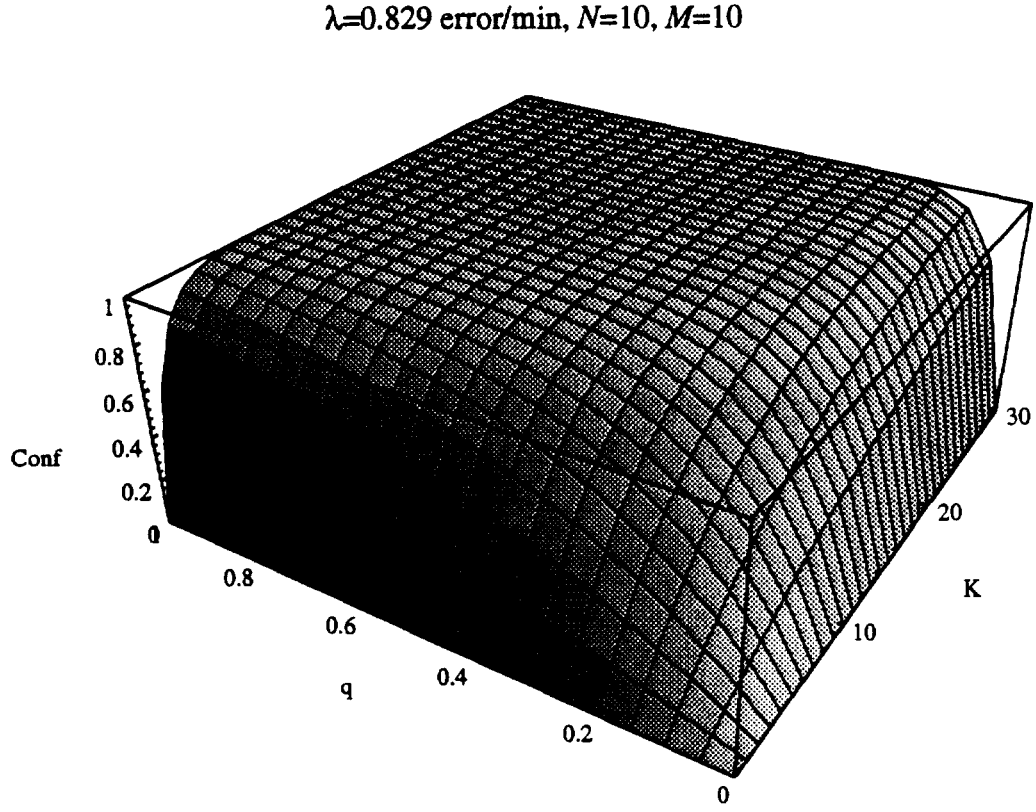


Figure 3.6(a). Fault-active intervals, C vs. q ($0 \leq C \leq 1$).

λ=0.829 error/min, N=10, M=10

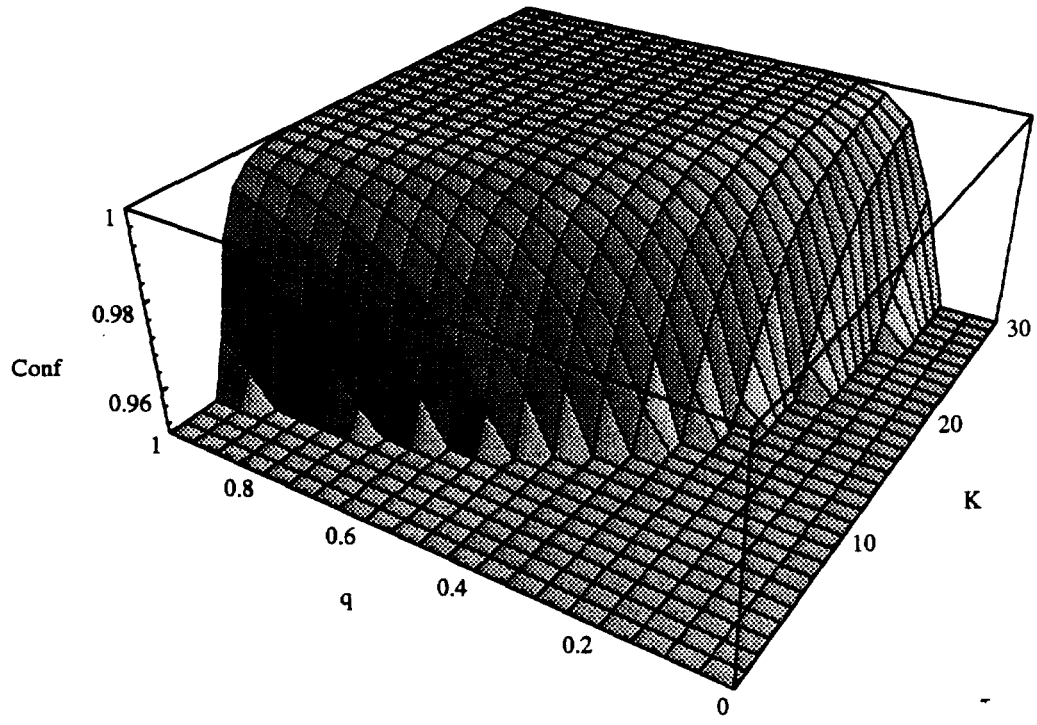


Figure 3.6(b). Fault-active intervals, C vs. q ($C \geq 0.95$).

0 to 1 (Figures 3.5(a), (b)) is equivalent to holding N/M constant at 1 while varying q from 0 to 1 (Figures 3.6(a), (b)). Thus, Figures 3.5 and 3.6 show identical plots but were both included and shown from different viewpoints to simplify the exposition.

When using the time intervals of length K to determine the confidence to place on outputs, it is assumed that if no other detected errors are found in the intervals, then with a certain probability the fault has become inactive. By using the times between the detected error and the first undetected error (looking backward) or the last undetected error (looking forward), time intervals of length $L < K$ can be obtained and fewer outputs need be suspected as possibly erroneous. The next subsection derives L using two different approaches: one to determine L looking backward in time from a detected error and the second to determine L looking forward in time from a detected error.

3.2.2. Undetected-errors intervals

To begin, it is shown that undetected errors, like detected errors, arrive following a Poisson distribution. Let E_t represent the number of error arrivals in a time interval of length t and U_t represent the number of undetected error arrivals in a time interval of length t. The proof of the following lemma is substantially similar to that of Lemma 3.1.

LEMMA 3.2: In a processor using PACED with $1 \le N \le M$ and where the CED technique has detection probability $q \le 1$, undetected error arrivals are Poisson distributed.

PROOF:

$$\Pr\{U_t = k\} = \sum_{n=k}^{\infty} \Pr\{E_t = n\} \binom{n}{k} \left(1 - q\frac{N}{M}\right)^k \left(q\frac{N}{M}\right)^{n-k}$$

$$= \left(\frac{1 - q\frac{N}{M}}{q\frac{N}{M}} \right)^k \sum_{n=k}^{\infty} \frac{(\lambda t)^n}{n!} e^{-\lambda t} \frac{n!}{k!(n-k)!} \left(q\frac{N}{M} \right)^n$$

$$= \left(\frac{1 - q\frac{N}{M}}{q\frac{N}{M}} \right)^k \frac{e^{-\lambda t}}{k!} \sum_{n=k}^{\infty} \frac{\left(\lambda t\right)^n \left(q\frac{N}{M} \right)^n}{(n-k)!}$$

$$= \frac{\left(\lambda \left(1 - q\frac{N}{M} \right) t \right)^k}{k!} e^{-\lambda(1-q\frac{N}{M})t}$$

This is a Poisson distribution, with modified error arrival rate $\lambda'' = \lambda(1 - qN/M)$. □

Lemma 3.3 establishes that the detected and undetected error Poisson processes are independent. This result will be used in Theorem 3.2 to form a joint pdf.

LEMMA 3.3:

$$\Pr\{U_t = k \;\&\; D_t = l\} \;=\; \Pr\{U_t = k\} \cdot \Pr\{D_t = l\}$$

PROOF:

$$\frac{\Pr\{U_t = k \;\&\; D_t = l\}}{\Pr\{D_t = l\}} = \frac{\Pr\{E_t = k+l\} \binom{k+l}{k} \left(1 - q\frac{N}{M}\right)^k \left(q\frac{N}{M}\right)^l}{\frac{\left(\lambda q \frac{N}{M} t\right)^l}{l!} e^{-\lambda q \frac{N}{M} t}}$$

$$= \frac{\frac{(\lambda t)^{k+l}}{(k+l)!} \binom{k+l}{k} \left(1 - q\frac{N}{M}\right)^k \left(q\frac{N}{M}\right)^l}{\frac{\left(\lambda q \frac{N}{M} t\right)^l}{l!} e^{-\lambda q \frac{N}{M} t}}$$

$$= \frac{\left(\lambda\left(1 - q\frac{N}{M}\right)t\right)^k}{k!} e^{-\lambda(1-q\frac{N}{M})t}$$

$$= \Pr\{U_t = k\}$$

Thus,

$$\Pr\{U_t = k \;\&\; D_t = l\} = \Pr\{U_t = k\} \cdot \Pr\{D_t = l\} \quad .$$

□

In the previous section, Theorem 3.1 determined the length K of time intervals during which the fault was probably active. Outputs from intervals of length K backward and forward from the time of an error detection were then suspected as possibly erroneous. The following two theorems use a less stringent criterion: looking backward from a detected error, only those outputs produced since the first undetected error need be suspected; or, looking forward, only those outputs produced up to the last undetected error need be suspected. These two intervals have length L: Theorem 3.2 determines L for the backward case and Theorem 3.3 determines L for the forward case. As will be shown, $L \leq K$. Figure 3.7 shows the relationship between the time intervals of lengths K and L, as well as which outputs to suspect and which to trust when using the L-length intervals.

THEOREM 3.2: Let a processor use PACED where $1 \leq N \leq M$ and the CED technique has detection probability $q \leq 1$. Upon error detection, outputs produced prior to a time interval of length L before the detected error can be trusted with confidence C, where the time interval extends backward from the time of the detected error to reach the first undetected error with probability C. The length L satisfies

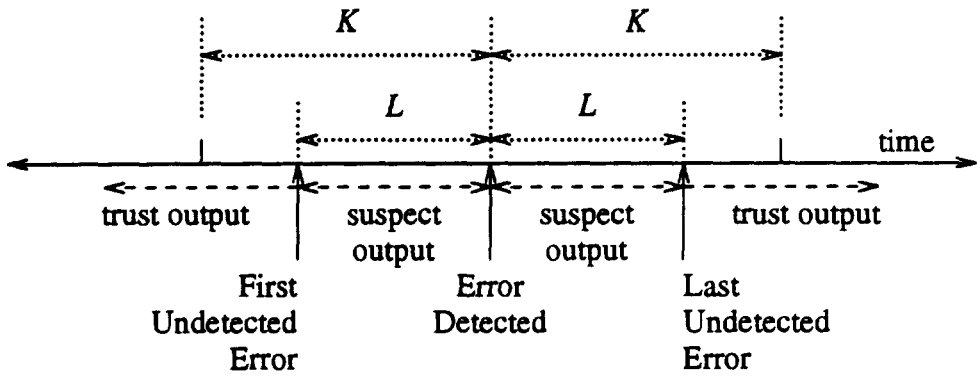


Figure 3.7. Outputs to suspect in undetected-errors intervals of length L.

$$L \geq -\frac{M}{N}\frac{1}{\lambda q}\ln\left(\frac{1-C}{1-q\frac{N}{M}}\right).$$

Outputs produced within length of time L before the detected error should be suspected as possibly erroneous.

PROOF: Let D and U represent the detected and undetected error interarrival times, respectively. From Lemmas 3.1 and 3.2, both random variables are exponentially distributed with parameters $\lambda' = \lambda qN/M$ and $\lambda'' = \lambda(1 - qN/M)$, respectively.

The quantity $D - U$ represents the time between the first undetected error and the first detected error. The probability $\Pr\{D - U > t\}$ is now determined using a joint probability distribution.

$$\Pr\{D - U > t\} = \int_0^\infty \int_{x+t}^\infty \lambda'' e^{-\lambda'' x} \cdot \lambda' e^{-\lambda' y} dy dx$$

$$= \int_0^\infty \lambda'' e^{-\lambda'' x} \cdot e^{-\lambda'(x+t)} dx$$

$$= e^{-\lambda' t} \frac{\lambda''}{\lambda'' + \lambda'} \int_0^\infty (\lambda'' + \lambda') e^{-(\lambda'' + \lambda')x} dx$$

$$= \frac{\lambda''}{\lambda'' + \lambda'} e^{-\lambda' t}$$

It follows that $\Pr\{U - D \le t\} = 1 - \frac{\lambda''}{\lambda'' + \lambda'} e^{-\lambda' t}$. Let L be the length of a time interval such that $\Pr\{D - U \le L\} \ge C$. Then,

$$1 - \frac{\lambda''}{\lambda'' + \lambda'} e^{-\lambda' L} \ge C$$

$$L \ge -\frac{M}{N} \frac{1}{\lambda q} \ln\left(\frac{1 - C}{1 - q\frac{N}{M}}\right).$$

Hence, with confidence C, the first undetected error occurred within a time interval of length L before the time of the detected error. Outputs produced prior to L time units before the time of the detected error can be trusted with confidence C and outputs produced within L units of time prior to the time of the detected error should be suspected as possibly erroneous. □

THEOREM 3.3: Let a processor use PACED where $1 \le N \le M$ and the CED technique has detection probability $q \le 1$. Upon error detection, outputs produced subsequent to a time interval of length L after the detected error can be trusted with confidence C, where the time interval extends forward from the time of the detected error to reach the last undetected error with probability C. The length L satisfies

$$L \geq -\frac{M}{N}\frac{1}{\lambda q}\ln\left(\frac{1-C}{1-q\frac{N}{M}}\right).$$

Outputs produced within length of time L after the detected error should be suspected as possibly erroneous.

PROOF: Let U represent the time to the last undetected error before the next detected error, and V, the time to the next detected error. From Lemmas 3.1 and 3.2, detected and undetected errors are exponentially distributed with parameters $\lambda' = \lambda qN/M$ and $\lambda'' = \lambda(1 - qN/M)$, respectively.

First, the probability is determined that the last undetected error occurs in some infinitesimal time slice du at time u while the next detected error occurs in some infinitesimal time slice dv at time v, where $v \geq u$. (If it were known that $v < u$, i.e., no undetected errors occur before the next detected error, then none of the outputs produced between the two error detections would have to be suspected.)

The expression below has a term for each of the following conditions: 1) no errors are detected in a time interval of length u starting from the time of the current error detection; 2) at least one error is undetected in an interval of length du; 3) no errors occur in an interval $v - u$; and 4) at least one error is detected in an interval dv. (The variable U should be defined as the time of the last undetected error before the fault becomes inactive, but since the distribution of fault lifetimes is unknown, U is predicated instead on the next error detection. In the derivation, then, the next detection is allowed to take place at any time slice dv from u to infinity, in effect allowing the fault to become inactive.)

$$\Pr\{(u \le \mathrm{U} \le u+du) \,\&\, (v \le \mathrm{V} \le v+dv)\} = e^{-\lambda' u}(1-e^{-\lambda'' du})e^{-\lambda(v-u)}(1-e^{-\lambda' dv})$$
$$= \lambda'\lambda'' e^{-\lambda' u} du \cdot e^{-\lambda(v-u)} dv$$

The terms $1-e^{-\lambda'' du}$ and $1-e^{-\lambda' dv}$ have been simplified using the approximation $1-e^{-x} \approx x + o(x)$ as $x \to 0$.

Now, the probability that U is greater than some L is determined, using the joint probability just derived.

$$\Pr\{\mathrm{U} \ge L\} = \int_L^\infty \int_u^\infty \lambda'\lambda'' e^{-\lambda' u} e^{\lambda u} e^{-\lambda v} dv du$$
$$= (1 - q\frac{N}{M}) \int_L^\infty \lambda' e^{\lambda(1-q\frac{N}{M})u} du \int_u^\infty \lambda e^{-\lambda v} dv$$
$$= (1 - q\frac{N}{M}) \int_L^\infty \lambda' e^{\lambda(1-q\frac{N}{M})u} du \cdot e^{-\lambda u}$$
$$= (1 - q\frac{N}{M}) e^{-\lambda' L}$$

L is determined such that $\Pr\{\mathrm{U} \ge L\} \le 1 - C$, where C, the confidence, is set arbitrarily close to 1.

$$\Pr\{\mathrm{U} \ge L\} \le 1 - C$$
$$(1 - q\frac{N}{M}) e^{-\lambda' L} \le 1 - C$$
$$L \ge -\frac{M}{N}\frac{1}{\lambda q} \ln\left(\frac{1-C}{1-q\frac{N}{M}}\right)$$

□

Note that Theorems 3.2 and 3.3 arrive at the same expression for L. It is attractive that the same time interval L is used both looking backward and forward from an error detection, as in

the case using K. It is also reasonable that the interval from the first undetected error to the first detected error should be the same length as that from the last detected error (the current detection can be considered the "last") to the last undetected error, given that the error distributions used in each case are the same. Also, it can be seen that the expression for L leads to smaller values than that for K: they differ only in the natural logarithm term. Both the numerator and denominator of this term in the expression for L are less than one. Hence, the quantity $1 - C$ is increased closer to 1, reducing the absolute value of the natural logarithm of the fraction and making L smaller than K for equal values of the other parameters.

EXAMPLE 3.4: If a single processor uses PACED with the same parameter values as in Example 3.3, viz., $q = 1$, $\lambda = 0.829$ error/min, $N/M = 0.5$, and $C = 0.99$, then when an error is detected, using Theorem 3.2, the outputs generated prior to 9.4 min before, or subsequent to 9.4 min after, the detected error can be trusted with a confidence of 0.99, as long as no other errors are detected in those time intervals. All outputs produced less than 9.4 min before the detected error or less than 9.4 min after should be suspected as possibly erroneous. □

In Example 3.3, outputs produced 11.1 min before and after the detected error had to be suspected, since Theorem 3.1 suspects all outputs generated while the fault was probably active. By suspecting only those outputs generated since the first or before the last undetected error, a time savings of about 15% can be realized in this case. Figure 3.8 plots the time savings $(K - L)/K$ of using L instead of K as a function of N/M, when $q = 1$, $\lambda = 0.829$ error/min, and $C = 0.99$.

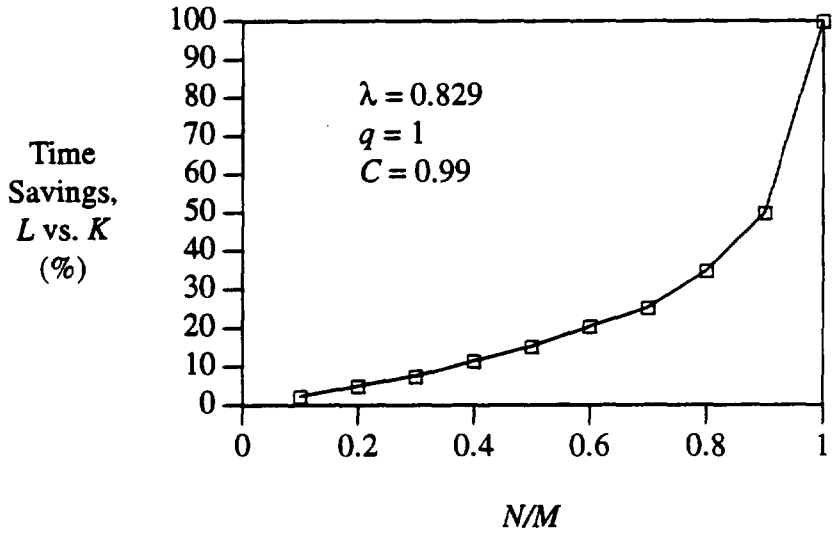


Figure 3.8. Time savings using L instead of K.

Figures 3.9–3.11 show the effect of L on the confidence. These graphs use the same axes and scales as Figures 3.4–3.6, respectively, and can be compared therewith directly. (As with Figures 3.5 and 3.6, Figures 3.10 and 3.11 show identical plots from different points of view.) For each figure a zoom plot shows confidences over 0.95.

By Theorems 3.2 and 3.3, $C \leq 1 - (1 - qN/M)e^{-\lambda q \frac{N}{M} L}$. As $L \to 0$, the confidence C becomes bounded above by qN/M. In Figure 3.9(a), $qN/M = 0.5$, so the plot never falls below 0.5; in Figures 3.10(a) and 3.11(a), qN/M varies from 0 to 1 and bounds C when L is close to 0.

All the figures show that for a given set of parameter values, a desired confidence level can be achieved with a value of L smaller than the necessary value of K. Figure 3.10 shows that the confidence, as in the case for K, is relatively insensitive to the checking ratio N/M, given $L > 10$ min; likewise, Figure 3.11 shows that for large enough L (> 10 min) the confidence is relatively unaffected by q. These results again indicate that high confidence can be achieved without the

N=5, M=10, q=1

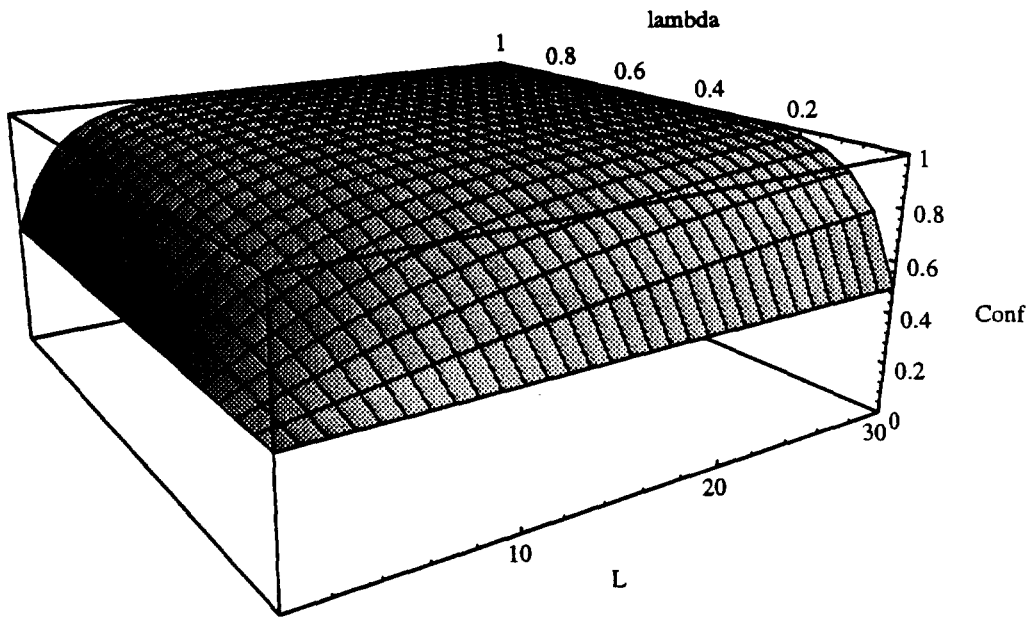


Figure 3.9(a). Undetected-errors intervals, C vs. λ ($0 \leq C \leq 1$).

N=5, M=10, q=1

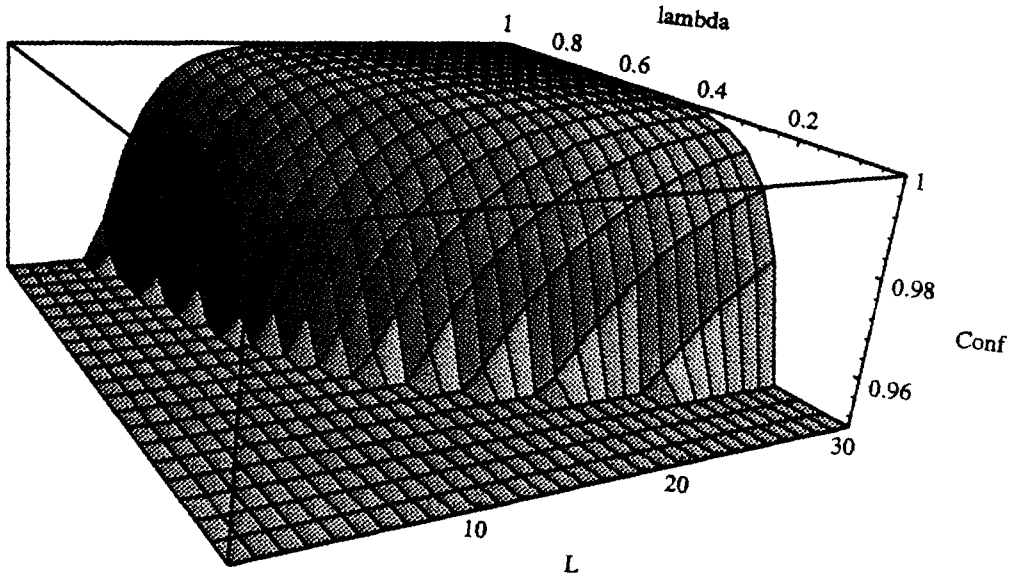


Figure 3.9(b). Undetected-errors intervals, C vs. λ ($C \geq 0.95$).

$\lambda = 0.829$ error/min, M=10, q=1

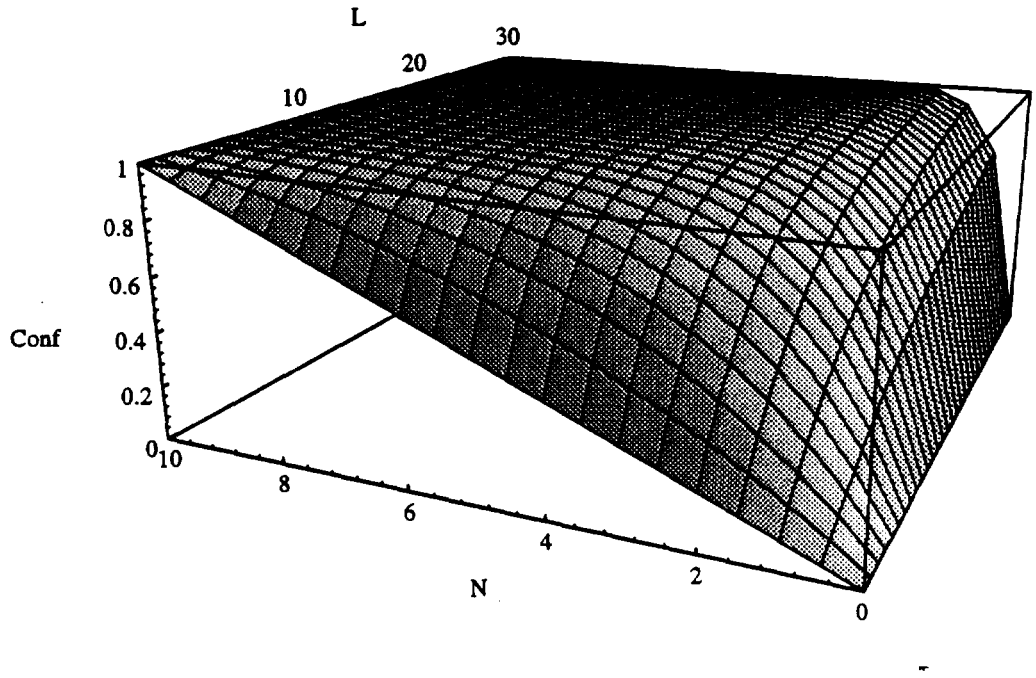


Figure 3.10(a). Undetected-errors intervals, C vs. N ($0 \leq C \leq 1$).

$\lambda = 0.829$ error/min, M=10, q=1

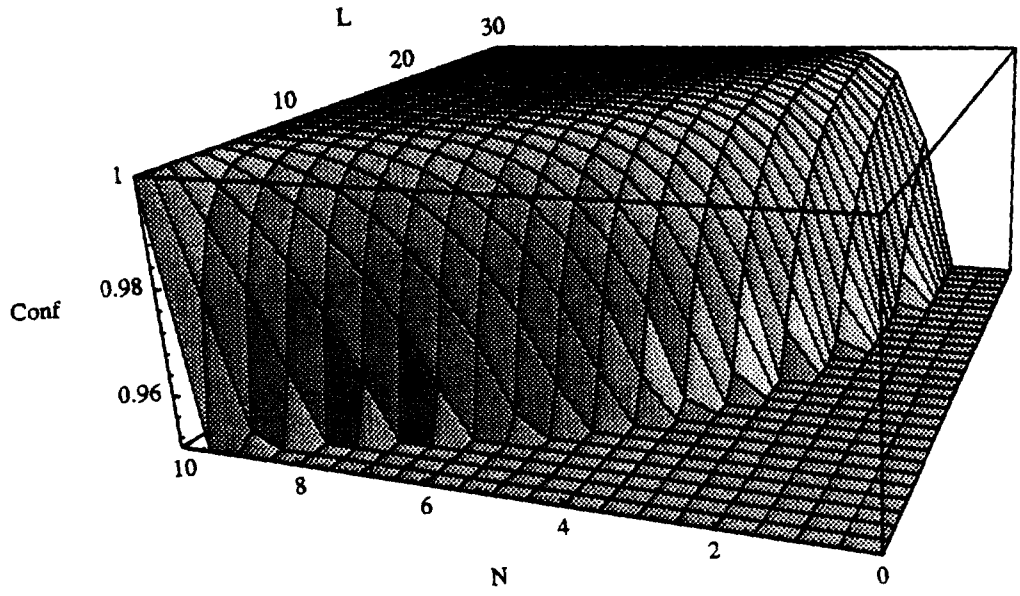


Figure 3.10(b). Undetected-errors intervals, C vs. N ($C \geq 0.95$).

$\lambda = 0.829$ error/min, N=10, M=10

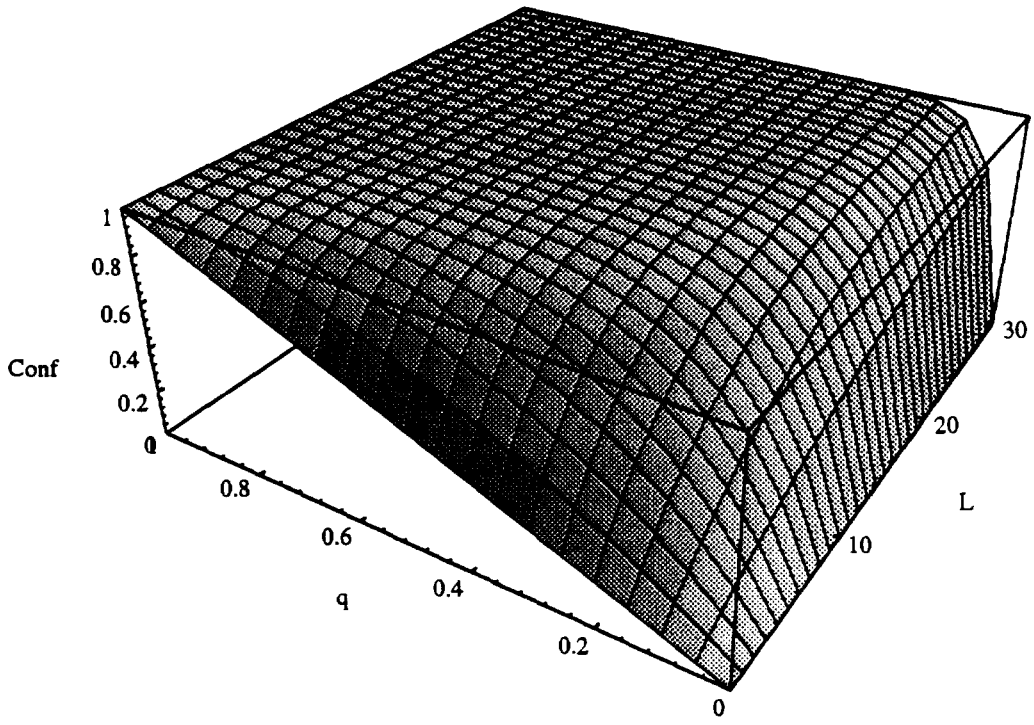


Figure 3.11(a). Undetected-errors intervals, C vs. q ($0 \leq C \leq 1$).

$\lambda = 0.829$ error/min, N=10, M=10

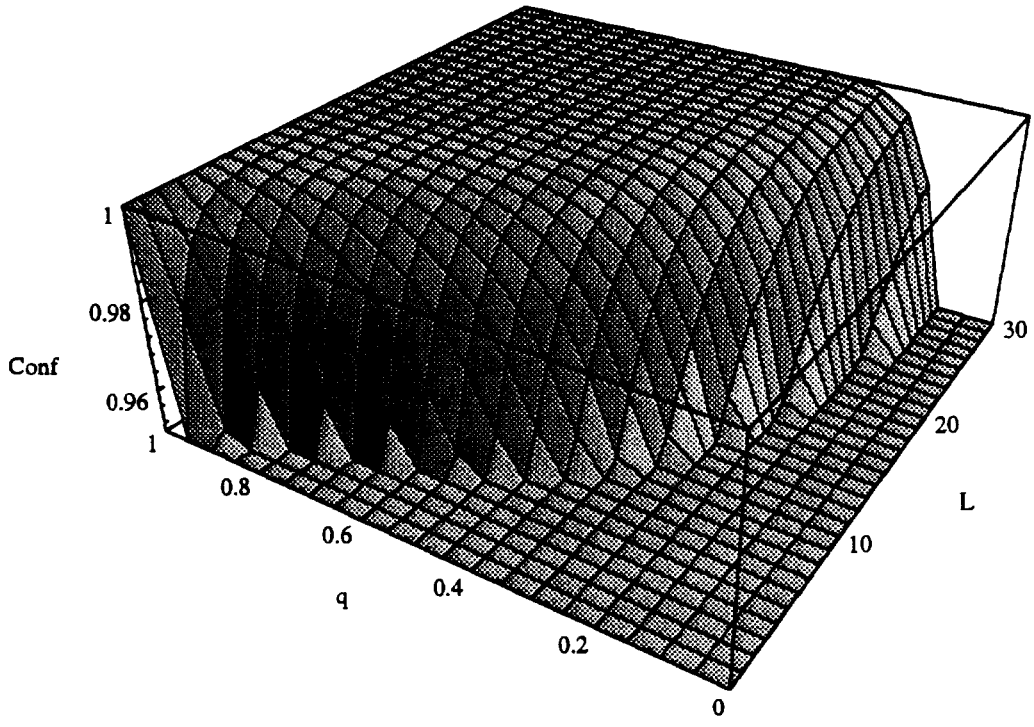


Figure 3.11(b). Undetected-errors intervals, C vs. q ($C \geq 0.95$).

need for either precise values of q or high checking ratios. Yet marked improvement on the amounts of output to suspect upon error detection will be made in the following two chapters, in which PACED is applied to processor array architectures. It will become evident that through cooperation among the constituent PEs, the amounts of output to suspect can be significantly reduced.

3.3. Error Coverage

The *error coverage* of the PACED technique is the probability that if an error occurs then it will be detected. In a single processor using PACED, the error coverage can be estimated as qN/M: the processor performs checks N/M of the time, and each check has detection probability q. Even with perfect detection ($q = 1$), it is clear that low values of the checking ratio N/M would have low error coverage.

Consider, however, the undetected-errors intervals of length L calculated in the previous section. When an error is detected, the backward interval will, in effect, "detect" all the undetected errors in that interval, by casting them under suspicion; the forward interval would similarly "detect" any future undetected errors. Hence, an error can only escape "detection" if no other errors are actually detected in the time intervals of length L before and after it.

The following theorem determines an expression for the estimated error coverage of a single processor using PACED. It first finds the probability that an error goes undetected; this probability depends on the average number of errors in an interval of length L. As an approximation, there will be an average of L/μ errors in an L-length interval, where μ is the mean interarrival time and L is given by Theorem 3.2 or 3.3. If error arrivals are modeled by a two-phase

hyperexponential distribution (as in Example 3.1) of the form $\hat{f}(t) = \alpha\ \lambda_1\ e^{-\lambda_1 t} + (1-\alpha)\ \lambda_2\ e^{-\lambda_2 t}$, the mean interarrival time μ can be found as follows.

$$\mu = \int_0^\infty t\ \hat{f}\ (t)\ dt$$

$$= \int_0^\infty t\ (\alpha\lambda_1 e^{-\lambda_1 t} + (1-\alpha)\lambda_2 e^{-\lambda_2 t})\ dt$$

$$= \frac{\alpha}{\lambda_1} + \frac{(1-\alpha)}{\lambda_2}$$

If error arrivals are modeled by a simple exponential distribution of the form $\hat{f}\ (t) = \lambda\ e^{-\lambda t}$, the mean interarrival time $\mu = 1/\lambda$.

THEOREM 3.4: Let a single processor use PACED where $1 \leq N \leq M$ and the CED technique has detection probability $q \leq 1$. The estimated error coverage of the processor is given by

$$1 - (1 - q\,\frac{N}{M})^{\frac{2L}{\mu}+1}\ .$$

PROOF: The probability that an error goes undetected is the probability that the error itself is not detected, and that no other errors are detected in the time intervals of length L before and after the error. The following expression for the probability of an undetected error has terms for each of the following conditions: 1) no errors are detected in a time interval of length L prior to an undetected error; 2) the error is itself undetected; and 3) no errors are detected in an interval of length L after an undetected error.

$$\Pr\{\text{error undetected}\} = (1 - q\,\frac{N}{M})^{\frac{L}{\mu}} \cdot (1 - q\,\frac{N}{M}) \cdot (1 - q\,\frac{N}{M})^{\frac{L}{\mu}}$$

$$= (1 - q\frac{N}{M})^{\frac{2L}{\mu}+1}$$

If an error is detected in either of the two L-length intervals, then when the two L-length intervals are taken around the error detection according to Theorems 3.2 and 3.3, the error in question will be "detected" in the sense that the outputs it could have corrupted will be suspected as possibly erroneous. Since Pr{error detected} = 1 – Pr{error undetected}, then

$$\text{estimated error coverage} = 1 - (1 - q\frac{N}{M})^{\frac{2L}{\mu}+1} . \qquad \square$$

EXAMPLE 3.5: For a single processor using PACED, let the CED technique have perfect detection (q = 1), N/M = 0.1, C = 0.99, and the error interarrival time be modeled by the two-phase hyperexponential distribution given in Example 3.1, *viz.*, $\hat{f}(t) = 0.88(0.829\ e^{-0.829t}) + 0.12(0.012\ e^{-0.012t})$. This gives μ = 11.1 min. From Theorems 3.2 and 3.3, L = 86.9 min, so the expected number of error arrivals in time L is L/μ = 7.8. The estimated error coverage is then $1 - (1 - 0.5)^{16.6} = 0.99993$. Hence, with only 10% checking, an estimated error coverage greater than 99% can be achieved.

Figure 3.12 plots the estimated error coverage for the above error arrival distribution as a function of N/M when q = 1, M = 10, and C = 0.99. It can be seen that the coverage is very high: over 99.99% for all values of $N/M \geq 0.1$. This makes sense for values of N/M close to 1, since an error is more likely to be detected when more checking is performed. When N/M is small, high coverage is still obtained because the length L of the undetected-errors interval is very long. Thus, many error arrivals would be expected to occur in a time interval of length $2L$. For any given error, then, it would be quite likely that at least one error in an interval of length

$2L$ would be detected, leading to the "detection" of the error by casting suspicion on outputs produced at the time of the error. □

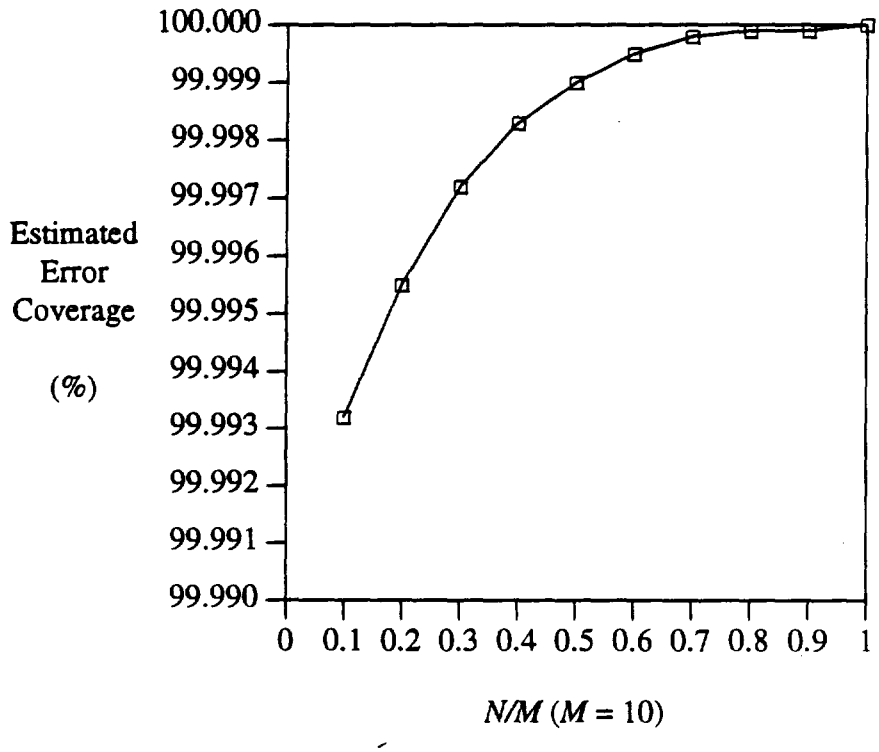


Figure 3.12. Single processor estimated error coverage, $q = 1$, $\mu = 11.1$ min/err.

CHAPTER 4.

PACED IN A LINEAR ARRAY

This chapter considers a unidirectional linear processor array composed of V linearly connected PEs (Figure 4.1). Inputs enter at the top and left; outputs are produced at the bottom and right. Data flow only from left to right and from top to bottom. Such arrays have been used to implement algorithms such as FFT processing [39], matrix computations [40], and image edge detection [41]. For two PEs in the array PE_i and PE_j, if $i < j$, then PE_i is *upstream* of PE_j and PE_j is *downstream* from PE_i.

When PACED is used in this array, checking patterns can be designed so that PEs check the unchecked computations of upstream PEs. Each PE_i in the array may have its own separate values of M and N: M_i and N_i. The offset parameter O_i, introduced in Chapter 2, determines the pattern of checking that appears in the array. It is implemented as an offset into a PE_i's $CS_{M,N}$ array and governs at what point in its M_i-cycle checking sequence to begin. With

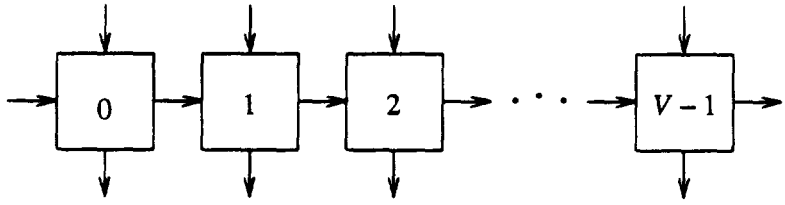


Figure 4.1. A V-PE unidirectional linear processor array.

PACED applied to the linear array, this chapter will study the confidence to place on array outputs upon error detection, the error coverage, and the performance of the array.

Having investigated the confidence in a single processor's outputs under PACED, the confidence in the outputs from a unidirectional linear processor array using PACED upon error detection will be examined first. The confidence analysis is based on three assumptions. 1) All communication channels in the array are fault-free. 2) If an erroneous array output is produced by a PE, an erroneous propagating output will also be produced and sent downstream (e.g., by using the AN-code [42]: see Section 4.5.1.) 3) PEs are code-disjoint: use of erroneous inputs or state values causes erroneous PE outputs to propagate.

Assumption 2) ensures that no erroneous array outputs can be produced without the possibility that a downstream PE will detect a propagated error. To ensure that errors are propagated, each PE may produce an additional propagating check output, generated from all of its array outputs using some code-preserving operation (e.g., the sum of all its array outputs if the AN-code is used). This additional check output can be piggy-backed onto an existing data message to avoid increasing the message traffic. Any downstream PE that is checking will check this output as well, and then clear it; any downstream PE that is not checking will simply include the output when calculating its own check output to send further downstream. Since errors are of interest only if they affect PE outputs, Assumption 3) ensures that detecting errors at PE outputs will catch input and state errors as well.

Two time intervals are determined in which to suspect linear array outputs upon error detection. The analysis begins with a discussion of the error detection latency in the array and the error propagation distance. This distance is used in the determination of the backward time

interval. After the forward time interval is determined, the error coverage in the linear array is examined, followed by a study of the performance of the linear array using both simulations and experiments.

4.1. Error Detection Latency

If an error occurs, of interest is the error detection latency in the array. The *error detection latency*, L, is the number of computation cycles through which an output is propagated until it is detected. The maximal value of L is denoted by L_{max}.

Lemma 4.1 determines the detection latency for errors created at any given PE_i for each of its $M_i - N_i$ unchecked cycles in one M_i-cycle period. Here and in the remainder of this chapter, the CED scheme in all PEs is assumed to have perfect detection (i.e., $q = 1$) and the checking pattern is assumed to be set by $O_i = (N_i \cdot i) \bmod M_i$. This choice of checking pattern has been shown to minimize L_{max} in linear arrays [19].

LEMMA 4.1: Given a V-PE unidirectional linear processor array using PACED with perfect detection ($q = 1$), let $M_i = M$, $N_i = N$, and $1 \le N \le M$. Using $O_i = (Ni) \bmod M$, the detection latency of an error created in the unchecked cycle r at PE_i, L_r, is $\lceil (M - r)/N \rceil$, where $N \le r \le M - 1$ and $i < V - L_{max}$. The maximum error detection latency in the array, L_{max}, is $\lceil (M - N)/N \rceil$, for all PE_i such that $i < V - L_{max}$.

PROOF: By design of the checking pattern, if $CS_{M,N}[r]$ is the checking activity at PE_i in some computation cycle c, then $CS_{M,N}[(r + y(N - 1) + z) \bmod M]$ is the checking activity at PE_{i+y} in cycle $c + z$. With perfect detection, errors only propagate through unchecked cycles, so the proof only considers $N \le r \le M - 1$.

If an error occurs at PE_i during its N^{th} cycle, it will go undetected: this cycle is unchecked ($CS_{M,N}[N] = 0$). In the next cycle, the error will propagate to PE_{i+1} and be detected if $CS_{M,N}[(2N) \bmod M] = 1$ (i.e., if PE_{i+1} is checking). If $CS_{M,N}[(2N) \bmod M] = 0$, then the error will propagate to PE_{i+2} in the next cycle, where it will be detected if $CS_{M,N}[(3N) \bmod M] = 1$, and so on.

The latency of detection of this error, L_N, is the number of computation cycles required for the error to reach a checked cycle. In terms of the checking sequence, L_N is the smallest integer number of N-bit hops needed to reach s such that $CS_{M,N}[s] = 1$ (i.e., $0 \le s \le N-1$) from N, where $CS_{M,N}[N] = 0$. This is a distance of $M - N$ bits.

$$L_N \cdot N \ge M - N$$

$$L_N = \lceil (M-N)/N \rceil$$

Similarly, L_{N+1}, the latency of an error created during the $N+1^{st}$ cycle (an unchecked cycle, since $CS_{M,N}[N+1] = 0$), is $\lceil (M-N-1)/N \rceil$. In general, an error created during cycle r (an unchecked cycle: $CS_{M,N}[r] = 0$) will have latency $L_r = \lceil (M-N-(r-N))/N \rceil = \lceil (M-r)/N \rceil$, $N \le r \le M-1$. Clearly, $L_N \ge L_{N+1} \ge \cdots \ge L_{M-1}$. Therefore, the maximum error detection latency, L_{max}, is L_N: $L_{max} = L_N = \lceil (M-N)/N \rceil$.

This analysis applies to all PEs in the array except the end elements, PE_i where $i \ge V - L_{max}$. At these PE_i, an error may propagate undetected out of the array since for these PE_i there are fewer than L_{max} PEs downstream. □

EXAMPLE 4.1: Figure 4.2 shows the checking pattern in a 7-PE unidirectional array as it begins work on a problem, with $M_i = 5$, $N_i = 2$, and $O_i = (2i) \bmod 5$; $CS_{5,2} = (1, 1, 0, 0, 0)$.

Computation cycles are shown on the vertical axis; each row shows the checking activity in the array during a cycle. Notice that the checking pattern sets up waves of checked cycles that advance upstream over time to catch propagating errors.

In the figure, L_2 for an error created at PE_2 in cycle 10 (marked by *) is $\lceil (5-2)/2 \rceil = 2$: the error would be detected two cycles later, by PE_4 in cycle 12 (labeled L_2). For an error created at PE_2 in cycle 11 (marked by o), $L_3 = \lceil (5-3)/2 \rceil = 1$: the error would be detected by PE_3 in cycle 12 (labeled L_3). For an error created at PE_2 in cycle 12 (‡), $L_4 = \lceil (5-4)/2 \rceil = 1$, since the error would be detected by PE_3 in cycle 13 (labeled L_4). Finally, $L_{max} = L_2 = 2$. □

computation cycle	PE 0	1	2	3	4	5	6
0	×	/	/	/	/	/	/
1	×	—	/	/	/	/	/
2	—	—	—	/	/	/	/
3	—	—	×	×	/	/	/
4	—	×	×	—	—	/	/
5	×	×	—	—	—	×	/
6	×	—	—	—	×	×	—
7	—	—	—	×	×	—	—
8	—	—	×	×	—	—	—
9	—	×	×	—	—	—	×
10	×	×	*	—	—	×	×
11	×	—	o	—	×	×	—
12	—	—	‡	L_3	L_2	—	—
13	—	—	×	L_4	—	—	—
14	—	×	×	—	—	—	×
15	×	×	—	—	—	×	×

/ = PE idle — = PE doing task × = PE doing checked task

Figure 4.2. Checking pattern in a 7-PE array.

To prevent errors from propagating out of the linear array and escaping detection, a modification to PACED can be applied in which the last PE in the array, PE_{V-1}, performs 100% checking. This variation of PACED, PACED′, can be implemented by duplicating PE_{V-1} in hardware; this can prevent PE_{V-1} from becoming a performance bottleneck. Only the normal PACED performance costs would then be incurred. A less hardware-expensive implementation of PACED′ might monitor the outputs of PE_{V-1} using a hardware code checker.

4.2. Error Propagation Distance

The following lemma gives an expression for the maximum number of unchecked cycles through which a detected error could have propagated. This result will be used in Theorem 4.1 to determine the amount of previously produced output to suspect as possibly erroneous from each PE in the linear array, upon error detection.

LEMMA 4.2: Given a V-PE unidirectional linear processor array using PACED with perfect detection ($q = 1$), let $M_i = M$, $N_i = N$, and $1 \le N \le M$. Using $O_i = (Ni) \bmod M$, an error detected by $CS_{M,N}[r]$ at PE_i, $0 \le r \le N - 1$, propagated through at most D_r unchecked cycles, where $D_r = \min(i, \lceil (M + r + 1)/N \rceil - 2)$.

PROOF: Let $CS_{M,N}[0]$ at PE_i detect an error in computation cycle c. The checking activity at PE_{i-1} during cycle $c - 1$ is $CS_{M,N}[(-N) \bmod M]$. The maximum number of unchecked cycles through which the detected error may have propagated, D_0, is the number of computation cycles required to reach a checked cycle, minus 1, counting backwards in time. In terms of the checking sequence, $D_0 + 1$ is the smallest integer number of N-bit hops needed to reach $CS_{M,N}[r]$, $0 \le r \le N - 1$, from $CS_{M,N}[0]$. This is a distance of $M - N + 1$ bits.

$$(D_0 + 1)N \geq M - N + 1$$

$$D_0 = \lceil (M - N + 1)/N \rceil - 1$$

$$= \lceil (M + 1)/N \rceil - 2$$

Similarly, $D_1 = \lceil (M + 2)/N \rceil - 2$. In general, $D_r = \lceil (M + r + 1)/N \rceil - 2$, $0 \leq r \leq N - 1$. For PEs near the beginning of the array, there may be fewer than D_r PEs through which the error propagated. Hence, at PE_i, $D_r = \min(i, \lceil (M + r + 1)/N \rceil - 2)$, for $0 \leq r \leq N - 1$. □

EXAMPLE 4.2: Using the array of Example 4.1 (Figure 4.2), D_0 for an error detected at PE_3 at computation cycle 12 is $\lceil (5 + 0 + 1)/2 \rceil - 2 = 1$, because PE_1 checked computation cycle 10. For an error detected at PE_3 at computation cycle 13, $D_1 = \lceil (5 + 1 + 1)/2 \rceil - 2 = 2$, because PE_0 checked computation cycle 10. □

4.3. Suspected Outputs

Upon error detection, outputs produced both in the recent past and the near future should be suspected as possibly erroneous. The following theorem determines which of the previously produced outputs to suspect when an error is detected by a PE in the linear array; Theorem 4.2 considers which of the future outputs to suspect.

THEOREM 4.1: Given a V-PE unidirectional linear array using PACED with perfect detection ($q = 1$), let $M_i = M$, $N_i = N$, $1 \leq N \leq M$, and $O_i = (Ni) \bmod M$. If PE_i detects an error at its r^{th} checked cycle in computation cycle c, $0 \leq r \leq N - 1$, then the output from PE_i in c should be suspected as possibly erroneous. In addition, the outputs produced by PE_{i-k} in cycle $c - k$, for $1 \leq k \leq D_r$, should be suspected. All other unsuspected, previously produced outputs can be

trusted with a confidence of 1, unless a later error detection makes it necessary to suspect them.

PROOF: By Lemma 4.2, the detected error propagated through at most D_r unchecked cycles to reach PE_i. Thus, the error was created at some PE_{i-k} in a cycle $c-(k+y)$, where $1 \leq k \leq D_r$ and $y = 1, 2, 3, \cdots$.

Figure 4.3 shows the checking activity in a 10-PE array in the midst of a problem, with $M = \infty$ and $N = 2$. The **X** marks an error detection at PE_5 in cycle c and the *s mark the D_r cycles through which an error may have propagated to reach PE_5.

Suppose that the error had occurred at PE_4 in cycle $c-2$, $c-3$, or $c-4$. The error would have been detected by PE_6 in cycle c, $c-1$, or $c-1$, respectively. Suppose the error had occurred at PE_3 in cycle $c-3$, $c-4$, or $c-5$. This error would have been detected by PE_6 in cycle c or $c-1$, or by PE_7 in cycle $c-1$, respectively.

computation cycle	PE 0	1	2	3	4	5	6	7	8	9
$c-12$	—	—	—	—	—	—	—	—	—	—
$c-11$	—	—	—	—	—	—	—	—	—	—
$c-10$	—	—	—	—	—	—	—	—	—	—
$c-9$	—	—	—	—	—	—	—	—	—	—
$c-8$	—	—	—	—	—	—	—	—	—	—
$c-7$	—	—	—	—	—	—	—	—	—	—
$c-6$	—	—	—	—	—	—	—	—	—	—
$c-5$	*	—	—	—	—	—	—	—	—	—
$c-4$	—	*	—	—	—	—	—	—	—	×
$c-3$	—	—	*	—	—	—	—	—	×	×
$c-2$	—	—	—	*	—	—	—	×	×	—
$c-1$	—	—	—	—	*	—	×	×	—	—
c	—	—	—	—	—	**X**	×	—	—	—

Figure 4.3. Error propagation in a 10-PE array.

In general, any error created at PE_{i-k} before cycle $c-k$ would either have been detected by cycle c (and the appropriate outputs already suspected), or gone undetected (if the error propagated out of the array). This is a result of the checking pattern, in which each PE_i performs its last checked cycle ($CS_{M,N}[N-1]$) during the same computation cycle that PE_{i-1} performs its first checked cycle ($CS_{M,N}[0]$). Hence, only the outputs from PE_{i-k} in cycles $c-k$ need be suspected, $1 \le k \le D_r$, as well as that from PE_i in c. All other unsuspected, previously produced outputs can be trusted with a confidence of 1, unless a later error detection makes it necessary to suspect them. □

EXAMPLE 4.3: Figure 4.4 shows the checking pattern in a 10-PE unidirectional linear array in the midst of a problem, with $M_i = 13$, $N_i = 3$, and $O_i = (3i) \bmod 13$. Let PE_8 detect an error in cycle c (**X** in the figure) by check $CS_{13,3}[2]$. The output from PE_8 in cycle c should be suspected. Also, since $D_2 = \lceil (13+2+1)/3 \rceil - 2 = 4$ (Lemma 4.2), the outputs of PE_7, PE_6, PE_5, and PE_4 in cycles $c-1$, $c-2$, $c-3$, and $c-4$, respectively, (marked by *) should be suspected as possibly erroneous, by Theorem 4.1. All other outputs generated up through cycle c can be trusted with a confidence of 1, unless a later error detection makes it necessary to suspect them.□

Section 4.1 mentioned a modification of PACED, PACED′, which eliminates the possibility that errors escape undetected from the linear array. Besides this boon, PACED′ also has the advantage that, upon error detection, only outputs produced just prior to the detection need be suspected. Since all errors are eventually detected, there is no need to suspect outputs produced after an error detection unless a later error detection warrants it. However, future outputs have

computation cycle	PE 0	1	2	3	4	5	6	7	8	9
$c-12$	×	×	—	—	—	—	—	×	—	—
$c-11$	×	—	—	—	—	—	×	×	—	—
$c-10$	×	—	—	—	—	—	×	—	—	—
$c-9$	—	—	—	—	—	×	×	—	—	—
$c-8$	—	—	—	—	—	×	—	—	—	—
$c-7$	—	—	—	—	×	×	—	—	—	—
$c-6$	—	—	—	—	×	—	—	—	—	—
$c-5$	—	—	—	×	×	—	—	—	—	—
$c-4$	—	—	—	×	*	—	—	—	—	×
$c-3$	—	—	×	×	—	*	—	—	—	×
$c-2$	—	—	×	—	—	—	*	—	×	×
$c-1$	—	×	×	—	—	—	—	*	×	—
c	—	×	—	—	—	—	—	×	**X**	—

Figure 4.4. Suspected previously produced outputs, 10-PE array.

to be suspected if normal PACED is used in the linear array and an error is detected at one of the end elements PE_i, where $i \geq V - L_{max}$. Theorem 4.2 determines which future outputs to suspect if an error is detected at one of these PEs.

THEOREM 4.2: Given a V-PE unidirectional linear array using PACED with perfect detection ($q = 1$), let $M_i = M$, $N_i = N$, $1 \leq N \leq M$, and $O_i = (Ni) \bmod M$. If $PE_{V-L_{max}+i}$ detects an error at its r^{th} checked cycle in computation cycle c, where $0 \leq r \leq N-1$ and $0 \leq i \leq L_{max} - 1$, then the following outputs should be suspected as possibly erroneous.

a) If $(r + (L_{max} - 1 - i)N + k) \bmod M \geq N$, then the outputs from $PE_{V-L_{max}+i+j}$ in cycle $c + j + k$ should be suspected, where $0 \leq j \leq L_{max} - 1 - i$; if $r < N - 1$, then $k = 0$, otherwise $0 \leq k \leq M - N$ ($r = N - 1$).

b) All output from PE_{V-1} in cycles $c + L_{max} - 1 - i$ until its next checked cycle should be suspected.

All other unsuspected, future outputs can be trusted with a confidence of 1, unless a future error detection makes it necessary to suspect them.

PROOF: By use of $O_i = (Ni) \bmod M$ in the linear array, when PE_i in cycle c performs its r^{th} checked task ($CS_{M,N}[r] = 1$), then PE_{i+y} in cycle $c + z$ will perform $CS_{M,N}[(r + y(N-1) + z) \bmod M]$.

Now, let $PE_{V-L_{max}+i}$ detect an error in cycle c by $CS_{M,N}[r]$, for $0 \le i \le L_{max} - 1$. These $PE_{V-L_{max}+i}$ are those PEs that could create errors that propagate undetected out of the array. The detected error will propagate to PE_{V-1} in cycle $c + L_{max} - 1 - i$. In that cycle, if PE_{V-1} is not checking (i.e., $(r + (L_{max} - 1 - i)N) \bmod M \ge N$), then this error will propagate out of the array and outputs from all PEs and cycles through which the error propagated should be suspected as possibly erroneous. That is, if $(r + (L_{max} - 1 - i)N) \bmod M \ge N$, then the output from $PE_{V-L_{max}+i+j}$ in cycle $c + j$ should be suspected, $0 \le j \le L_{max} - 1 - i$. If $PE_{V-L_{max}+i}$ will check at the next cycle $c + 1$, then this gives part **a)** when $r < N - 1$ ($k = 0$).

If $r = N - 1$ ($PE_{V-L_{max}+i}$ won't check in cycle $c + 1$), then as in the above case when $r < N - 1$, if $(r + (L_{max} - 1 - i)N + k) \bmod M \ge N$, then the output from $PE_{V-L_{max}+i+j}$ in cycle $c + j + k$ should be suspected, where $0 \le j \le L_{max} - 1 - i$ and $k = 0$. In addition, for each of the next $M - N$ unchecked cycles, errors may propagate out of the array. This is likely since an error has already been detected at $PE_{V-L_{max}+i}$ and the fault may still be active while that PE is not checking. The additional outputs to suspect depend upon whether PE_{V-1} is not checking when

the errors arrive there. That is, for each cycle $c+k$, $1 \le k \le M-N$, if $(r + (L_{max} - 1 - i)N + k)$ mod $M \ge N$, then the output from $PE_{V-L_{max}+i+j}$ in cycle $c+j+k$ should be suspected, for $0 \le j \le L_{max} - 1 - i$. This completes part **a)** when $r = N - 1$.

Once an error propagates to PE_{V-1} while it is not checking, all of its outputs until its next checked cycle should be suspected as possibly erroneous since its outputs are not checked by any other PE. Hence, all of the outputs from PE_{V-1} in cycles $c + L_{max} - 1 - i$ (the earliest that the error, first detected at $PE_{V-L_{max}+i}$ in cycle c, could corrupt PE_{V-1}) until its next checked cycle should be suspected as possibly erroneous. This gives part **b)** in the statement of the theorem.

All other unsuspected, future outputs from the array can be trusted with a confidence of 1, unless a future error detection makes it necessary to suspect them. □

EXAMPLE 4.4: Figure 4.5 shows the 10-PE linear array of Example 4.3, in which $M_i = 13$, $N_i = 3$, and $O_i = (3i)$ mod 13; by Lemma 4.1, $L_{max} = 4$. PE_6 has detected an error at check $r = 2$ in cycle c (marked **X** in the figure). Using Theorem 4.2, the future outputs to suspect will be determined. Since PE_9 will not check cycle $c+3$ (since $(2 + (4 - 1 - 0)3)$ mod $13 \ge 3$), then the outputs from the following PEs should be suspected: PE_7 in cycle $c+1$, PE_8 in cycle $c+2$, and PE_9 in cycle $c+3$.

As PE_6 detected the error at its N^{th} check ($r = N - 1$), its next $M - N$ cycles may also create undetected errors. But PE_9 begins checking in cycle $c+5$, so only the outputs from the following PEs need be suspected: PE_6 in cycle $c+1$, PE_7 in cycle $c+2$, PE_8 in cycle $c+3$, and PE_9 in cycle $c+4$. In addition, the outputs from PE_9 in cycles $c+3$ to $c+4$ should be suspected (both already are), since PE_9 doesn't begin checking again until cycle $c+5$. The outputs to

suspect are marked * in the figure, plus the site of the detection (**X**). All other unsuspected, future outputs can be trusted with a confidence of 1, unless a later error detection makes it necessary to suspect them. □

The detection of an error by one of a PE's N checks leads to two static patterns of outputs to suspect as possibly erroneous: one for the previously produced outputs and one for the future outputs. For example, Figure 4.4 shows the pattern of previous outputs to suspect if $M_i = 13$, $N_i = 3$, $O_i = (3i) \bmod 13$, and $CS_{13,3}[2]$ detects an error. Figure 4.5 shows the pattern of future outputs to suspect for the same parameter values when the error is detected at $PE_{V-L\,max}$. For

computation cycle	PE 0	1	2	3	4	5	6	7	8	9
$c-6$	—	—	×	—	—	—	—	—	×	×
$c-5$	—	×	×	—	—	—	—	—	×	—
$c-4$	—	×	—	—	—	—	—	×	×	—
$c-3$	×	×	—	—	—	—	—	×	—	—
$c-2$	×	—	—	—	—	—	×	×	—	—
$c-1$	×	—	—	—	—	—	×	—	—	—
c	—	—	—	—	—	×	**X**	—	—	—
$c+1$	—	—	—	—	—	×	*	*	—	—
$c+2$	—	—	—	—	×	×	—	*	*	—
$c+3$	—	—	—	—	×	—	—	—	*	*
$c+4$	—	—	—	×	×	—	—	—	—	*
$c+5$	—	—	—	×	—	—	—	—	—	×
$c+6$	—	—	×	×	—	—	—	—	—	×
$c+7$	—	—	×	—	—	—	—	—	×	×
$c+8$	—	×	×	—	—	—	—	—	×	—
$c+9$	—	×	—	—	—	—	—	×	×	—
$c+10$	×	×	—	—	—	—	—	×	—	—
$c+11$	×	—	—	—	—	—	×	×	—	—

Figure 4.5. Suspected future outputs, 10-PE array.

these parameter values, there would be a total of $3(L_{max} + 1)$ possible patterns of outputs to suspect: three patterns for previous outputs (one for each check $CS_{13,3}[r]$, $0 \le r \le 2$) and $3L_{max}$ patterns for future outputs (one for each check $CS_{13,3}[r]$, $0 \le r \le 2$ at each $PE_{V-L_{max}+i}$, $0 \le i \le L_{max} - 1$). These patterns can be computed once for the array and stored, indexed by r and i. Upon error detection, given the PE and which of its checks detected the error, the outputs to suspect can be determined with no extra computation by simply using the template stored for that check.

The amount of output to suspect upon error detection in the linear array is much less than that necessary upon error detection in a single processor using PACED. Example 3.4 showed that using the undetected-errors intervals in the single processor, 18.8 min worth of outputs (9.4 min both prior and subsequent to an error detection) should be suspected. In the linear array, the outputs from perhaps only a few tens of computation cycles need be suspected; with cycles times in the range of 15 μs to 20 ms in VSLI array implementations [41], this means on the order of just one second's output need be suspected. By using the ability of PEs to check other PE outputs, PACED can give high confidence in most array outputs upon error detection with less than continuous checking.

4.4. Error Coverage

If it is assumed that errors occur uniformly distributed among the constituent PEs of the linear array, an estimate of the error coverage can be made. This assumption can be valid in arrays with homogeneous PEs running the same algorithm: no PE would be more or less susceptible to errors than any other PE. In any M consecutive cycles in the array, each PE will have

completed one pass through its $CS_{M,N}[]$ array. Hence, one M-cycle period has the same coverage as any other M-cycle period, so it suffices to examine a single such period.

In one M-cycle period in a V-PE linear array, there are MV potential sites at which errors may occur: one for each PE of the array, in each cycle. Since it is assumed that errors propagate through the array and are not masked, when normal PACED is applied to the array, only some of these sites could lead to the propagation of undetected errors out of the array, if an error were to occur. (When PACED′ is used, the estimated coverage is 100% for all values of N/M, since no errors can escape undetected from the array.) By counting these sites and dividing by the total number of potential sites, an estimate of the error coverage can be made.

Figure 4.6 shows the estimated error coverage for a 16-PE linear array as a function of N/M, when $M = 10$ and $q = 1$. It can be seen that even for small values of N/M, the error coverage is quite high (greater than 70% for $N/M = 0.1$). The coverage climbs quickly as N/M increases, so that any checking ratio greater than 0.4 will have an estimated error coverage greater than 95%. The cooperation among the PEs that allows propagated errors to be detected causes this rise in coverage for small N/M. Hence, low values of the checking ratio can yield high error coverage. This result is promising, as it allows the possibility of low checking ratios, and thus, low performance cost, while still maintaining good error coverage.

4.5. Performance

The performance of linear processor arrays using PACED was studied in two ways. First, the performance costs were estimated by using an algorithm-independent, simulation-based, analysis model that was written in C to study the effect of PACED when used in linear, square,

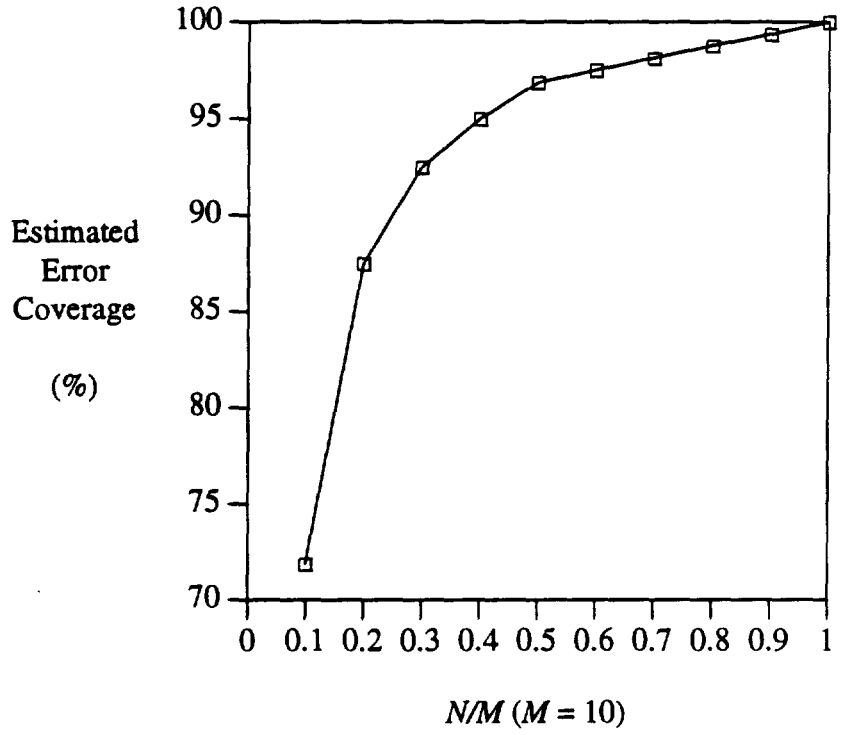


Figure 4.6. Estimated error coverage for a 16-PE linear array.

and triangular processor array architectures. The simulator uses mean execution times required by basic arithmetic operations, so that the activity in the array PEs can be simulated without actually being performed. This event-driven, reduced simulation gives an estimate of the completion times for algorithms with and without the use of PACED, allowing the PACED overhead to be determined.

Second, full simulations of a linear array performing an image processing edge-detection algorithm were run on an Intel iPSC/2 hypercube, to obtain more accurate values of the performance costs to expect when using PACED. The results from these two performance analyses are presented in the following two subsections.

4.5.1. Simulation model

The inputs to the simulation model include the dimension(s) of one of the modeled architectures, the mean task and check times for the PEs, the values of M_i, N_i, and O_i PACED parameters, and the desired length of simulation (the number of computational cycles required by the PE producing the final output). Given these inputs, the model can estimate the performance costs incurred through the use of PACED in the array.

The *task time* is the user's estimate of the mean time that a PE requires to complete a task. Similarly, the *check time* is the mean time that a PE requires to complete the CED for a task. It is assumed that the deviations from these means are small. (This assumption has been verified from actual simulations, described in the next section. However, in cases where the assumption is not valid, the simulation results will be more inaccurate.) These times are determined by analyzing the implementations of the task and check algorithms. If the array consists of more than one type of PE, task and check times for each type of PE require specification. Communication costs are not explicitly modeled; they can, however, be incorporated into the mean task and check times.

The model uses these parameters to simulate the activity of the array PEs without performing the computations specified by the algorithm. The partial-simulation saves time and affords the model algorithm independency. Though not as accurate as a full simulation, this reduced simulation model is intended to provide good results at a low computation cost.

EXAMPLE 4.5: The simulation model was used to estimate the performance costs of using PACED in a linear unidirectional array running an image edge-detection algorithm [41]. Two

CED schemes were considered in determining the mean task and CED times: RESO and AN-coding. Briefly, in RESO-*k*, each arithmetic operation is performed twice: the first normally, the second using *k*-bit arithmetic-shifted operands to produce a bit-shifted result. Different amounts of shifting can be used, depending on the operation, to obtain maximum error coverage. For these experiments, the basic RESO recommendations were employed: RESO-2 for addition and multiplication [3], and RESO-2,3 for division (i.e., 2-bit shift of numerator and 3-bit shift for denominator) [43].

In AN-coding, every operand is encoded by multiplying by the base A. All results, intermediate as well as final, must be 0 modulo A or an error has occurred [42]. For a low-cost encoding, the base A should be $2^c - 1$, where c is the number of bits needed to represent A [44].

Table 4.1 shows how the approximate mean task and CED times were determined. The first column in the table shows the different types of basic arithmetic operations that were counted for one computation cycle in each of three versions of the algorithm: the basic algorithm, one using RESO, and a third using AN-coding. The operations are integer add, integer multiply, modulo, arithmetic shift left, and two types of compare: compare register-with-memory and compare register-with-immediate. The second column shows the number of clock cycles required to perform each operation. These were taken from the Intel 80386 instruction timing data as an example ALU [45]. The columns headed "Basic" show, for the basic algorithm, how many of each operation and how many clock cycles are required for one computation cycle. The columns headed "RESO" and "AN-coding" show the same information for each of the CED versions. The penultimate row of the table shows the total clock cycles required per computation cycle of each algorithm. Since the simulator uses these numbers to control the

number of time-slice iterations performed for each computation cycle, these totals were reduced and rounded to small, whole integers, and are given in the last row.

Using the reduced clock cycles, the simulator was run for 1024 computation cycles (to simulate processing an image of 1024 rows) with $M = 10$, N varied from 0 to 10, and the detection probability $q = 1$. Figure 4.7 shows the relative completion times to be expected from running the three versions when the amount of checking is varied from 0% to 100%. The simulation predicts that the performance cost for RESO should be approximately linear with the amount of checking performed. The slight deviation from linearity arises from the initialization of the array, during which no checking is performed in many of the PEs, thereby slightly reducing the overhead due to CED.

The simulation also predicts that the cost of using AN-coding will be higher than that of RESO. This can be attributed to the large number of clock cycles required to perform a modulo

TABLE 4.1.
COMPUTATION CYCLE TIMES, EDGE DETECTION PEs.

operation		Basic		RESO		AN-coding	
type	cycles per	#	tot cycles	#	tot cycles	#	tot cycles
+	7	54	378	104	728	54	378
×	17	27	459	54	918	27	459
mod	114.5	21	2404.5	38	4351	62	7099
<<	5			108	540		
cmp w/mem	5			9	54		
cmp w/imm	2					41	82
	total cycles		3241.5		6591		8018
	reduced cycles		2		4		5

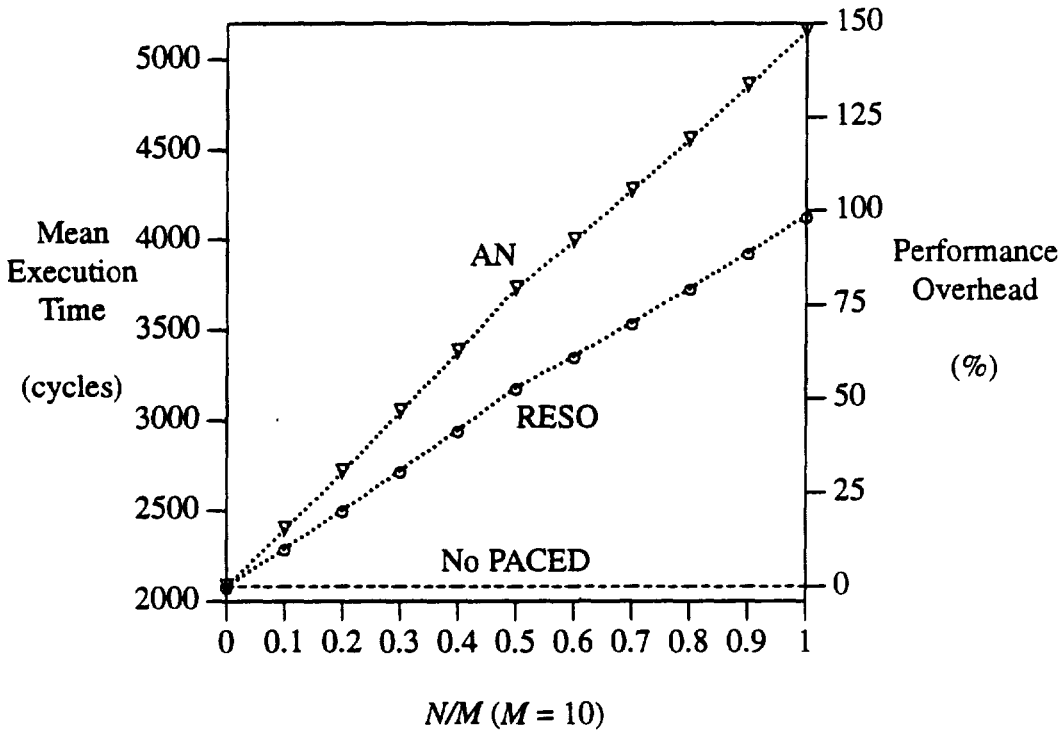


Figure 4.7. Simulated linear array performance, edge detection.

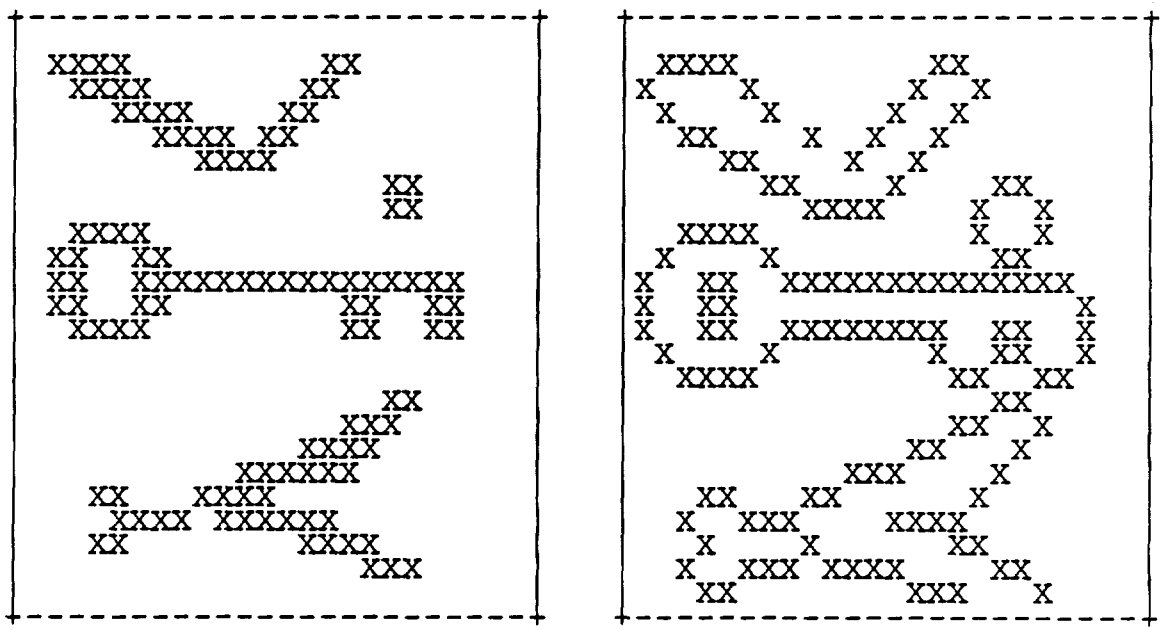
operation, which forms the crux of the checking in the AN-coding technique. All of the data for the graph were obtained from 22 runs of the simulator, which required less than 8 min. Hence, the simulator can be a valuable tool to estimate the performance costs of different CED techniques when used with PACED in the linear array. □

4.5.2. Hypercube simulations

A simulation of a linear processor array was performed on an Intel iPSC/2 hypercube using the nodes as PEs and the shortest internode connections to minimize the communication overhead. The application was the image edge-detection algorithm modeled in Section 4.5.1. Using

this algorithm, a 1-by-$V/3$ array of homogeneous PEs can process a $U \times V$ image. Figure 4.8 shows an example input image to the array and its corresponding output. (Due to the data delay through the array, the first row of output image is blank, and the last row is absent.) The algorithm repeatedly convolves a 3×3 mask with 3×3 windows of the image. First, the mask is sent by the host to each PE; three columns of image are then sent to each PE, row by row. As the data are processed, intermediate results are sent by each PE to its predecessor and outputs are sent back to the host row by row, three columns at a time from each PE.

The first simulation used all 16 nodes of the hypercube to process a 1024×48 image. Two CED techniques were employed: RESO and AN-coding. The base A of $255 = 2^8 - 1$ was used



Input image Output image

Figure 4.8. Sample input and output, edge detection algorithm.

in the experiments, so that the largest encoded numbers generated by the application would still fit in 32 bits, the size of an integer on the hypercube.

Figure 4.9, constructed from completion times of the three versions of the algorithm, shows how the performance was degraded by the use of CED in varying checking ratios *N/M*. The completion times do not include the initializations of either the host programs or the individual node programs. For each run, the individual completion times of each of the 16 nodes were averaged together. The averages from five runs were then averaged to obtain each data point on the graph. Just five runs were deemed sufficient for two reasons: 1) the greatest standard deviation for the individual node completion times was less than 0.15% of the average

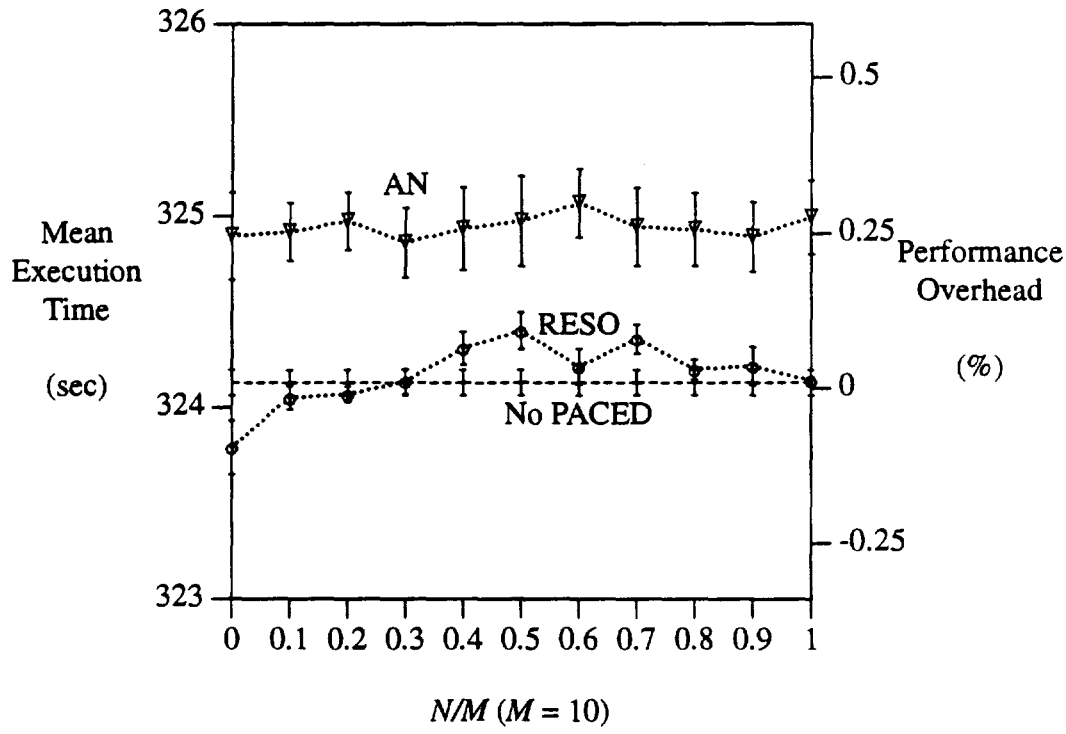


Figure 4.9. Linear array performance, edge detection.

node completion time, and 2) the greatest standard deviation for the run averages was less than 1.1% of the average run completion times. The figure displays the 95% confidence interval for each datum as a set of vertical bars above and below the point: these intervals are quite small.

From the figure, it can be seen that the use of CED, either AN-coding or RESO, in any checking ratio had little effect on the completion time. When the algorithm was checked using AN-coding, a very slight increase in the completion times is noticed ($\approx$ 0.25%), but this slight difference is probably spurious, due to slight differences in the operating conditions of the hypercube when the separate experiments were performed. It was hypothesized that communication costs in the hypercube were much larger than anticipated. Since VLSI processor arrays were developed in part to achieve great processing speed, the communication costs in such arrays should be quite small. Apparently, in this experiment, communication costs dominated the computation time so both the RESO and AN-coding results showed very little overhead.

To test this hypothesis and to obtain more accurate results of a simulated processor array using the hypercube, all inter-PE communication was removed from the algorithm's implementation and the experiments were repeated. As expected, the completion times of the application were very much smaller than when the PEs performed communication, even when a larger input image (16384×48) was used.

The results are shown in Figure 4.10. The RESO and AN-coding performances are shown as dotted lines; the right vertical side of the graph shows the performance scale. These results were closer to expectation: the performance exhibited gradual degradation as the checking ratio N/M increased, with very little degradation at $N = 0$ and rising linearly to just over 100% degradation for AN-coding and just about 100% degradation for RESO. (The 95% confidence

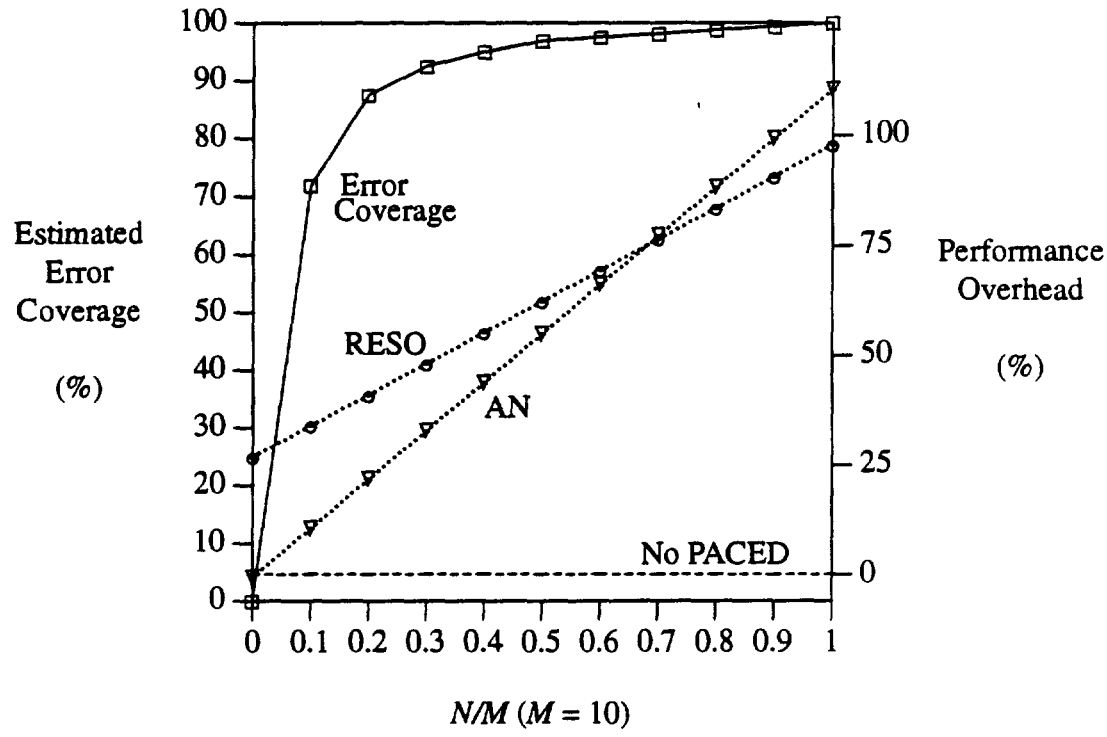


Figure 4.10. Linear array performance, edge detection, no communication.

interval bars are too small to be seen on the graph.) The RESO curve exhibits a slight degradation even when no checking is performed ($N = 0$). This is due to a slight increase in code size, as modifications were made to some operations that would normally destroy an operand needed to perform RESO.

Figure 4.10 also overlays on the same axes the estimated error coverage as a function of N/M from Figure 4.6. The left vertical side of the graph shows the scale for the error coverage in percent. Coverages over 95% can be achieved with $N/M \geq 0.4$: fairly low values of the checking ratio can yield good error coverage, for which the performance penalty can be under 50%.

From these experiments it can be concluded that the use of PACED can reduce the performance costs incurred through the use of CED in a linear processor array, while still maintaining good error coverage. A designer of such an array can trade off between performance and the amount of output to suspect when an error is detected (and thereby, the error coverage) by choosing the checking ratio *N/M*, provided a coding technique is used to allow error propagation between the PEs in the array (e.g., the AN-code for integer applications).

These experiments also validate the simulation model described in Section 4.5.1. There, the simulator had predicted that RESO would perform better under PACED than AN-coding, on a 16-node linear array running the edge detection algorithm to process a 1024-row image. The overhead at 0% checking of the CED versions was not predicted by the simulator since the mean task and check times used in the model did not reflect the code expansion required by the RESO and AN-coding versions. Also, the simulator predicted a higher-than-realized overhead for the AN-coding technique when applied continuously. However, considering the short time required to generate the simulation results, in this example, the simulator provided a fast and fairly accurate estimate of the performance costs to expect when using PACED in a linear array.

CHAPTER 5.

PACED IN A TWO-DIMENSIONAL ARRAY

The two-dimensional (2-D) processor array considered in this chapter is composed of $U \times V$ mesh-connected PEs (Figure 5.1), which accept data at their top and left inputs and send data through their right and bottom outputs. The PEs on the top and left edges of the array accept external inputs; PEs on the right and bottom edges produce external outputs. Data may only flow from left to right and from top to bottom. Note that at the onset of problem execution some PEs may be idle until their input data arrive. Such arrays have been used to implement algorithms to perform matrix operations [46], image processing [47], digital filtering [48], and polynomial evaluation [49]. For two PEs in the array $PE_{i,j}$ and $PE_{k,l}$, if $i < k$ or $j < l$, then $PE_{i,j}$ is *upstream* of $PE_{k,l}$ and $PE_{k,l}$ is *downstream* from $PE_{i,j}$.

Checking patterns in these arrays can be designed so that PEs check the unchecked computations of upstream PEs. As in the linear array, each $PE_{i,j}$ may have its own distinct $M_{i,j}$ and $N_{i,j}$ values. The offset $O_{i,j}$ creates checking patterns in the array and is determined by two parameters called *RISE*, and *RUN*: *RISE/RUN* gives the slope of the waves of checking in the checking pattern. With PACED applied to the 2-D array, this chapter will investigate the confidence to place on array outputs at error detection time, the error coverage, and the performance of the array.

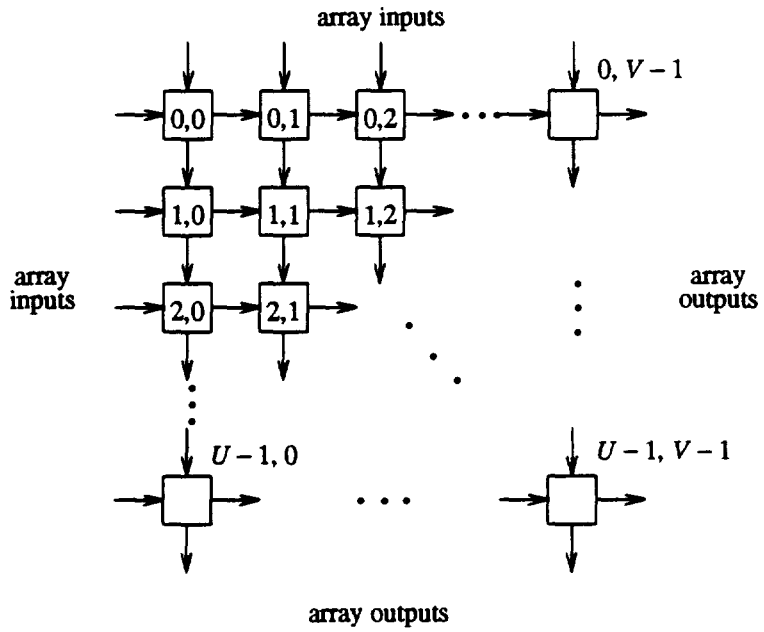


Figure 5.1. A $U \times V$ 2-D mesh processor array.

First, the use of PACED in a 2-D array will be analyzed to determine which outputs to suspect upon error detection. The confidence analysis is based on three assumptions similar to those used in Chapter 4. 1) All communication channels in the array are fault-free. 2) If an erroneous output is produced by a PE, it will be propagated downstream both rightward and downward. 3) PEs are code-disjoint: use of erroneous inputs or state values causes erroneous PE outputs to propagate both rightward and downward.

5.1. Error Detection Latency

In order to alert the external world of an error detection in the array, an error signal must reach a PE that produces external outputs. The error detection latency does not include this signal delay. Upon error detection, a message is sent by the detecting $PE_{i,j}$ downstream with its

output data indicating the PE and computation cycle of the detection. The time for a user to become aware of an error detection at $PE_{i,j}$ is proportional to $\min(U-i-1, V-j-1)$.

In 2-D arrays, an algorithm is used to determine L_{max} and L_r, the latency of an error created in an unchecked computation cycle r of $PE_{i,j}$, $N_{i,j} \le r \le M_{i,j-1}$. When PACED is applied to a 2-D array, the checking pattern is set by $O_{i,j} = (M_{i,j} + i + j - (U-1-i)RUN - (V-1-j)RISE) \bmod M_{i,j}$. This particular $O_{i,j}$ was derived empirically, based on the shape of the optimal checking pattern for linear arrays: since errors propagate downstream in the array, waves of checking that proceed upstream in time were desired to reduce the detection latency. The algorithm propagates an error from $PE_{i,j}$ in cycle c downstream through the array until it is detected in cycle $c+z$, giving $L_r = z$. The algorithm uses the fact that when the checking activity at $PE_{i,j}$ in cycle c is $CS_{M,N}[r]$, the checking activity at $PE_{i+y,j+x}$ in cycle $c+z$ is $CS_{M,N}[(r + xRISE + yRUN + z) \bmod M_{i,j}]$.

As in the linear processor array, errors may be created that propagate undetected out of the array. However, in the linear array, the checking pattern was designed such that only a few of the endmost PE_i could create undetected errors. Such is not the case for the 2-D array, in which *RISE* and *RUN* can be chosen to create a variety of checking patterns. Therefore, L_{max} for the 2-D array is defined as the largest finite error detection latency.

EXAMPLE 5.1: Figure 5.2 shows several snapshots of a 10×10 array in the midst of some computation, with $M_{i,j} = 10$, $N_{i,j} = 3$, *RISE/RUN* = 2/1, and $O_{i,j} = (2i + 3j - 17) \bmod 10$. The detection latency for an error created at $PE_{2,5}$ in cycle c (marked in the figure by **e**), when the checking activity at $PE_{2,5}$ is $CS_{10,3}[5]$, is called L_5 and equals 2, since both $PE_{3,6}$ and $PE_{2,7}$

detect the error in cycle $c+2$. The figure shows how the error propagates through the array (* in the figure) until detection (in the figure, **X**). For this array, $L_{max} = L_N = L_3 = 3$. □

5.2. Suspected Outputs

This section considers which outputs to suspect as possibly erroneous when an error is detected at $PE_{i,j}$ in a 2-D processor array. As in the single-processor and linear-array discussions, outputs produced both prior to the detection as well subsequent thereto are considered. For the first case, a simple algorithm works backwards in time from the point of detection, to determine through which upstream PEs the error could have propagated; the outputs from those PEs should be suspected. The algorithm runs in $O(UV \cdot N_{i,j})$ time, assuming $N_{i,j}$ is constant for all $PE_{i,j}$.

EXAMPLE 5.2: Figure 5.3 shows five snapshots of a 10×10 processor array using standard PACED with $M_{i,j} = 13$, $N_{i,j} = 5$, *RISE/RUN* = 3/1, and $O_{i,j} = (2i + 4j - 23) \bmod 13$. Each grid

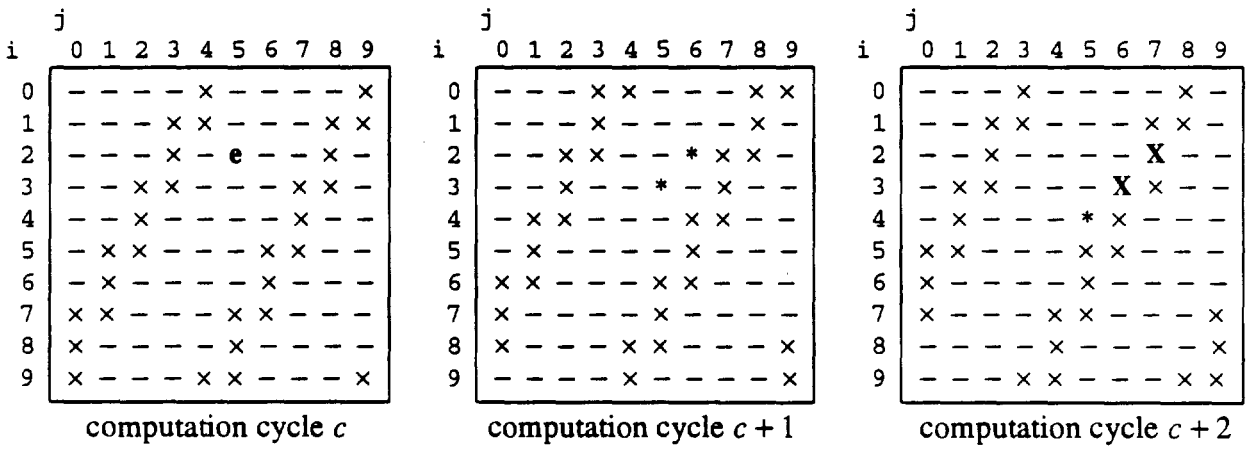


Figure 5.2. Error detection latency.

represents the checking activity in the array in one computation cycle. The outputs to suspect are marked either as @ (where the error was detected) or * (from where the error might have propagated).

If an error is detected at $PE_{9,9}$ in cycle c, its output should be suspected as possibly erroneous. Also, the outputs from the following PEs should be suspected as possibly erroneous: $PE_{9,8}$ and $PE_{8,9}$ in cycle $c-1$; $PE_{9,7}$, $PE_{8,8}$, and $PE_{7,9}$ in cycle $c-2$; $PE_{7,8}$ and $PE_{6,9}$ in cycle $c-3$; and $PE_{4,9}$ in cycle $c-4$. All other unsuspected, previously produced outputs can be trusted with a confidence of 1, unless a later error detection makes it necessary to suspect them.□

The PACED′ modification in the 2-D array performs 100% checking at $PE_{U-1,V-1}$, either by duplicating $PE_{U-1,V-1}$ or by monitoring its outputs with a hardware code checker. With PACED′ in use, errors cannot escape undetected from the array. As in the linear array case, this modification obviates the need to suspect any future outputs from the array: all errors are eventually

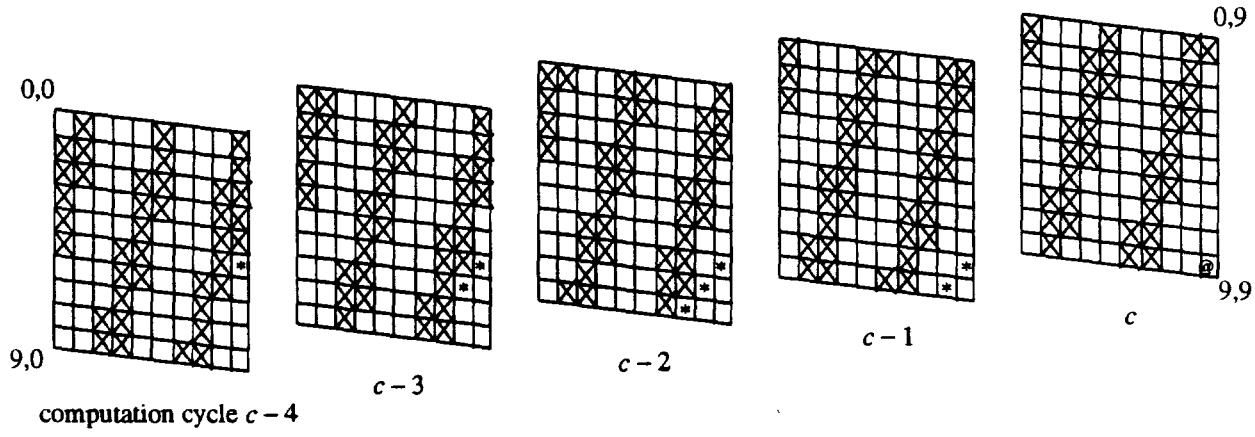


Figure 5.3. Suspected previously produced outputs, 10×10 array.

detected, so only previously produced outputs have to be suspected at error detection time. However, in the standard PACED implementation, some detected errors may propagate downstream, corrupting other outputs before escaping the array.

An algorithm similar to that used to find the suspected previous outputs first works backwards from each unchecked cycle of $PE_{U-1,V-1}$ to determine from which upstream PEs, in earlier checked cycles, undetected errors may have propagated. Also, potential sites of suspected outputs are marked in this step. From these detection sites, errors are then propagated forward retracing the paths found in the first step; errors on paths that do not lead to subsequent detections are marked suspect. This algorithm runs in $O(UV \cdot N_{i,j})$ time, assuming $N_{i,j}$ is constant for all $PE_{i,j}$.

EXAMPLE 5.3: Figure 5.4 shows three snapshots of a 10×10 processor array using standard PACED with $M_{i,j} = 10$, $N_{i,j} = 3$, $RISE/RUN = 2/1$, and $O_{i,j} = (2i + 3j - 17) \bmod 10$. The figure is notated as in Figure 5.3.

If an error is detected at $PE_{8,8}$ in cycle c (marked @ in the figure), its output should be suspected as possibly erroneous. Furthermore, the outputs from the following $PE_{i,j}$ should also be suspected as possibly erroneous: $PE_{8,9}$ and $PE_{9,8}$ in cycle $c+1$, and $PE_{9,9}$ in cycle $c+2$ (all marked by *). All other unsuspected, future outputs can be trusted with a confidence of 1 (until, of course, the next error detection). □

The detection of an error by one of the $N_{i,j}$ checks at $PE_{i,j}$ leads to static patterns of previous and future outputs to suspect as possibly erroneous. For example, Figure 5.3 is the pattern of previous outputs to suspect if $M_{i,j} = 13$, $N_{i,j} = 5$, $RISE/RUN = 3/1$ (giving $O_{i,j}$ =

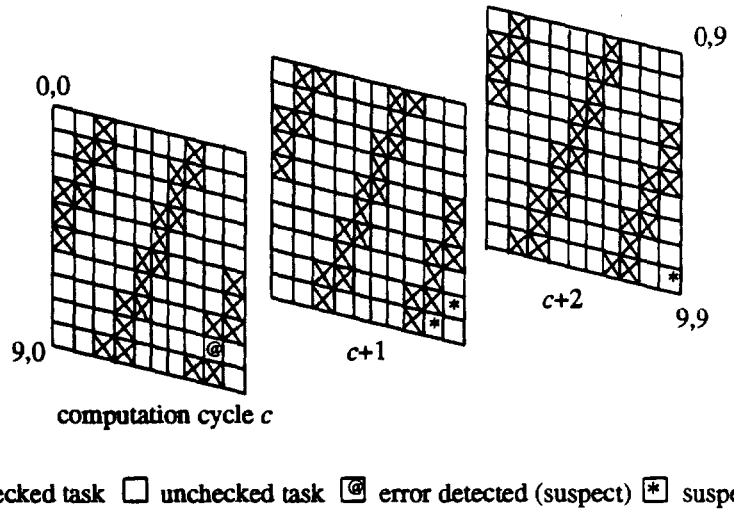


Figure 5.4. Suspected future outputs, 10×10 array.

$(2i + 3j - 23) \bmod 13)$, and $CS_{13,5}[0]$ detects an error; Figure 5.4 is the pattern of future outputs to suspect if $M_{i,j} = 10$, $N_{i,j} = 3$, *RISE/RUN* = 2/1 (giving $O_{i,j} = (2i + 3j - 17) \bmod 10$), and $PE_{U-2,V-2}$ detects an error by $CS_{10,3}$ [1].

For given values of $M_{i,j}$, $N_{i,j}$, *RISE*, and *RUN*, there are a fixed number of possible patterns of suspected outputs: one for each $CS_{13,5}[r]$, $0 \le r \le 4$, for the previously produced outputs, and a variable number of patterns generated from each $CS_{13,5}[r]$, $5 \le r \le 12$, for the future outputs. Because the PACED parameter values are known, these patterns can be computed once for the array using the algorithms described and stored, indexed by r, i, and j. Upon error detection, given which check detected the error (the index of $CS_{M,N}$) at which $PE_{i,j}$, the outputs to suspect can be determined with no extra computations by recalling the appropriate template.

In the linear array case, it was possible to determine analytically which checking pattern, for a given $M_{i,j}$ and $N_{i,j}$, would lead to the minimal maximum error detection latency [19].

Such an analytical treatment is less tractable for the 2-D array, so a pattern generator program and analyzer program were written in C to examine the search space. The pattern generator program takes as input the architecture of the array (linear, 2-D, or triangular), the dimensions of the array, values of the PACED parameters M, N, and O (for the linear array case) or *RISE* and *RUN* (for the other architectures), and the number of computation cycles to generate. It produces a series of snapshots of the array for the requisite number of computation cycles, showing the checking pattern generated by the PACED parameter values. The analyzer program takes the output of the pattern generator as input and determines L_{max}, as well as the number of outputs to suspect, both forward and backward, for the particular PACED parameter values.

A 20×20 array was tested, setting $M_{i,j} = 15$, $N_{i,j} = 1, 2, \cdots 15$, $O_{i,j} = (15 + i + j - (19 - i)RUN - (19 - j)RISE)$ mod 15, and $q = 1$. By varying *RISE* and *RUN*, patterns with waves of different slopes were generated. These patterns were then analyzed to determine their maximum error detection latency, as well as the pattern and number of both previous outputs (Table 5.1) and future outputs (Table 5.2) to suspect when an error is detected.

For each row of the tables, the first two columns give the checking ratio and percentage, and the third column gives the particular *RISE* and *RUN* values used to obtain the other values in that row. The fourth column, L_{max}, gives the minimal maximum error detection latency that achieves the minimum number of previous outputs (Table 5.1) or future outputs (Table 5.2) that should be suspected as possibly erroneous (sixth column). The fifth column gives the number of computation cycles that these suspected outputs span.

TABLE 5.1.
NUMBER OF SUSPECTED PREVIOUS OUTPUTS, 2-D ARRAY.

N/M	% checking	*RISE/RUN*	L_{max}	# cycles	min # bwd susp. o/p	min # fwd susp. o/p
1/15	6.7	0/0 1/1 3/3	14	15	120	679
2/15	13	2/2	4	5	30	68
3/15	20	2/2	4	5	45	102
4/15	27	4/4	2	3	24	36
5/15	33	4/4	2	3	30	45
6/15	40	-7/8 5/5 8/8	2	3	27	36
7/15	47	-8/7 6/6 7/7	2	3	24	27
8/15	53	<8 options>	1	2	22	21
9/15	60	<32 options>	1	2	21	18
10/15	67	<72 options>	1	2	20	15
11/15	73	<128 options>	1	2	19	12
12/15	80	<200 options>	1	2	18	9
13/15	87	<325 options>	1	2	17	6
14/15	93	<544 options>	1	2	16	3
15/15	100	<all options>	0	0	0	0

It is interesting to note that particular patterns that work well in the backward interval do not generally work well in the forward interval. The last columns in each table are provided for the purposes of comparison. For example, with $N = 1$, Table 5.1 suggests that using *RISE/RUN* = 0/0, 1/1, or 3/3 gives a minimum of 120 previously produced outputs to suspect, but when these slopes are used, 679 outputs must be suspected subsequent to certain error detections. The reverse is also true: with $N = 5$, Table 5.2 suggests using *RISE/RUN* = -1/4 to limit the amount of future suspected outputs to 25, yet 205 previously produced outputs should also be suspected upon error detection. Clearly, the search space is large and complex; use of the pattern generator and analysis programs can aid a designer of such a system to choose the best *RISE/RUN* for a desired checking ratio, to minimize the amount of output to suspect.

TABLE 5.2.
NUMBER OF SUSPECTED FUTURE OUTPUTS, 2-D ARRAY.

N/M	% checking	*RISE/RUN*	L_{max}	# cycles	min # fwd susp. o/p	min # bwd susp. o/p
1/15	7	-1/4	2	3	5	41
2/15	13	-1/4	2	3	10	82
3/15	20	-1/4	2	3	15	123
4/15	27	-1/4	2	3	20	164
5/15	33	-1/4	2	3	25	205
6/15	40	-1/5 -1/8	2	3	21	186
7/15	47	-1/6 -1/7	2	3	17	167
8/15	53	-1/6 -1/7	1	2	14	148
9/15	60	-1/5 -1/6 -1/7 -1/8	1	2	12	129
10/15	67	-1/4 -1/5 -1/6, -1/7 -1/8 -1/9	1	2	10	110
11/15	73	<32 options>	1	2	8	11
12/15	80	<40 options>	1	2	6	12
13/15	87	<64 options>	1	2	4	13
14/15	93	<83 options>	1	2	2	14
15/15	100	<all>	0	0	0	0

The tables show that only about one second's worth of output (on the order of 10 cycles' worth with cycle times less than 100 ms) need be suspected, either forward or backward in the 2-D array, upon error detection. As in the linear array case, this is a great improvement over the amount of suspected output in the single processor case and shows again how PACED utilizes the cooperation of PEs checking other PE outputs to afford high confidence in outputs with only periodic checking.

5.3. Error Coverage

As in the linear array case, the error coverage in the 2-D array can be estimated if it is assumed that errors occur uniformly distributed in space among the PEs in the array. Again,

only one M-cycle period has to be examined, as all other M-cycle periods are identical and have the same coverage.

In one M-cycle period in a $U \times V$ 2-D mesh array, there are MUV potential sites at which error may occur: one for each PE of the array, in each cycle. Since it is assumed that errors propagate through the array and are not masked, only a fraction of the potential sites can lead to the propagation of undetected errors out of the array, if an error occurs. (This is when normal PACED is applied; use of PACED′ would result in 100% error coverage, as no errors can escape from the array undetected.) The estimated error coverage is just the total number of sites from which undetected errors may propagate out of the array, divided by the total number of potential sites.

Figure 5.5 shows the estimated error coverage for a 4×4 PE mesh array as a function of N/M, when $M = 10$ and $q = 1$. When N/M is small, the error coverage is low; but the coverage increases quickly as N/M increases: greater than 95% coverage can be achieved with N/M just 0.5 or greater. As in the linear array case, low values of the checking ratio can yield high error coverage — and low checking ratios can lead to reduced performance cost of applying CED.

5.4. Performance

As in the linear array case, the performance of 2-D processor arrays was studied in two ways: from reduced simulations using the C simulation model, and from full simulations on the Intel iPSC/2 hypercube running a matrix-multiply algorithm. The results from these two performance analyses are presented in the following two subsections.

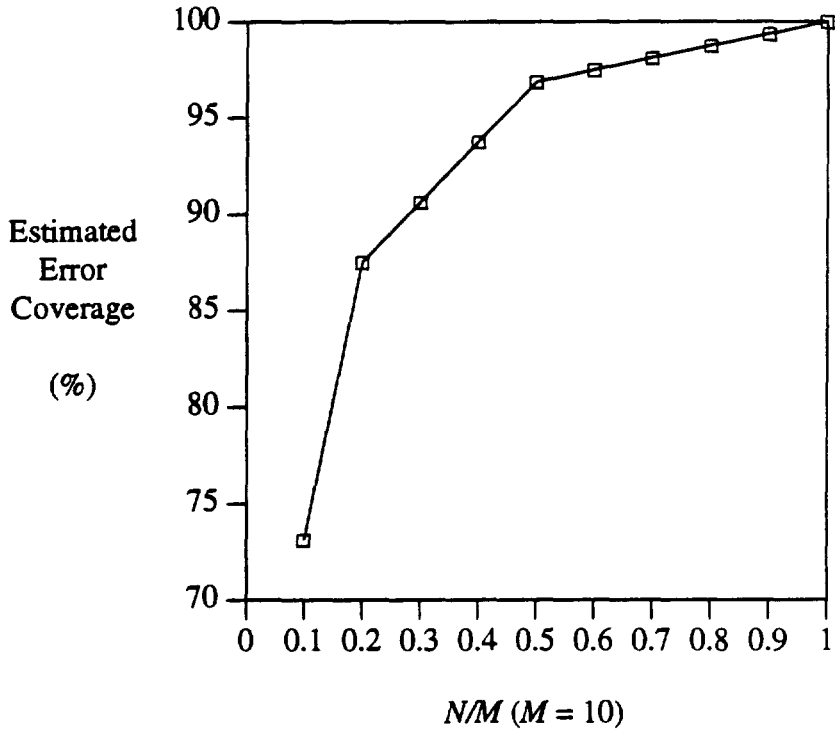


Figure 5.5. Estimated error coverage for a 4×4 mesh array.

5.4.1. Simulation model

The simulation-based analysis model introduced in Section 4.5.1 was also used to estimate the performance of PACED when applied to square and triangular processor array architectures. One array investigated was a triangular array running an adaptive beamforming algorithm.

EXAMPLE 5.4: Digital adaptive beamforming is a signal processing algorithm that optimizes the reception of a desired signal received at an antenna array. A triangular processor array has been designed for high-performance, adaptive, digital beamforming [50], and is shown in Figure 5.6. The triangular array consists of four types of PEs: boundary, internal, Y-column and a residual former. During each computation cycle, PEs of each of the first three types

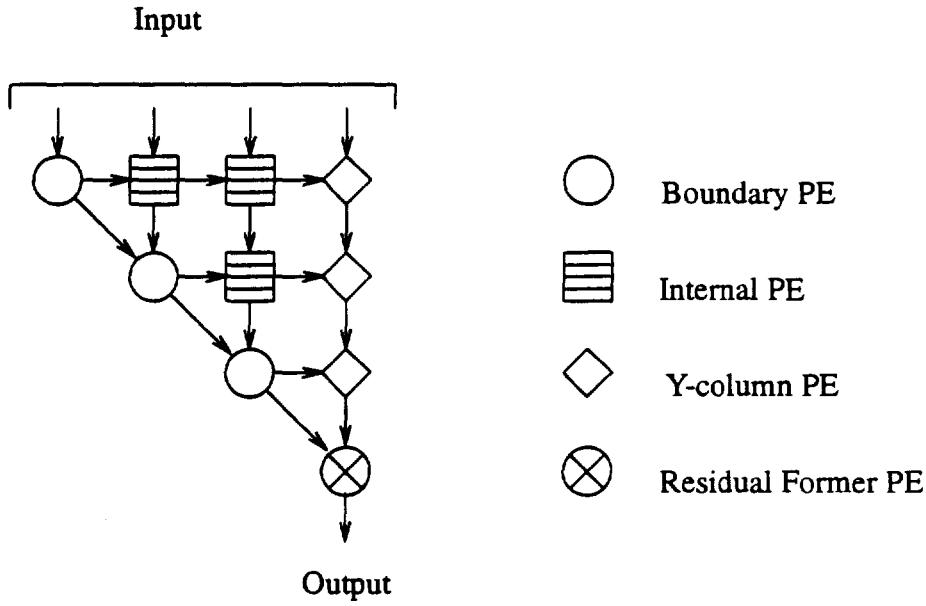


Figure 5.6. Triangular array for adaptive digital beamforming.

compute outputs and update an internal state variable; the residual former does not maintain any state, and only computes an output.

The modeled CED scheme replicated with duplicate data the computations at each PE. A full simulation using the OODRA (Object-Oriented Design of Reliable/reconfigurable Architectures) workbench [51] was used to determine the mean task and check times for each type of PE in the array, for one computation cycle. The mean task times are given in the second column of Table 5.3; the units in the table are defined such that three units equal the average time required for the residual former to complete one task.

The table shows that the boundary PE task required at least an order of magnitude more time than any of the other PE tasks, because of its costly state update computation (involving a square root). Therefore, five different variations of the CED scheme were considered. In each

TABLE 5.3.
TASK AND CHECK TIMES,
ADAPTIVE BEAMFORMING PEs.

PE type	task time	check time using CED scheme: I	II	III	IV	V
Boundary	104	106	17	17	88	88
Internal	15	16	16	8	8	0
Y-column	15	16	16	8	8	0
Residual	3	4	4	4	0	0

variation, only a subset of the computations performed at each PE in a computation cycle were checked whenever the CED technique is performed.

I/ All output and state computations at each PE were checked. This provided the greatest probability of detecting an error, if one were to occur.

II/ All output and state computations except the boundary PE state update computation were checked. This scheme attempted to check as many of the computations as possible, while saving the most time by not replicating the longest operation.

III/ Only output computations at each PE were checked.

IV/ Only state update computations at each PE were checked.

V/ Only the boundary PE state update computation was checked. This scheme covered the most time at the boundary PEs while trying to minimize the number of computations to replicate.

The last five columns of Table 5.3 show the mean check times for each PE type, for each of the five CED schemes; again, the units are relative to the residual former task time.

The simulation-based analysis model determined the performance of a 4×4 triangular array running the adaptive, digital, beamforming algorithm using PACED with $M_{i,j} = M$, $N_{i,j} = N$, and $q = 1$. The simulation was run for 500 computation cycles. For each of the five CED schemes, five different checking patterns were applied, in which different subsets of the PEs in the array were checking at any particular computation cycle: the entire array, a row, a column, a forward wavefront with slope 1, and a backward wavefront with slope 1. (These simulations were performed before the 2-D array was analyzed. Hence, the formula given for $O_{i,j}$ in Section 5.1 was not used; other formulas for $O_{i,j}$ were derived to fit the desired PE subsets.) If T_0 and T_C represent the time units estimated by the model to run an algorithm without and with using CED, respectively, then the degraded performance is T_0/T_C and the checking overhead is $(T_C - T_0)/T_0$. Figure 5.7(a) shows the performances, and Figure 5.7(b), the checking overheads, of each of the five CED schemes as a function of *N/M*.

It was found that the performance degradations resulting from the five checking patterns were practically identical, for any of the CED schemes employed: the performance impact of PACED depended only upon *M* and *N*. Therefore, each curve in Figure 5.7(a) represents the (identical) performances using the five checking patterns considered, and each curve in Figure 5.7(b) represents the (identical) overheads of those patterns. □

For $N = 0$, the modeled PACED system suffers no performance degradation, regardless of the CED scheme used. Since checks involve a replicated computation plus a comparison, for $N/M = 1$, the checking overhead for CED scheme I exceeds 100% and the performance is slightly less than 50% of the basic performance. The pair of CED schemes II and III have the

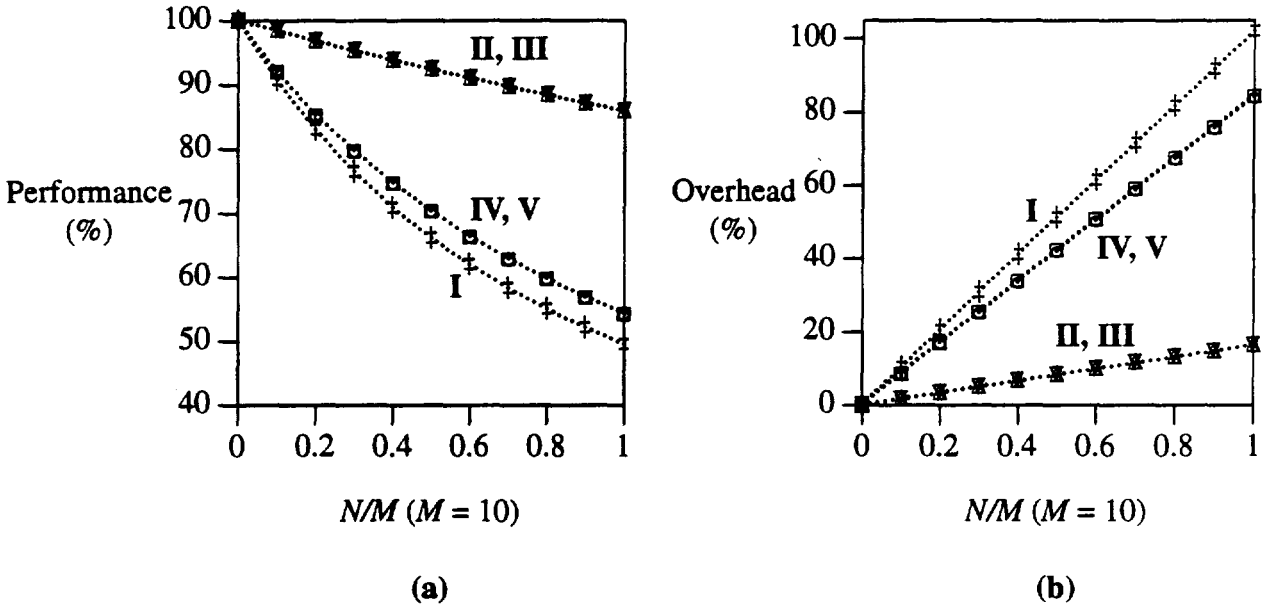


Figure 5.7. Adaptive beamforming array.
(a) Performance degradation. **(b)** Checking overhead.

same performance and overhead, as do the pair IV and V. This means that even though CED scheme II replicates the state update computations (which scheme III does not), these extra computations can be done essentially with no added cost, because the large boundary task time forces the other PEs in the array to wait and it is in these idle times that the checking of schemes II and III is performed. Since no extra wait states are propagated to the residual former, and since the residual former performs the same amount of checking in the two schemes, no performance difference is observed. For the same reason, if the boundary PE state update computations are checked (scheme V), then all PE state update computations can be checked with no extra performance cost (scheme IV). Hence, of these five CED schemes, I, II and IV represent the most intelligent options.

In this example the performance degradation and overhead were constant for any PACED checking pattern chosen, given a particular CED scheme. This has been shown to be true for the linear array [19] and should be true in general, since the $O_{i,j}$ parameter only affects the initialization of PACED at each PE in the array: the performance depends only upon the checking ratio *N/M*.

5.4.2. Hypercube simulations

A simulation of a 2-D mesh processor array was performed on an Intel iPSC/2 hypercube, with the nodes serving as the PEs and using the shortest internode connections to minimize the communication overhead. A matrix-multiply algorithm was implemented in C in which rows and columns of each input matrix were distributed to the PEs through the top and left edges of the array and sent thereon through the array. Each PE computed a submatrix of the final matrix result; the final result was collected by the host at the end of computation.

The simulations used all 16 nodes of the hypercube to multiply together two 136×136 matrices of random floating-point numbers. With $M_{i,j} = M$ and $N_{i,j} = N$, two CED techniques were employed: RESO and neighbor-assist. (AN-coding only applies to integer applications; to date, there are no known arithmetic codes for floating-point numbers.) The RESO employed was the same as that used in the simulations of the linear array performing image edge detection (Section 4.5.2). In the neighbor-assist technique, each $PE_{i,j}$ requests a recomputation of N of its computations from a nearest neighbor PE, which then sends back the results. Both PEs perform a comparison of the two sets of results and any discrepancy greater than an error tolerance (1.5×10^{-15} times the value of a result) triggers an error detection. The neighbor assist

technique is patterned after CORP (concurrent retry procedure) of Manolakos et al. [52], which is used by NEAR (neighbor-assisted recovery) [53], the Γ-processes [17], and the overlapping H-processes [18] for 2-D processor arrays. In this implementation of NEAR, to reduce the number of CED-related messages, each $PE_{i,j}$ saves *N*-out-of-*M* sets of its operands and requests CED assistance only once every *M* computation cycles.

Figure 5.8 shows the performance cost of using CED in varying checking ratios *N/M*, by comparing the completion times of the different versions of the algorithm. The completion times do not include the initializations of the host or node programs. For each run, the individual completion times of each of the 16 nodes were averaged together. The averages from five

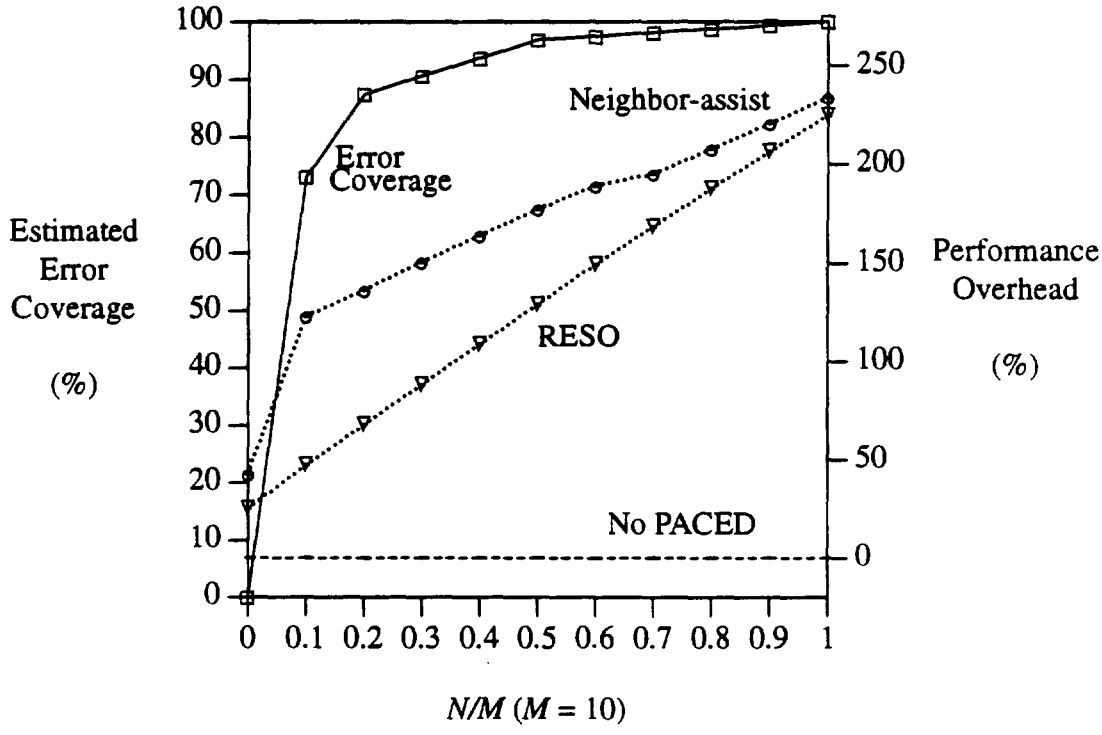


Figure 5.8. Mesh array performance, matrix multiply.

runs were then averaged to obtain each data point on the graph. Just five runs were deemed sufficient for two reasons: 1) the greatest standard deviation for the individual node completion times was less than 3.4% of the average node completion time, and 2) the greatest standard deviation for the run averages was less than 0.15% of the average run completion times. The graph does show the 95% confidence intervals for each data point but they are too small to be seen.

Unlike the linear array performance results, the use of RESO or neighbor-assist clearly degrades the performance of the 2-D mesh array, in an almost linear fashion. This is probably because each computation cycle in the matrix-multiply algorithm has but four data sends/receives, compared with ten in the edge detection algorithm, and only 136 iterations of the computation cycle were required for the 136×136 matrix multiply, whereas 1024 iterations were performed in each run of the edge detection algorithm: overall, the matrix-multiply algorithm required far less communication than the edge detection algorithm so that the effects of CED were more pronounced on the completion time of the matrix-multiply algorithm.

Both the neighbor-assist and RESO curves show a significant amount of overhead at N/M = 0 since both techniques replace each operand-destroying assignment statement with two statements and a temporary variable, resulting in code expansion. However, the short overall execution time of the matrix-multiply algorithm amplifies the apparent overhead introduced by CED. The absolute time overhead for RESO at $N/M = 0$ is about 1 second; this is approximately the same absolute time overhead exhibited by RESO in the edge detection array. Since the matrix-multiply execution time is smaller, the percent overhead is much larger.

The figure shows that at 100% checking, both the RESO and neighbor-assist techniques display almost 250% overhead. In the basic version of the algorithm, the main computation consists of one floating-point multiply and one floating-point add. Use of RESO adds six extra floating-point multiplies as well as one extra floating-point add for each checked computation cycle. Though this more than triples the original amount of computation, more overhead is not apparent since the extra work incurred by RESO is a smaller proportion of the total amount of computation performed in each computation cycle.

The basic version of the algorithm performs two data receives and two data sends each computation cycle. In the neighbor-assist case, every M cycles, two extra CED messages are both sent and received. This is the cause of the jump in execution time exhibited between $N/M = 0$ and $N/M = 0.1$. Thereafter, the extra-message overhead remains constant, and the increase of overhead with increased N/M comes from the extra computations each node performs as CED for a neighbor. The slope of the curve from $N/M = 0.1$ to $N/M = 1$ is gentler than that of the RESO case, as less extra computation is performed: just one extra floating-point multiply and add, for each checked computation cycle, in addition to copying the operands and partial products for its own neighbor assistant.

From these experiments, it can be concluded that, as expected, use of PACED can reduce the performance costs incurred through the use of CED in a 2-D processor array. A designer of such an array can trade off between performance and the amount of outputs to suspect (and thereby, the error coverage) by choosing appropriate levels of the checking ratio N/M, provided a coding technique is used that facilitates error propagation in the array.

CHAPTER 6.

SUMMARY

In this thesis, it was shown that the use of periodic application of concurrent error detection (PACED) in VLSI processor array architectures can be an attractive alternative to the continuous use of CED in linear and two-dimensional processor array architectures.

It was shown that for PACED applied in a single processor, high confidence can be achieved when only a small amount of output is suspected as possibly erroneous. This is possible assuming that errors arrive in clusters, with a fairly high arrival rate occurring for intracluster errors and a very small arrival rate for clusters themselves.

For PACED applied in a unidirectional linear or two-dimensional mesh-connected processor array, even fewer of the array's previous outputs have to be suspected upon error detection, if a suitable coding scheme can be found to ensure the propagation of errors. Then, PEs in such arrays can cooperate to check the unchecked outputs of other PEs. Furthermore, future outputs have to be suspected only for PEs near the ends of linear arrays, since only these PEs can create errors that could possibly propagate undetected from the array. Any PE in a two-dimensional array can create an undetected error, and in these cases, somewhat more output has to be suspected, depending on the position of the PE in the array. However, the sum total of outputs and the time interval which they encompass are smaller than those required for PACED in a single processor.

For each possible error detection site in the linear or two-dimensional array, a static pattern of outputs to suspect can be predetermined and stored. Upon error detection, knowledge of the particular check and PE that detected the error can be used to retrieve an error pattern that determines which outputs to suspect. Therefore, very little run-time overhead is required at error detection time to determine which outputs should be suspected.

For all three of the architectures considered, the error coverage was found to be quite high even for low values of the checking ratio *N/M*. In the single processor case, this was due to the ability of the undetected-errors intervals to, in effect, "detect" errors that would otherwise have gone undetected, by casting suspicion on outputs that may have been corrupted. In the array cases, high coverage is achieved by the cooperation of the constituent PEs in the arrays to check the unchecked outputs of other PEs.

In empirical studies of the performance cost of PACED in linear and two-dimensional arrays, it was found that performance was degraded approximately linearly with the amount of checking performed. Hence, PACED can reduce the performance cost of performing CED in such architectures by performing CED periodically instead of continuously. Coupled with the potentially high confidence that can be placed on most outputs at error detection time as well as the high error coverages possible even with infrequent checking, PACED can be an attractive alternative to continuous CED for some applications.

This thesis has also described a simulation model that can estimate the performance cost of PACED in unidirectional linear, two-dimensional mesh and triangular processor arrays. This model, plus the confidence theorems and algorithms as well as the error coverage estimates presented in this thesis, form a powerful package that can aid a designer in choosing the PACED

parameter values to trade off the performance cost of using CED for a minimal error detection latency, minimal number of outputs to suspect, and high error coverage.

REFERENCES

[1] H. Yamamato, T. Watanabe, and Y. Urano, "Alternating logic and its application to fault detection," *Proc. 1970 IEEE Int. Computer Group Conf.*, pp. 220-228, June 1970.

[2] D. A. Reynolds and G. Metze, "Fault detection capabilities of alternating logic," *IEEE Trans. Computers*, vol. C-27, no. 12, pp. 1093-1098, Dec. 1978.

[3] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," *IEEE Trans. Computers*, vol. C-31, no. 7, pp. 589-595, July 1982.

[4] R. K. Gulati and S. M. Reddy, "Concurrent error detection in VLSI array structures," *Proc. IEEE Int. Conf. Computer Design*, pp. 488-491, Oct. 1986.

[5] Y. H. Choi and M. Malek, "A fault-tolerant FFT processor," *IEEE Trans. Computers*, vol. 37, no. 5, pp. 617-621, May 1988.

[6] F. T. Luk and E. K. Tomg, "Fault tolerance techniques for systolic arrays," *Proc. SPIE, Vol. 827, Real-Time Signal Processing X*, pp. 30-36, 1987.

[7] J. H. Kim and S. M. Reddy, "A fault-tolerant systolic array design using TMR method," *Proc. Int. Conf. Computer Design*, pp. 769-773, 1985.

[8] A. Majumdar, C. S. Raghavendra, and M. A. Breuer, "Fault tolerance in linear systolic arrays using time redundancy," *IEEE Trans. Computers*, vol. 39, no. 2, pp. 269-276, Feb. 1990.

[9] J. Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *Proc. 15th Int. Symp. Fault-Tolerant Computing*, pp. 338-343, 1985.

[10] K. H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Computers*, vol. C-33, no. 6, pp. 518-528, June 1984.

[11] J. C. Fabre, Y. Deswarte, J. C. Laprie, and D. Powell, "Saturation: reduced idleness for improved fault-tolerance," *Proc. 18th Int. Symp. Fault-Tolerant Computing*, pp. 200-205, 1988.

[12] A. T. Dahbura, K. K. Sabnani, and W. J. Hery, "Spare capacity as a means of fault detection and diagnosis in multiprocessor systems," *IEEE Trans. Computers*, vol. 38, no. 6, pp. 881-891, June 1989.

[13] P. Banerjee, J. T. Rahmeh, C. B. Stunkel, V. S. S. Nair, K. Roy, and J. A. Abraham, "An evaluation of system-level fault tolerance on the Intel Hypercube multiprocessor," *Proc. 18th Int. Symp. Fault-Tolerant Computing*, pp. 362-367, June 1988.

[14] C. L. Wey, "Concurrent error detection in array dividers by alternating input data," *IEE Proceedings-E*, vol. 139, no. 2, pp. 123-130, Mar. 1992.

[15] W. T. Cheng and J. H. Patel, "Concurrent error detection in iterative logic arrays," *Proc. 14th Int. Symp. Fault-Tolerant Computing*, pp. 10-15, June 1984.

[16] S. W. Chan and C. L. Wey, "The design of concurrent error diagnosable systolic arrays for band matrix multiplications," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 1, pp. 21-37, Jan. 1988.

[17] E. S. Manolakos and M. Bletsas, "The Γ-process: A time redundancy mechanism for concurrent error diagnosis in wavefront arrays," submitted to *23rd Int. Symp. Fault-Tolerant Computing*, 1993.

[18] E. Manolakos, D. Dakhil, and M. Vai, "Concurrent error diagnosis in mesh array architectures based on overlapping H-processes," *Proc. IEEE Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 139-152, Nov. 1991.

[19] Y. M. Wang, P. Y. Chung, and W. K. Fuchs, "Design and scheduling for periodic concurrent error detection and recovery in processor arrays," Technical Report CRHC-92-08, Center for Reliable and High-Performance Computing, Univ. of Illinois, Urbana, IL, May 1992.

[20] G. S. Sohi, M. Franklin, and K. K. Saluja, "A study of time-redundant fault tolerance techniques for high-performance pipelined computers," *Proc. 19th Int. Symp. Fault-Tolerant Computing*, pp. 436-443, 1989.

[21] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu, "IMPACT: An architectural framework for multiple-instruction-issue processors," *Proc. 18th Int. Symp. Computer Architecture*, pp. 266-273, 1991.

[22] J. Holm and P. Banerjee, "Low cost concurrent error detection in a VLIW architecture using replicated instructions," *Proc. 1992 Int. Conf. Parallel Processing*, vol. I, pp. 192-195, Aug. 1992.

[23] M. A. Schuette and J. P. Shen, "Exploiting instruction-level resource parallelism for transparent, integrated control-flow monitoring," *ONR 2d Annual Review & Workshop*, Nov. 1991.

[24] M. A. Breuer, "Testing for intermittent faults in digital circuits," *IEEE Trans. Computers*, vol. C-22, no. 3, pp. 241-246, Mar. 1973.

[25] S. Kamal and C. V. Page, "Intermittent faults: A model and a detection procedure," *IEEE Trans. Computers*, vol. C-23, no. 7, pp. 713-719, July 1974.

[26] S. Y. H. Su, I. Koren, and Y. K. Malaiya, "A continuous-parameter Markov model and detection procedures for intermittent faults," *IEEE Trans. Computers*, vol. C-27, no. 6, pp. 567-570, June 1978.

[27] I. Koren and S. Y. H. Su, "Reliability analysis of N-modular redundancy systems with intermittent and permanent faults," *IEEE Trans. Computers*, vol. C-28, no. 7, pp. 514-520, July 1979.

[28] R. K. Iyer and P. Velardi, "Hardware-related software errors: Measurement and analysis," *IEEE Trans. Software Engineering*, vol. SE-11, no. 2, pp. 223-231, Feb. 1985.

[29] D. Tang and R. K. Iyer, "Dependability measurement and modeling of a multicomputer system," *IEEE Trans. Computers*, vol. 42, no. 1, pp. 62-75, Jan. 1993.

[30] *SAS user's guide: Statistics.* Cary, NC: SAS Institute, Inc., 1985.

[31] I. Lee, D. Tang, R. K. Iyer, and M. C. Hsueh, "Measurement-based evaluation of operating system fault tolerance," to appear, *IEEE Trans. Reliability*, vol. 42, no. 6, June 1993.

[32] M. Shoga, P. Adams, D. L. Chenette, R. Koga, and E. C. Smith, "Verification of single event upset rate estimation methods with on-orbit observations," *IEEE Trans. Nuclear Science*, vol. NS-34, no. 6, pp. 1256-1261, Dec. 1987.

[33] D. Chlouber, P. O'Neill, and J. Pollock, "General upper bound on single-event upset rate," *IEEE Trans. Nuclear Science*, vol. 37, no. 2, pp. 1065-1071, Apr. 1990.

[34] A. Patterson-Hine, personal communication, Oct. 1990.

[35] L. L. Sivo, J. C. Peden, M. Brettschneider, W. Price, and P. Pentecost, "Cosmic ray-induced soft errors in static MOS memory cells," *IEEE Trans. Nuclear Science*, vol. NS-26, no. 6, pp. 5042-5047, Dec. 1979.

[36] D. Binder, E. C. Smith, and A. B. Holman, "Satellite anomalies from galactic cosmic rays," *IEEE Trans. Nuclear Science*, vol. NS-22, no. 6, pp. 2675-2680, Dec. 1975.

[37] J. C. Pickel and J. T. Blandford, Jr., "Cosmic-ray-induced errors in MOS devices," *IEEE Trans. Nuclear Science*, vol. NS-27, no. 2, pp. 1006-1015, Apr. 1980.

[38] J. B. Blake and R. Mandel, "On-orbit observations of single event upset in Harris HM-6508 1K RAMs," Report SD-TR-86-89, Space Division, Air Force Systems Command, Los Angeles, Feb. 1987.

[39] E. Swartzlander, Jr., "Systolic FFT processors," pp. 133-140 in *Systolic Arrays*. Ed. W. Moore, A. McCabe, R. Urquhart. Bristol: Adam Hilger, 1987.

[40] V. K. P. Kumar and Y. C. Tsai, "Synthesizing optimal family of linear systolic arrays for matrix computations," pp. 51-60 in *Systolic Array Processors*. Ed. J. McCanny, J. McWhirter, E. Swartzlander, Jr.. New York: Prentice Hall, 1988.

[41] J. A. Vlontzos and S. Y. Kung, "A wavefront array processor using dataflow processing elements," *Proc. 1st Int. Conf. Supercomputing (Lecture Notes in Computer Science* **297***)*, pp. 744-767, Springer-Verlag, 1987.

[42] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits, and Applications.* New York: North-Holland, 1978.

[43] J. H. Patel and L. Y. Fung, "Concurrent error detection in multiply and divide arrays," *IEEE Trans. Computers*, vol. C-32, no. 4, pp. 417-422, Apr. 1983.

[44] V. Piuri, "Fault-tolerant array processors: An approach based upon A*N codes," *Proc. IEEE Intl. Symp. Circuits and Systems (ISCAS)*, pp. 199-203, June 1988.

[45] J. Crawford and P. Gelsinger, *Programming the 80386.* San Francisco: Sybex, 1987.

[46] S. Y. Kung, *VLSI Array Processors.* Englewood Cliffs: Prentice Hall, 1988.

[47] R. Bayford, "The bit-serial systolic back-projection engine (BSSBPE)," pp. 43-54 in *Application Specific Array Processors.* Ed. S. Y. Kung, E. Swartzlander, Jr., J. A. B. Fortes, K. W. Przytula. Los Alamitos: IEEE Computer Society Press, 1990.

[48] X. H. Wu and Z. Y. He, "Efficient systolic arrays for transform domain adaptive digital filters," pp. 23-32 in *Systolic Array Processors*. Ed. J. McCanny, J. McWhirter, E. Swartzlander, Jr.. New York: Prentice Hall, 1989.

[49] O. Menzilcioglu and H. T. Kung, "A highly configurable architecture for systolic arrays of powerful processors," pp. 156-165 in *Systolic Array Processors*. Ed. J. McCanny, J. McWhirter, E. Swartzlander, Jr.. New York: Prentice Hall, 1989.

[50] C. R. Ward, P. J. Hargrave, and J. G. McWhirter, "A novel algorithm and architecture for adaptive digital beamforming," *IEEE Trans. Antennas and Propagation*, vol. AP-34, no. 3, pp. 338-346, Mar. 1986.

[51] D. K. Hwang, T. L. Wernimont, and W. K. Fuchs, "Evaluation of a reconfigurable architecture for digital beamforming using the OODRA workbench," *Proc. 26th ACM/IEEE Design Automation Conf.*, pp. 614-617, June 1989.

[52] E. S. Manolakos and S. Y. Kung, "CORP — A new recovery procedure for VLSI processor arrays," *1988 IEEE Symp. Engineering in Computer-Based Medical Systems (EMB-88)*, June 1988.

[53] E. S. Manolakos and S. Y. Kung, "Neighbor assisted recovery in VLSI processor arrays," *Signal Processing IV: Theories and Applications*, pp. 1245-1249, Sept. 1988.

VITA

Paul Peichuan Chen was born in [redacted], on [redacted]. He earned his B.S. degree in Electrical Engineering from Stanford University, California, in 1984. For his work with the Tau Beta Pi Engineering Course Evaluation project at Stanford, Mr. Chen received the Dean's Award for Service, given by the Stanford School of Engineering, in 1984. Also in 1984, he was a Rhodes Scholar State Finalist for the state of Arizona.

In 1987, Mr. Chen obtained the M.S. degree, also in Electrical Engineering, from the University of Illinois, where he was employed first as a teaching assistant for the Department of Electrical and Computer Engineering from 1984 to 1986 and subsequently as a research assistant at the Center for Reliable and High-Performance Computing from 1986 to 1990. He received the Harold L. Olesen Award for Excellence in Undergraduate Teaching by a Graduate Student in 1986, and consulted for the School of Veterinary Biosciences on a project involving the cycle of the seminiferous epithelium from 1990 to 1992.