May 1992

UILU-ENG-92-2214
CRHC-92-08

Center for Reliable and High-Performance Computing

DESIGN AND SCHEDULING FOR PERIODIC CONCURRENT ERROR DETECTION AND RECOVERY IN PROCESSOR ARRAYS

Yi-Min Wang, Pi-Yu Chung, and W. Kent Fuchs

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

Design and Scheduling for Periodic Concurrent Error Detection and Recovery in Processor Arrays

Yi-Min Wang, Pi-Yu Chung and W. Kent Fuchs

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Correspondent: Yi-Min Wang

Coordinated Science Laboratory
1101 W. Springfield Ave.
University of Illinois
Urbana, IL 61801

E-mail: ymwang@crhc.uiuc.edu
Phone: (217) 244-7161
FAX: (217) 244-5686

Abstract

Periodic application of time-redundant error checking provides the trade-off between error detection latency and performance degradation. The goal is to achieve high error coverage while satisfying performance requirements. In this paper, we derive the optimal scheduling of checking patterns in order to uniformly distribute the available checking capability and maximize the error coverage. Synchronous buffering designs using data forwarding and dynamic reconfiguration are described. Efficient single-cycle diagnosis is implemented by error pattern analysis and direct-mapped recovery cache. A rollback recovery scheme using start-up control for local recovery is also presented.

Acknowledgement: This research was supported in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), and in part by the Joint Services Electronics Program (U.S. Army, U. S. Navy and U. S. Air Force) under Contract N00014-90-J-1270.

1. INTRODUCTION

A variety of processor arrays have been proposed for signal and image processing and scientific computation applications [1]. In order to detect errors produced by faults in these arrays a variety of off-line testing procedures have been developed for detecting permanent faults and concurrent error detection (CED) techniques have been developed for transient and intermittent failures [2]. The focus of this paper is on processor arrays for systolic algorithms and the use of time redundancy techniques for concurrent detection of errors [3,17].

Traditionally, CED is applied continuously to each computation activity so that an error resulting from a fault in the processing element (PE) can be detected immediately. However, when time redundancy techniques are used for error detection this continuous checking scheme may greatly degrade the array performance, e.g., by a factor of two for RESO [4] or alternating logic [5]. For some applications where high processing speed is crucial and error detection latency is tolerable, it may be possible to maintain the desired throughput while keeping a reasonably high error coverage by turning the CED mechanism on and off periodically. Periodic Application of CED (PACED) offers such a trade-off in error latency and probability of error detection versus performance degradation.

Several techniques regarding the utilization of idle processor cycles for CED have been proposed [6-12]. For general-purpose machines with processor-level parallelism, a technique called *saturation* has been introduced for utilizing the idle processors to execute replicated versions of tasks and employing majority voting to determine the output [6]. For processors with multiple pipelined functional units, like the Cray-1, RESO has been applied to the idle function units and was shown to equip the scalar unit with error checking capability at the cost of minor performance degradation [7]. Another recently proposed technique, called Available-Resource Control-flow monitoring (ARC) [8], is aimed at the resource parallelism of instruction-level

parallel processors such as superscalar and Very Long Instruction Word (VLIW) processors. The idle resources in these processors were utilized to detect the control-flow errors.

In the area of systolic architectures, one approach has been developed to take advantage of the existing bypassing links in a reconfigurable array to pass the same input data to two adjacent PEs and then compare the outputs to do the error detection [9]. A control bit, called *test token*, was inserted periodically from outside and passed along the array to determine when a particular PE should invoke a duplicated operation on its neighbor. Related results using error checking code to achieve algorithm-based fault tolerance for a systolic sorter have also been developed [10].

The incorporation of CED capability in systolic arrays for band matrix multiplication has been developed with design parameters such as throughput latency, per-cycle PE utilization rate and I/O bandwidth [11]. The arrays were required to have a per-cycle PE utilization rate less than 50% in order to leave room for the RESO-based CED technique. Flexible designs were proposed [12] which allow the user to either employ the full throughput rate capability of the system or trade off the throughput rate for greater reliability.

In the initial description of the general concept of periodic application of CED (PACED) [3] by Chen et al., error pattern analysis was performed only for a specific set of PACED parameters and the actual implementation was not discussed. The major contribution of this current paper is that we start from a general formulation by defining a set of PACED parameters which are optimized to achieve the maximum error coverage and reduce the hardware cost.

The PACED implementation considered utilizes the following properties:

(1) Each PE is capable of performing time-redundant computation checking for itself as well as input code checking for the possibly erroneous output data produced and propagated by previous PEs.

(2) A single fault is present between the time of the initial fault occurrence and error detection.

The processor arrays considered in this study are unidirectional linear processor arrays consisting of Q processing elements with inputs entering from the top and left [1,13,14]. A PACED array driven by the original clock and equipped with the capability of concurrent error detection and automatic error recovery is shown in Fig. 1. The control logic consists of circuitry to perform buffering, diagnosis, rollback and start-up control.

The outline of the paper is as follows: Section 2 establishes the system parameters; Section 3 gives the optimization of system parameters with respect to various metrics; Section 4 proposes the required design changes for data buffering, error diagnosis and recovery; Section 5 concludes the paper.

2. SYSTEM PARAMETERS

For two PEs in our processor array, PE_i is *upstream* of PE_j and PE_j is *downstream* from PE_i if $i < j$. PEs may not be identical, however each has approximately the same processing time

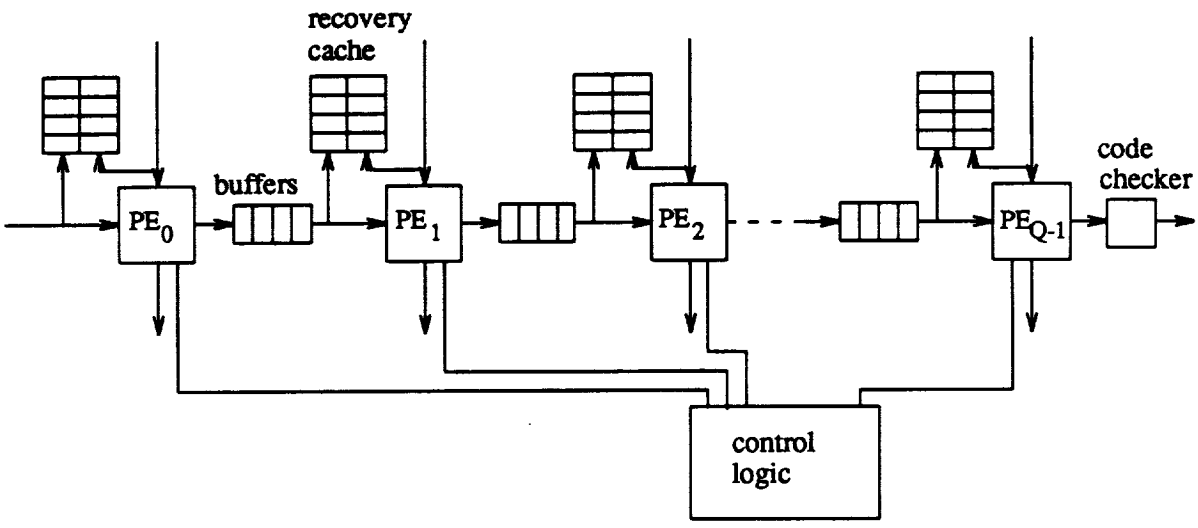


Figure 1. Block diagram of the PACED array with buffering and recovery caches.

so that, without PACED, the array forms a balanced pipeline with clock cycle time equal to a time units. When CED is applied, each PE needs another b time units to perform error checking. For the purpose of preserving the synchronous nature of the original processor array, b is rounded off to multiples of a, $b = ka$. Therefore, for example, k = 1 corresponds to 100% time overhead. The entire activity applied to a certain set of data at each PE is called a *computation cycle with or without checking*, as opposed to the physical clock cycle which always take a time units. At the beginning of a clock cycle, each PE reads from its local counter or a global counter the *checking bit* to determine whether it should perform the checking (1) or not (0). *Checking patterns* are the plots of checking bits as a function of computation cycle number as shown in Fig. 2.

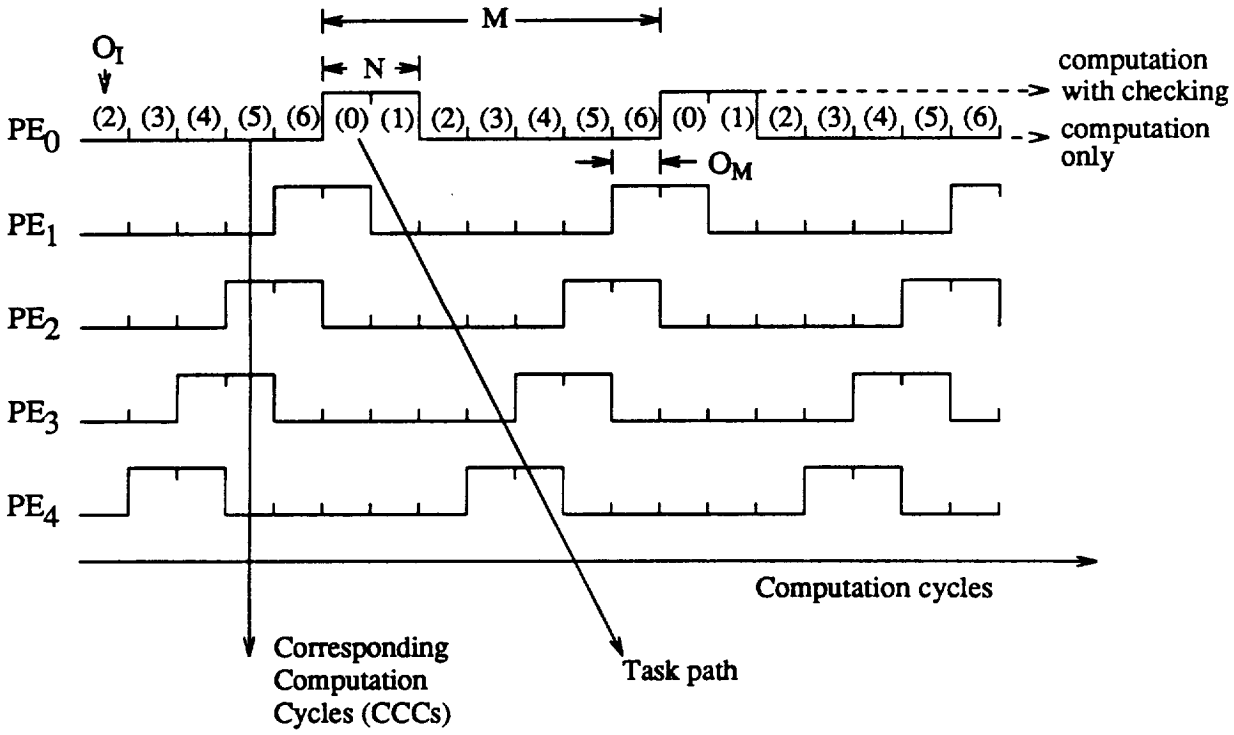


Figure 2. Checking pattern as a function of computation cycle number.

The basic idea of PACED is to schedule the checking patterns with the same checking frequency among PEs. Therefore, while some PEs are executing computation cycles with checking, some are not. The *Corresponding Computation Cycles (CCCs)* are defined to be all those cycles on different PEs which were originally executed at the same time in the array without CED. A *task* is defined to consist of all the activities applied to each input data by the processor array to obtain the corresponding output. The *task path* consists of all those cycles on different PEs at which a certain task is processed as it travels across the array. Each set of checking patterns is characterized by the following four parameters all in terms of computation cycles :

(1) M : length of one period;

(2) N : length of one checking burst;

(3) O_M : offset between checking patterns for adjacent PEs;

(4) O_I : initial offset (with respect to computation cycle 0) of the checking pattern for the first PE. The numbering of the computation cycles is shown in Fig. 2.

While the checking pattern plot in terms of computation cycle as in Fig. 2 is used to illustrate the idea of PACED, it is more convenient to use the Task/PE diagram shown in Fig. 3 for our analysis. In such a diagram, the checking patterns are adjusted so that each column corresponds to a single task path. The offset between adjacent patterns, J, becomes O_M plus one. The Task/PE diagram will be used to analyze problems related to computation cycle such as Error Detection Latency (EDL) analysis and diagnosis.

It can be shown that the choice of O_I does not affect the analysis. Moreover, we will consider N and $\frac{N}{M}$ as two of the parameters instead of M and N. Therefore, the three parameters involved in the the optimization problem will be N, $\frac{N}{M}$ and O_M.

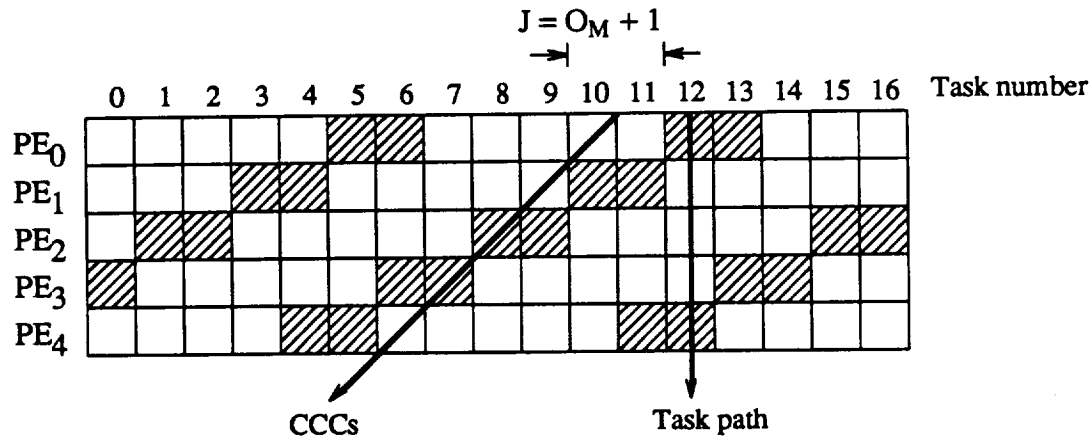


Figure 3. Task/PE diagram of the same array as in Fig. 2 with shaded squares representing computation cycles with checking.

3. OPTIMIZATION OF SYSTEM PARAMETERS

3.1. Performance Analysis

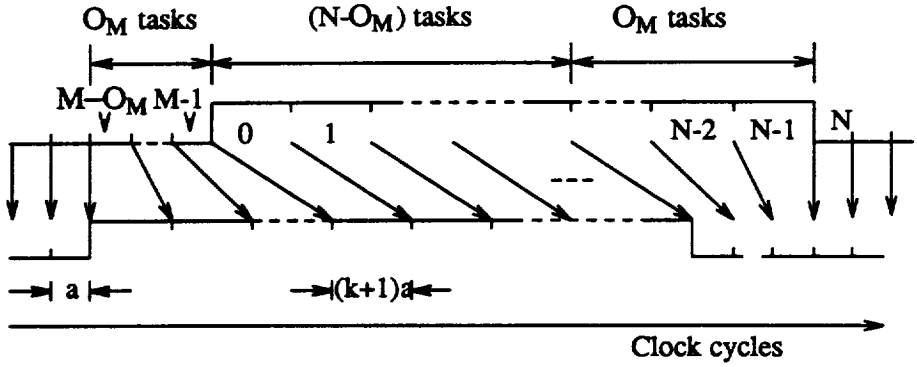
It is intuitive that $\frac{N}{M}$ is a measure of the checking frequency and, therefore, a measure of time overhead. However, because of the imbalance introduced by PACED, the wait time between the output of the upstream PE and the input of the downstream PE of each adjacent pair constitutes another time overhead in addition to the checking time. The total execution time of a certain task is then given as

Total execution time = computation time + checking time + wait time

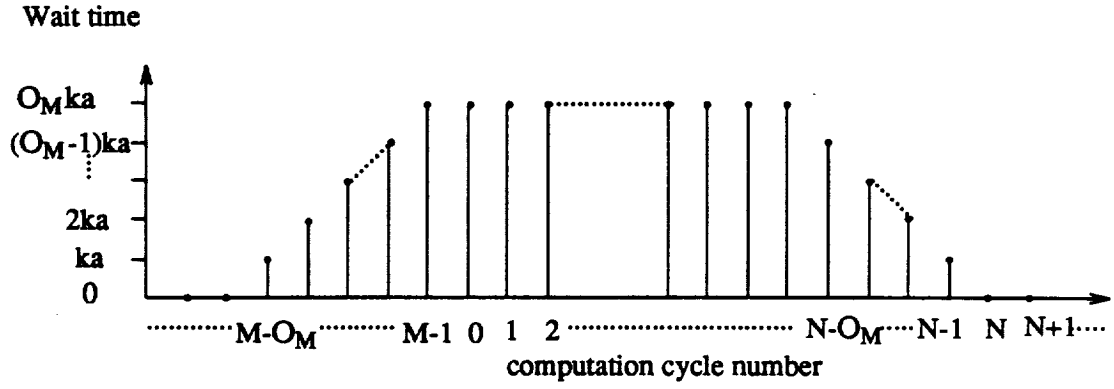
Fig. 4(a) shows the task waiting pattern with each arrow starting from the time the upstream PE outputs the data and pointing to the time the downstream PE reads in the data. The wait time after computation cycle j, Wait(j), in terms of number of clock cycles has the following dependence on O_M as well as on N and M and is shown in Fig. 4(b).

$$\text{Wait(j)} = \begin{cases} O_M \times k & 0 \le j \le N - O_M - 1 \\ (N - j - 1) \times k & N - O_M \le j \le N - 1 \\ 0 & N \le j \le M - O_M - 1 \\ (O_M - M + j + 1) \times k & M - O_M \le j \le M-1 \end{cases}$$

Therefore, fixing $\frac{N}{M}$ does not necessarily fix the time overhead. However, the most important performance measure for a processor array is the *throughput* instead of the total execution time of each single task. As shown in Fig. 4(a), although the data has to wait for the downstream



(a)



(b)

Figure 4. (a) Task waiting pattern between adjacent PEs (b) wait time as a function of computation cycle number.

PE to become free, the PEs are always kept busy. Therefore, the processor array will produce M outputs for every (M+N×k) clock cycles and the throughput can be calculated as

$$\text{throughput} = \frac{M}{(M+N\times k)a} = \frac{1}{(1+\frac{N}{M}\times k)a}$$

which is independent of O_M.

For N = 0, the PACED array reduces to the original array without CED which has the highest throughput $\frac{1}{a}$ but no error detection capability. For N = M, PACED reduces to continuous checking which has the lowest throughput $\frac{1}{(1+k)a}$ but can detect errors without latency. The problem of optimal scheduling for PACED is then formulated as: given a throughput requirement $\frac{1}{(1+\frac{N}{M}\times k)a}$, how to choose O_M to minimize the error detection latency and maximize the error coverage.

3.2. Potentially Infinite Error Detection Latency

There are two cases in which transient faults occurring at certain cycles will have infinite error detection latency (EDL), which means that the errors will escape with 100% probability.

Case 1 : Improper choice of the values for M, N and O_M.

According to the task/PE diagram in Fig. 3, as a task travels through the array, it can be viewed as advancing in the checking pattern with a speed of J computation cycles per PE where $J = O_M + 1$. If we can make sure that at least one cycle with checking, i.e., $0 \le j \le N-1$, is on the path of each task, we can prevent the infinite EDL from occurring under the assumption that errors can not be masked during the propagation. (Error masking is considered in Section 3.5.) The following Lemma 1 for proving Fermat's Little Theorem [15] is used to prove Theorem 1.

LEMMA 1. The M numbers $0 \bmod M$, $J \bmod M$, $2J \bmod M, \ldots,$ $(M-1)J \bmod M$ **consist of precisely d copies of the M/d numbers 0, d, 2d, ... , (M/d - 1)d where d = gcd(M,J) (gcd stands for greatest common divisor.)**

LEMMA 2. With the same notation in Lemma 1, define the remainder set $R_V = \{V+nd \mid 0 \le n \le M/d-1\}$ **for** $0 \le V \le d-1$, **then**

(1) $(j+i\times J) \bmod M \in R_{j \bmod d}$ **for** $0 \le j \le M-1$ **and all non-negative integer i.**

(2) $\{(j+i\times J) \bmod M \mid 0 \le i \le M-1\} = R_{j \bmod d}$, **where d = gcd(M, J). More precisely,** $(j + i \times J) \bmod M$, $0 \le i \le M - 1$ **contain d copies of each of the elements in** $R_{j \bmod d}$.

Proof. See Appendix.

THEOREM 1. Except for some faults occurring in the last (M-1) PEs of the array, all transient faults have finite EDL (less than M) if and only if gcd(M,J) = d ≤ N.

Proof. By the task/PE diagram in Fig 3, a task entering PE0 at cycle j will be processed by PEi at cycle $(j + i \times J) \bmod M$. By Lemma 2(1), $(j + i \times J) \bmod M \in R_{j \bmod d}$. If $N < d$, for any j such that $N \le j < d$, every integer in $R_{j \bmod d}$ is greater than N - 1, hence for all non-negative integers i, $(j + i \times J) \bmod M > N - 1$, which means a task enters PE 0 at such cycle j will never be checked, resulting in infinite EDL if it is affected by some transient fault.

Conversely, if $d \le N$, $(j \bmod d) \le N - 1$. By Lemma 2(2), we have $\{(j + i \times J) \bmod M \mid 0 \le i \le M - 1\} = R_{j \bmod d}$, which means that an erroneous task produced at any cycle j will be checked at least once at cycle (j mod d) by the faulty PE itself or one of its (M-1) immediate downstream PEs. Therefore, as long as the faulty PE is not one of the last (M-1) PEs of the array, the error will be detected. □

Case 2 : Faults occurring near the end of the array.

As long as the CED is not applied continuously, EDL must exist. When a transient fault occurs at some PE in a computation cycle with EDL larger than the number of the downstream PEs from it, the error will escape. This phenomenon exists no matter how we schedule the checking patterns (only the severity varies). One possible solution is to add a code checker with lower complexity and higher reliability at the end of the array to perform continuous code checking in order to intercept the escaping errors if desired.

3.3. Uniform Distribution of Checking Capability

When the checking frequency is set to $\frac{N}{M}$, it is just an average over all tasks and does not necessarily guarantee that each task will be processed with checking $\frac{N}{M}$ of the time along its path. Since all tasks have equal significance, it is desirable to schedule the checking patterns such that each task is treated as uniformly as possible. Theorem 2 gives the condition for this purpose.

THEOREM 2. Except for the difference due to the fact that array length Q might not be a multiple of M, the checking capability is uniformly distributed among all tasks if gcd(M,J) = d divides N.

Proof. If $N = m\,d$, where m is a positive integer, for every R_V, $0 \le V \le d - 1$, the first m elements $V + n\,d$, $0 \le n \le m - 1$, are less than N and others are not. By Lemma 2, $(j + i \times J) \bmod M$, $0 \le i \le M - 1$ contain d copies of each of the elements in $R_{j \bmod d}$, which implies that a task entering PE_0 at any cycle j will be checked $m \times d$ times among M computation cycles. Therefore, all tasks are fairly treated and checked with the same frequency $\frac{m \times d}{M} = \frac{N}{M}$. □

3.4. Minimization of Maximum Error Detection Latency

The EDL(j) of both permanent and transient faults occurring at computation cycle j with checking ($0 \le j \le N-1$) are defined to be zero in terms of number of computation cycles because they are detected immediately. The EDL(j) of other non-checking cycles under the constraint that $1 \le J \le N$ are given by Lemma 3 with superscript "C" indicating the error is detected as a computation error and "I" stands for input error. We can prove the optimal solution under this constraint actually achieves the optimality for general values of J. For the rest of this paper, we will assume the duration of a transient fault is much less than the clock cycle time so that it can only affect the result of one computation[1].

LEMMA 3. For $1 \le J \le N$ and $N \le j \le M-1$,

(1) under transient faults :

$$EDL^I(j) = \left\lceil \frac{M-j}{J} \right\rceil ; \; EDL^C(j) = \infty ,$$

(2) under permanent faults :

$$EDL^I(j) = \left\lceil \frac{M-j}{J} \right\rceil ; \; EDL^C(j) = M-j.$$

Proof. (1) Because we can view a task traveling across the processor array as advancing on the checking pattern with J cycles per step, we have the following formula for the EDL of a transient fault for general J:

$$EDL^I(j) = \left\lceil \frac{rM-j}{J} \right\rceil \text{ where } r = \min\left\{ \left\lfloor \frac{j+i\times J}{M} \right\rfloor \mid 0 \le (j+i\times J) \bmod M \le N-1,\ i \ge 0 \right\}.$$

To our knowledge, there is no closed-form solution for this general problem. The constraint $1 \le J \le N$ will make sure that an error caused by a fault occurring at a non-checking cycle be

[1]Most of the results will still be valid without this assumption except for more complicated error pattern analysis described later.

captured by the next checking burst (r = 1) of some downstream PE. Therefore,

$$EDL^I(j) = \left\lceil \frac{M-j}{J} \right\rceil$$

Under the assumption about the length of the transient faults, a PE can not detect such faults occurring at non-checking cycles, so the $EDL^C(j)$ is infinity.

(2) A permanent fault can be considered as consisting of a large number of transient faults occurring at consecutive computation cycles. Using the above formula, we have, for permanent fault,

$$EDL^I(j) = \min_{0 \le q < M-j} \left\{ \left\lceil \frac{M-(j+q)}{J} \right\rceil + q \right\}.$$

By manipulating the expression inside the bracket,

$$\left\lceil \frac{M-(j+q)}{J} \right\rceil + q = \left\lceil \frac{M-j+(J-1)q}{J} \right\rceil \ge \left\lceil \frac{M-j}{J} \right\rceil \quad \text{for } J \ge 1;$$

hence

$$EDL^I(j) = \left\lceil \frac{M-j}{J} \right\rceil$$

For a permanent fault starting at a non-checking cycle, the faulty PE will detect it as a computation error as soon as it enters next checking burst. Therefore, $EDL^C(j) = M - j$. □

LEMMA 4. For $1 \le J \le N$ and $N \le j \le M-1$, if we define EDL(j) to be the latency until the first error indication (computation or input error), then for both permanent and transient faults : $EDL(j) = \left\lceil \frac{M-j}{J} \right\rceil$.

Proof. This follows immediately from Lemma 3, because $\left\lceil \frac{M-j}{J} \right\rceil < \infty$ and $\left\lceil \frac{M-j}{J} \right\rceil \le M-j$, where equality holds for J = 1 or M - j = 1. Hence,

$$EDL(j) = \min\left\{EDL^{I}(j), EDL^{C}(j)\right\} = EDL^{I}(j) = \left\lceil \frac{M-j}{J} \right\rceil \quad . \ \square$$

Next we give some definitions for proving the optimal scheduling.

DEFINITION 1. Given a sequence of n numbers S = (s_1, s_2, ... ,s_n), $\Pi(S)$ is defined to be a nondecreasing permutation of S, i.e., $\Pi(S) = (\pi_1(S), \pi_2(S), \ldots, \pi_n(S))$ is a permutation of S where $\pi_1(S) \le \pi_2(S) \le \cdots \le \pi_n(S)$.

DEFINITION 2. Given two sequences of n numbers S and T, we define $S \le T$ if $s_i \le t_i$ for $1 \le i \le n$.

LEMMA 5. Define $E_J = (e_{1J}, e_{2J}, \ldots, e_{(M-N)J}) = (EDL_J(M-1), EDL_J(M-2), \ldots, EDL_J(N))$ for $0 \le J \le M-1$, then

(1) $\Pi(E_J) = E_J$ for $1 \le J \le N$

(2) $E_J \ge E_{J+1}$ for $1 \le J \le N-1$

(3) $\Pi(E_N) \le \Pi(E_J)$ for all integer J.

Proof. (1) By Lemma 4,

$$e_{iJ} = EDL_J(M-i) = \left\lceil \frac{M-(M-i)}{J} \right\rceil = \left\lceil \frac{i}{J} \right\rceil \text{ for } 1 \le J \le N \text{ and } 1 \le i \le M-N ; \qquad (1)$$

hence

$$e_{iJ} \le e_{jJ} \quad \text{for } 1 \le i \le j \le M-N$$

$$\Pi(E_J) = E_J \quad .$$

(2) By Eq. (1) we also have $e_{iJ} \ge e_{i(J+1)}$ for $1 \le i \le M-N$; hence

$$E_J \ge E_{J+1} \text{ for } 1 \le J \le N-1 \quad .$$

(3) By (1) and (2), we have proved $\Pi(E_N) \le \Pi(E_J)$ for $1 \le J \le N$. For general values of J, the proof is by contradiction. Suppose there exists J such that $\Pi(E_N) \le \Pi(E_J)$ is not true. This means there exists i, $1 \le i \le M-N$ such that $\pi_i(E_N) > \pi_i(E_J)$ and

$$\pi_i(E_N)-1 \geq \pi_i(E_J) \geq \cdots \geq \pi_1(E_J) \geq 1. \tag{2}$$

Because $\pi_i(E_N) = e_{iN} = \left\lceil \frac{i}{N} \right\rceil$ and $\frac{i}{N} + 1 > \left\lceil \frac{i}{N} \right\rceil$ by definition, we have $\frac{i}{N} + 1 > \pi_i(E_N)$ and

$$\frac{i}{\pi_i(E_N)-1} > N. \tag{3}$$

Hence, by Eqs. (2) and (3), there must exist m, $1 \leq m \leq \pi_i(E_N)-1$ such that more than N elements of $\{\pi_1(E_J), \cdots, \pi_i(E_J)\}$ are equal to m. However, for a checking burst of length N, the maximum number of non-checking cycles with the same EDL is N for any of the EDL values. Therefore, we have reached a contradiction and $\Pi(E_N) \leq \Pi(E_J)$ for all J. □

THEOREM 3. The maximum EDL, EDL_J^{max}, is minimized by setting J = N and the minimum value is $\left\lceil \frac{M-N}{N} \right\rceil$.

Proof. By (3) of Lemma 5, $EDL_J^{max} = \pi_{M-N}(E_J) \geq \pi_{M-N}(E_N) = EDL_N^{max}$ for all integer J. Hence J = N minimizes the maximum EDL and

$$EDL_N^{max} = \max_{N \leq j < M} \left\lceil \frac{M-j}{N} \right\rceil = \left\lceil \frac{M-N}{N} \right\rceil . \quad \square$$

3.5. Minimization of Error Escape Probability

The price paid by PACED to maintain the desired throughput is lower error coverage. The longer the error detection latency, the larger the possibility that an error will be masked during the propagation and escapes. Assume the probability that an error will be masked at any computation cycle is p_m. The error escape probability, $P_{esc}(j)$, for a transient fault occurring at computation cycle j is then given by $P_{esc}(j) = 1 - (1 - p_m)^{EDL_J(j)}$, $0 \leq j \leq M-1$.

THEOREM 4. The average error escape probability is minimized by setting J = N.

Proof. Assume a transient fault occurs at each cycle j, $0 \le j \le M-1$, with equal probability. The average error escape probability is

$$\sum_{j=0}^{M-1} \frac{1}{M} \times P_{esc}(j) = \frac{1}{M} \sum_{j=0}^{M-1} \left[1-(1-p_m)^{EDL_J(j)}\right]$$

$$= \frac{1}{M} \sum_{j=N}^{M-1} \left[1-(1-p_m)^{EDL_J(j)}\right]$$

because $EDL(j) = 0$ for $0 \le j \le N-1$. By (3) of Lemma 5,

$$\frac{1}{M} \sum_{j=N}^{M-1} \left[1-(1-p_m)^{EDL_J(j)}\right] = \frac{1}{M} \sum_{i=1}^{M-N} \left[1-(1-p_m)^{\pi_i(E_J)}\right]$$

$$\ge \frac{1}{M} \sum_{i=1}^{M-N} \left[1-(1-p_m)^{\pi_i(E_N)}\right]$$

$$= \frac{1}{M} \sum_{j=N}^{M-1} \left[1-(1-p_m)^{EDL_N(j)}\right] \text{ for all integer J.}$$

Hence, setting $J = N$ minimizes the average error escape probability. □

By Theorems 3 and 4, we conclude that for a given $\frac{N}{M}$, J should be set equal to the length of the checking burst N, i.e., the pattern offset between adjacent PEs should be $O_M = N - 1$, in order to minimize the maximum EDL and maximize the error coverage. This optimality is independent of the choice of N.

3.6. Summary of Optimization Results

Given a fixed checking frequency $\frac{N}{M}$, we choose M and N to be relatively prime in order to minimize both M and N. Minimizing M allows Theorem 2 to more accurately state the condition for uniform distribution of checking capability. We will show in the next section that minimizing N can minimize the hardware overhead for data buffering. Since J should be equal to N for optimal error detection, we have $gcd(M,J) = gcd(M,N) = 1$ which satisfies both condi-

tions in Theorems 1 and 2. Therefore, the possibility of infinite error detection latency is eliminated and the checking capability is uniformly distributed among all tasks.

4. DESIGN CHANGES

4.1. Synchronous Buffering Design

By scheduling checking patterns among PEs, resource (PE) conflict may occur when a PE is still checking old data but new data has been produced by its upstream PEs. It was shown in Fig. 4(b) that the maximum wait time is equal to $O_M \times k$ clock cycles. Hence, it is adequate to insert $O_M \times k$ buffers between each adjacent PE pair, driven by the original clock[2]. Since O_M should be equal to N-1 for optimal scheduling, the number of buffers will decrease as N decreases. Therefore, choosing M and N to be relatively prime also minimizes the hardware overhead for data buffering.

However as also shown in Section 3.1, the wait time, which determines the number of needed buffers, is not a constant but a function of computation cycle number. It becomes clear at this point that some kind of dynamic buffering technique has to be used to make the pipeline flow smoothly and correctly. We propose two such approaches, namely, data forwarding and dynamic reconfiguration.

The data forwarding approach to buffering is described as follows. When a PE is ready to output the processed data, the wait time logic shown in Fig. 5, which monitors the checking bit sequence, has determined the wait time, Wait(j), of current cycle and connected the PE output to the buffer which is Wait(j) stages away from the downstream PE. Once the data is placed into

---

[2]It can be shown that the minimum number of required buffers is equal to $\left\lceil \frac{O_M \times k}{k+1} \right\rceil$. However, more complicated control circuits are needed to reuse the buffers.

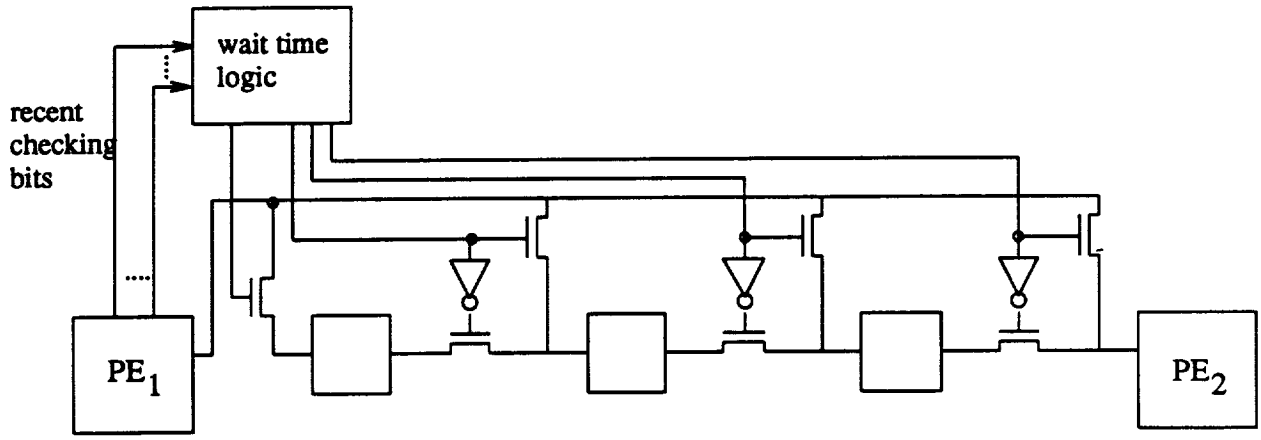


Figure 5. Buffering by data forwarding

the appropriate buffer, the synchronous buffering design will ensure the data arrives at the down-stream PE at the correct clock cycle.

As an alternative, since the number of buffers needed varies with time, we can treat the extra buffers at each clock cycle as being "faulty" and use the Diogenes approach [16] to dynamically reconfigure the "buffer arrays" by bypassing the "faulty" ones. A shifter clocked by the falling edge of the clock (assume the PEs are clocked by the rising edge) is used to set up the proper configuration of the buffers for the next data movement. The basic rule is :

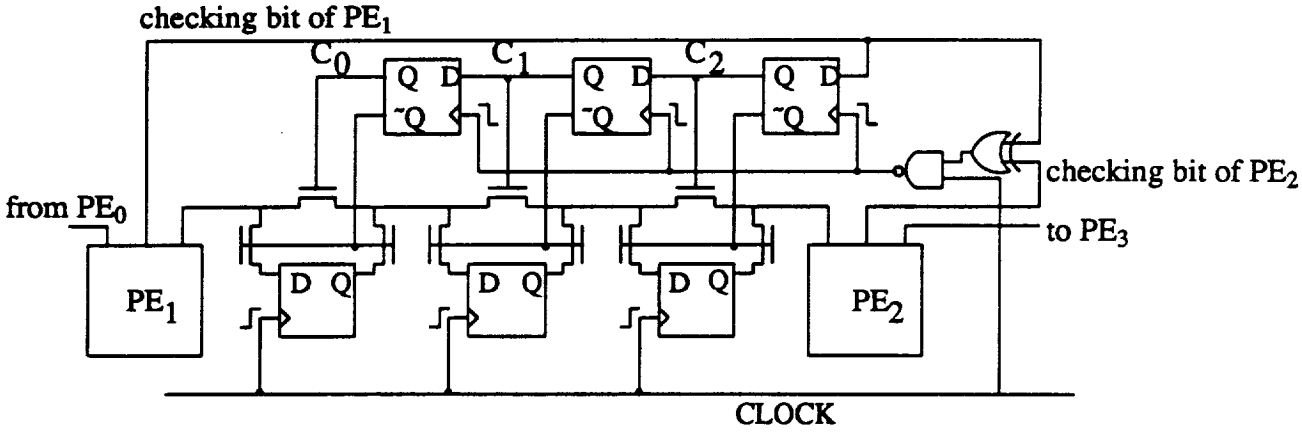


Figure 6. Dynamic reconfiguration circuit for buffering using Diogenes approach

(1) Include one more buffer if the downstream PE is in a computation cycle with checking while the upstream one is not;

(2) Bypass one more buffer if the upstream PE is in a computation cycle with checking while the downstream one is not;

(3) Maintain the current configuration (by disabling the clock input of the shifter) if the two PEs are both checking or non-checking.

Because of the regular pattern of wait time variation (Fig. 4(b)), the reconfiguration circuit is very simple, as shown in Fig. 6. An example showing the correct buffering at each clock cycle by using the reconfiguration approach is given in Fig. 7.

4.2. Diagnosis

Because permanent and transient faults occurring at different computation cycles will result in different combinations of computation and input error indications after various length of latency, it is important for diagnosis to analyze all possible error patterns and classify the faults into several categories according to their resultant error patterns.

Since the error escape probability is related to EDL in terms of computation cycles, and in order to make the diagnosis procedure independent of the checking overhead k, we will use the task/PE diagram (Fig. 3) and the error indications from CCCs for error pattern analysis. However, for a PACED array, the CCCs do not happen at the same clock cycle. It would be unacceptable if we have to wait for all the CCCs to finish before the analysis because that will delay the diagnosis and rollback by a considerable number of clock cycles. The following proof gives the upper bound for the number of error indications by any set of CCCs for $2 \leq J \leq N$. (The case when $J = 1$ will result in peculiar error patterns which can not share the same diagnosis and rollback procedures with other choices of J. Since $J = 1$ also results in large EDL, it will be excluded

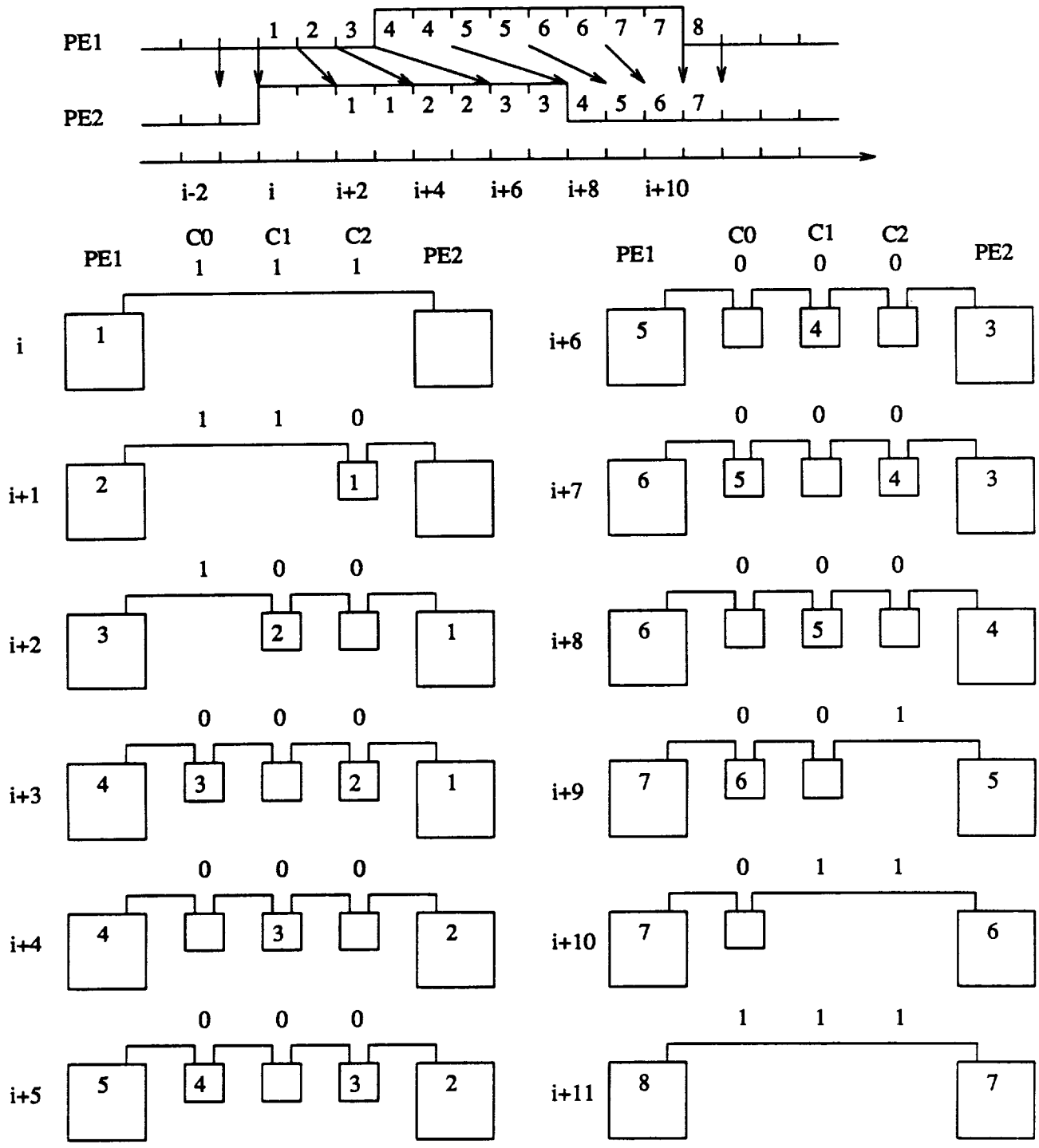


Figure 7. Example of dynamic reconfiguration for buffering. The parameters are N = 4, O_M = 3 and k = 1. When C_i is 1, the corresponding "faulty" buffer is bypassed.

from future discussion.)

THEOREM 5. For $2 \le J \le N$, at most two PEs will have the earliest error indications among all the CCCs for a single fault, and they must be adjacent.

Proof. Since a transient fault can only create one erroneous task, only one PE will detect it. For a permanent fault occurring at cycle j, $N \le j < M$, the faulty PE itself will detect a computation error after M–j cycles and the erroneous task produced at cycle $j + q$, $0 \le q < M-j$ will be detected $\left\lceil \frac{M-(j+q)}{J} \right\rceil + q$ cycles after the fault occurrence as an input error by $\left\lceil \frac{M-(j+q)}{J} \right\rceil$th downstream PE from the faulty PE. For $J \ge 2$ and $q \ge 2$, we have

$$\left\lceil \frac{M-(j+q)}{J} \right\rceil + q = \left\lceil \frac{M-j+(J-1)q}{J} \right\rceil \ge \left\lceil \frac{M-j}{J} + \frac{1}{2} \times 2 \right\rceil = \left\lceil \frac{M-j}{J} \right\rceil + 1$$

which is larger than the $EDL(j) = \lceil \frac{M-j}{J} \rceil$. Hence, the only erroneous tasks which will possibly be detected as the earliest input error indications are the ones produced at cycle j and j+1.

If $M-j = 1$, the two earliest error indications with $EDL(M-1) = 1$ are the computation error detected by the faulty PE and the input error detected by the immediate downstream PE. If $M-j \ge 2$, $(M-j) > \left\lceil \frac{M-j}{J} \right\rceil$ for $J \ge 2$, the two possible earliest error indications are both input errors detected by two adjacent PEs executing at CCCs ($\left\lceil \frac{M-j}{J} \right\rceil - \left\lceil \frac{M-(j+1)}{J} \right\rceil = 0$ or 1 for $J \ge 2$).

□

Therefore, in order to design a diagnosis procedure, we will always keep the pipeline flowing until the immediate downstream PE finishes the CCC once the first error indication is raised by some PE (called the *detective PE*). The number of clock cycles that the detective PE has to wait is equal to the wait time of the current cycle because when the downstream PE is ready to

process the data corresponding to the task resulting in the first error indication, it must have finished processing the previous task at the CCC and setup the error flags.

The next step is to classify all the faults according to their resultant error patterns. The notation is defined as: Class a.b where a = 1 means transient, a = 2 means permanent and b is the further classification within each category. The corresponding error patterns are shown in Fig. 8. "P" stands for permanent fault, "T" for transient fault and "F" can be either "P" or "T". "I" indicates an input error and "C" represents a computation error.

(1) **Class 1.1 and 2.1:** For both permanent and transient faults, Fig. 8(a) represents the case where a computation error indication occurs at computation cycle j, $1 \leq j \leq N-1$. The fault

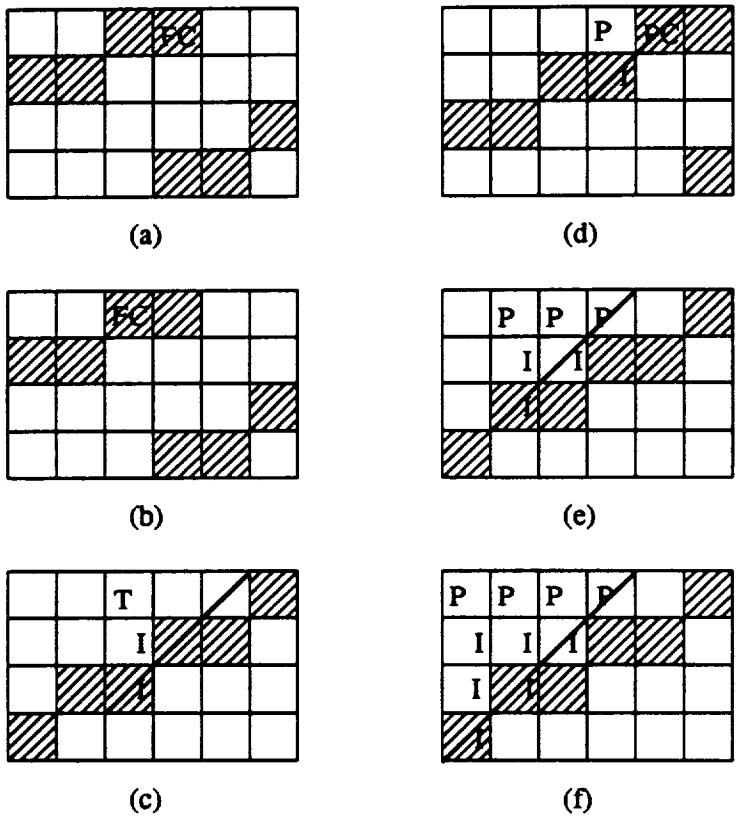


Figure 8. Error pattern analysis. (The thick line passes through all CCCs which are related to the detection of the present fault.)

must have just occurred at the detective PE; otherwise, it should have been detected at earlier cycle with checking. In order to distinguish between permanent and transient faults, the faulty PE is given a second chance to do the recomputation. If the recomputation still sets an error flag, the fault is permanent under the assumption that a transient fault never affects more than one computation; otherwise, it is transient.

A more complicated situation occurs when the computation error flag is raised at cycle 0 because it is possible that the fault is a permanent one which occurred during previous non-checking cycles. However, the proof of Lemma 4 shows that the input error will be detected no later than the computation error for such faults and the two kinds of error will be in the CCCs if and only if such faults occurred at cycle M-1. Therefore, if there is no input error indication in the immediate downstream PE, as in Fig. 8(b), the fault must have just occurred and the faulty PE is given a second chance.

(2) **Class 1.2:** A transient fault occurring at cycle j with $N \le j \le M-1$ will be detected as an input error by the downstream PE which is EDL(j) stages away from the error source PE (Fig. 8(c)).

(3) **Class 2.2:** A computation error detected at cycle 0 and an input error detected by the immediate downstream PE at CCC indicates the fault is permanent and occurred at cycle (M-1) in the upstream PE (Fig. 8(d)).

(4) **Class 2.3:** A permanent fault at cycle j with EDL(j) = EDL(j+1), $N \le j \le M-2$ will be detected by a single downstream PE as an input error (Fig. 8(e)).

(5) **Class 2.4:** A permanent fault at cycle j with EDL(j) = EDL(j+1) + 1, $N \le j \le M-2$ will be detected by two downstream PEs as input errors (Fig. 8(f)).

Among the classes of faults defined in the previous paragraph, Class 1.1, 2.1 and 2.2 are successfully identified by the error pattern analysis and Class 1.2, 2.3 and 2.4 need further diagnosis. The basic idea is to design a recovery cache for each PE for storing recent input data. When an input error is detected by some PE, each upstream PE suspected of producing the error reads from its recovery cache the input corresponding to the erroneous task and uses it as test input to perform recomputation with checking. The computation and input error flags resulting from these recomputations are used as *syndromes* and will uniquely identify the faulty PE and cycle under the assumption of a single fault. Therefore, the diagnosis takes only one computation cycle with checking. Again, for the regularity and simplicity of the diagnosis, the following rules are adopted:

(1) Although it is possible to calculate the exact number of suspects which is less than or equal to the maximum EDL, for each input error detected we will always use maximum EDL as the number of suspects. Because the diagnosis procedure for each PE is done in parallel, this does not increase the time overhead and allows regular hardware connection.

(2) For the faults in Class 2.4, we will ignore the first input error and only use the second error indication for further diagnosis because the second one corresponds to the erroneous task produced earlier.

The success of the above simple diagnosis procedure depends on the capability of each PE to retrieve the correct data from the recovery cache. Because the CCCs are skewed in a PACED array, it is very difficult to determine in which location of the cache the required data resides. The design of the *direct-mapped recovery cache* is aimed at simplifying the searching procedure. Similar to the direct-mapped cache design in the memory hierarchy for general-purpose computers, where each position of the cache can only hold data from certain addresses with identical least significant bits, each position i of our direct-mapped recovery cache can only hold data for

those tasks with id number n such that n mod (cache size) = i. Consequently, as long as we have a recovery cache of sufficient size, i.e. larger than the maximum EDL, so that each data will not have been overwritten by the data from later tasks when it is needed for diagnosis, every suspected PE only has to read the test input from the same position as that in the detective PE and the hardware connection is simplified.

Start-up control is a mechanism to setup the cache correctly once and for all when the pipeline starts flowing, so that whenever new data has to be placed into the cache, it is put into the next position or, when reaching the end, the first position. Fig. 9 shows how the start-up control works. The start-up delay for each PE is computed by accumulating the wait times. Each PE can only start reading in the data after the start-up delay. Therefore, the first data each PE places in the recovery cache will be for task 0 and later data can simply follow on top. The start-up control is also utilized for rollback which is discussed next.

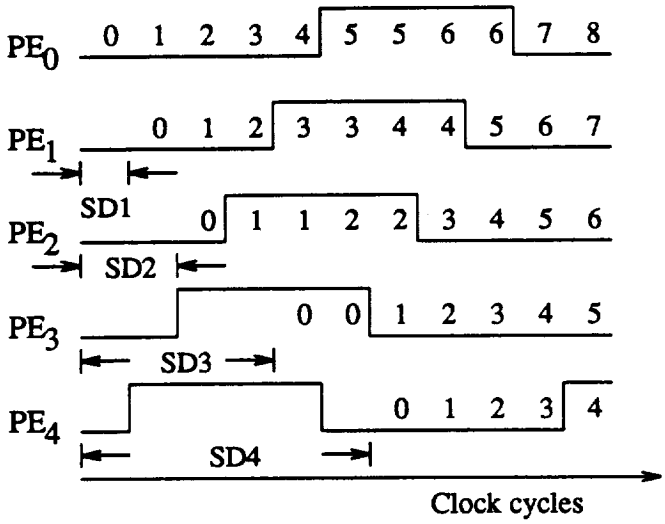


Figure 9. Start-up control (SD: start-up delay)

4.3. Rollback

Once the faulty PE and faulty cycle are identified, if the fault is permanent, a spare PE is brought in to replace the faulty one, and then rollback recovery starts after the reconfiguration; if the fault is transient, rollback directly follows diagnosis, or actually, overlaps with it because the recomputation in diagnosis can be used as the first step in rollback.

The rollback procedure can be divided into two steps: flushing and local recovery. Similar to the simplification in diagnosis, since the rollback is done in parallel for each PE, we will use the same procedure for the recovery of both permanent and transient faults even if the latter results in fewer number of erroneous data.

(1) **Flushing:** First, we define the *erroneous block* to consist of all the following data : (1) data inside the PEs and buffers between error source PE and detective PE; (2) data inside the recovery cache between these two PEs, from the position containing the data corresponding to the erroneous task up to the most recent position. The region enclosed by dash lines in Fig. 10(a) shows the erroneous block for the case where a transient fault occurred in PE1 when task number 7 was being processed and is detected by PE4 as an input error. The first step of rollback is to flush all the data in the erroneous block as shown in Fig. 10.(b).

(2) **Local recovery:** Since the fault only affects the PEs between the error source PE and the detective PE inclusive (called the *local recovery set*), the portion of the pipeline containing all the other PEs are frozen during the rollback. The *local recovery line* is defined to consist of all the PEs in the local recovery set with the erroneous task number. The local recovery scheme is to apply the start-up control to the local recovery line by viewing the local recovery set as a short pipeline, the erroneous task as the first task and the data in the recovery cache of the error source PE, which is correct and thus not flushed, as the input data. The erroneous block is rebuilt as shown in Fig. 10(c)-(h) after which all PEs proceed

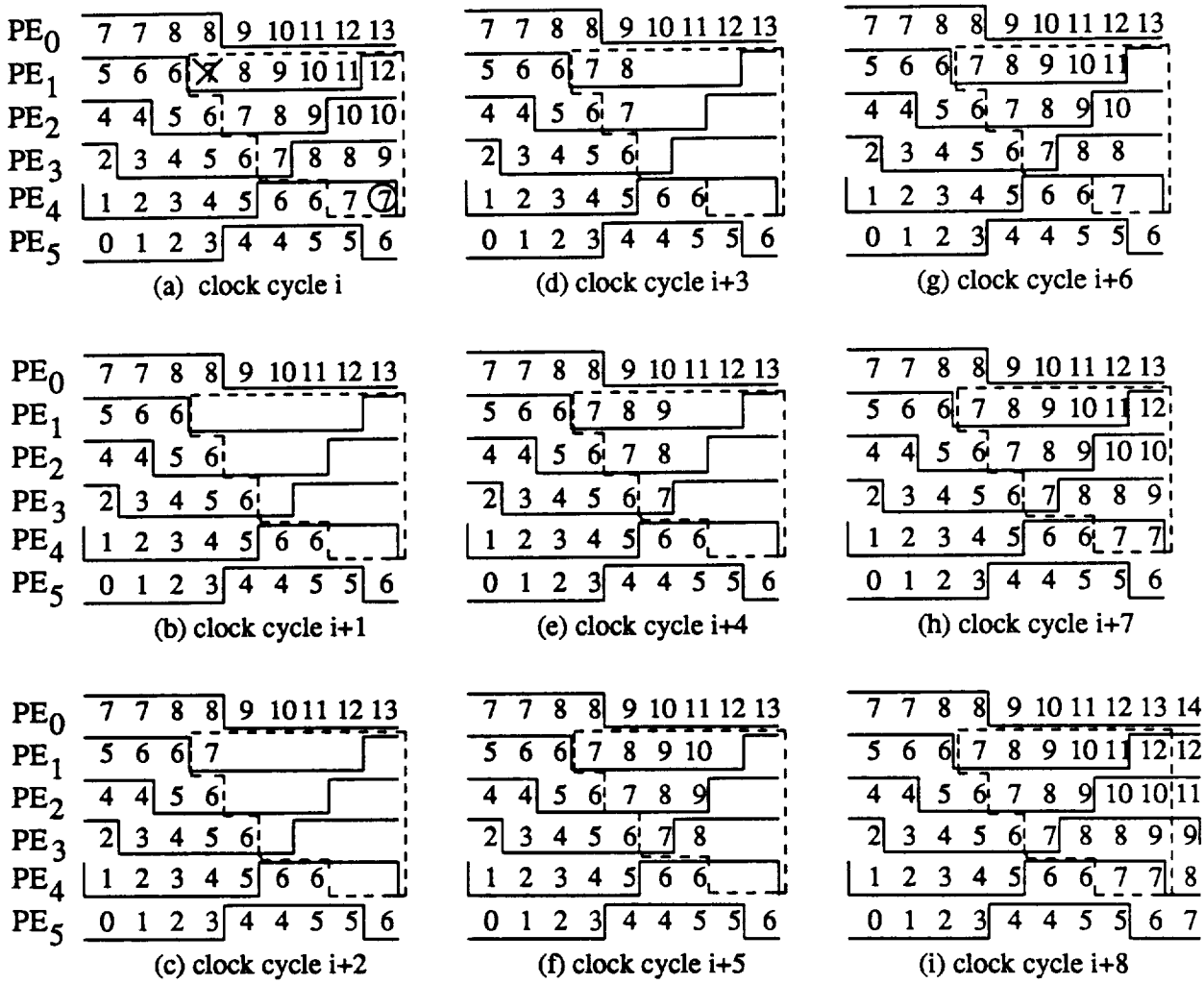


Figure 10. Local recovery procedure (a) fault occurrence and error detection; (b) data flushing; (c)-(h) local recovery using start-up control; (i) resumption of normal processing.

as before (Fig. 10(i)).

4.4. Summary of Design Changes

A PACED array equipped with the proposed design changes was shown in Fig. 1. Error checking circuits are built into each PE for time-redundant computation checking and input code checking. A code checker is added at the end of the array as discussed in Section 3.2. With op-

timal scheduling of checking patterns and 100% overhead time-redundant checking (k=1), N-1 buffers are inserted between each adjacent PE pair and a recovery cache of size $2\times\left\lceil\frac{M-N}{N}\right\rceil$ is attached to each PE for storing incoming data from the top and the left. The control logic is responsible for correct data buffering, start-up control, error diagnosis and local recovery. The techniques described in this paper have been simulated on an Alliant multiprocessors with eight processors to show their correct operations.

5. CONCLUSIONS

It was shown that, for a PACED array with the period of checking pattern equal to M computation cycles, the length of checking burst equal to N computation cycles and fixed throughput (determined by the checking frequency $\frac{N}{M}$), the optimal scheduling in terms of minimizing the maximum error detection latency and error escape probability is achieved by setting the checking pattern offset O_M to N - 1. Also, by choosing M and N to be relatively prime, the hardware overhead for data buffering is minimized and the checking capability is uniformly distributed among the tasks.

Dynamic buffering techniques to preserve the systolic nature and the implementation for rollback recovery under faults were presented. It was shown that the complexity in the diagnosis and recovery process, resulting from the error latency as a trade-off for performance, can be reduced through the use of direct-mapped recovery cache and start-up control. The design flexibility can be further improved by using a programmable control unit.

APPENDIX

Proof of Lemma 2.

$$(j + i \times J) \bmod M = (j + (i \times J) \bmod M) \bmod M$$

$$= (j + (i \bmod M \times J) \bmod M) \bmod M$$

$$\in \left\{ (j + nd) \bmod M \,|\, 0 \le n \le M/d-1 \right\} \text{ by Lemma 1.}$$

Also, $\{(j+i\times J) \bmod M \,|\, 0 \le i \le M-1\} = \{(j+nd) \bmod M \,|\, 0 \le n \le M/d-1\}$, and $(j+i\times J) \bmod M$, $0 \le i \le M-1$ contain d copies of each element in the set on the right hand side.

For $0 \le n \le M/d - \lfloor j/d \rfloor - 1$,

$$\Rightarrow 0 \le j + nd \le M + j \bmod d - d < M$$

$$\Rightarrow (j+nd) \bmod M = j + nd = j \bmod d + (n + \lfloor j/d \rfloor)d = j \bmod d + md,\ \lfloor j/d \rfloor \le m \le M/d-1.$$

For $M/d - \lfloor j/d \rfloor \le n \le M/d - 1$,

$$\Rightarrow M + j \bmod d \le j + nd \le M + j - d < 2M$$

$$\Rightarrow (j + nd) \bmod M = j + nd - M = j \bmod d + (n + \lfloor j/d \rfloor - M/d)d = j \bmod d + md,\ 0 \le m \le \lfloor j/d \rfloor - 1.$$

Therefore,

$$\left\{ (j+nd) \bmod M \,|\, 0 \le n \le M/d-1 \right\} = \left\{ (j \bmod d + md) \,|\, 0 \le m \le M/d-1 \right\}$$

Finally, we have $(j + i \times J) \bmod M \in R_{j \bmod d}$ and $\{(j+i\times J) \bmod M \,|\, 0 \le i \le M-1\} = R_{j \bmod d}$. □

REFERENCES

[1] W. Moore, A. McCabe and R. Urquhart, (eds.) *Systolic Arrays*, Adam Hilger, 1987.

[2] J. A. Abraham, P. Banerjee, C.-Y. Chen, W. K. Fuchs, S. Y. Kuo, A.L. N. Reddy, "Fault tolerance techniques for systolic arrays," *IEEE Computer*, vol. 20, no. 7, July 1987, pp. 65-75.

[3] P. P. Chen, A. N. Mourad and W. K. Fuchs, "Confidence in processor array outputs under periodic application of concurrent error detection," *IEEE Workshop on Defect and Fault Tolerance in VLSI Systems*, Nov. 1990.

[4] W. T. Cheng and J. H. Patel, "Concurrent error detection in iterative logic arrays", *Proc. 14th IEEE International Conference on Fault-Tolerant Computing*, 1984, pp. 286-291.

[5] 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.

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

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

[8] M. A. Schuette and J. P. Shen, "Exploiting instruction-level resource parallelism for transparent control-flow monitoring", Research Report No. CMUCAD-90-42, Dec. 1990, Carnegie-Mellon University.

[9] Y. H. Choi, S. H. Han and M. Malek, "Fault diagnosis of reconfigurable systolic arrays", *Proc. IEEE International Conference on Computer Design, Port Chester, NY*, Oct. 1984, pp. 451-455.

[10] Y. H. Choi and M. Malek, "A fault-tolerant systolic sorter", *IEEE Trans. on Computers*, Vol. 37, No. 5, May 1988, pp. 621-624.

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

[12] R. J. Cosentino, "Concurrent error correction in systolic architectures", *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 1, Jan. 1988, pp. 117-125.

[13] H. T. Kung and M. S. Lam, "Wafer-scale integration and two-level pipelined implementations of systolic arrays", *J. of Parallel and Distributed Computing*, Vol. 1, 1984, pp. 32-63.

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

[15] R. L. Graham, D. E. Knuth, O. Patashnik, "Concrete Mathematics", Addison-Wesley Publishing, New York, 1989.

[16] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant arrays of processors", *IEEE Trans. on Computers*, Vol. C-32, No. 10, Oct. 1983, pp. 902-910.

[17] E. S. Manolakos, "Transient fault recovery techniques for the VLSI processor arrays", *Ph.D. Dissertation,* University of Southern California, May 1989.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

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-92-2214 CRHC-92-08	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 National Aeronautics Space Administration IL Comput. Lab. Aerospace Sys Joint Electronic Services Programs
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Langley VA Chicago, IL Washington, DC
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NAG 1-613 N00014-90-J-1270
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)
Design and Scheduling for Periodic Concurrent Error Detection and Recovery in Processor Arrays

12. PERSONAL AUTHOR(S)
WANG, Yi-Min, Pi-Yu Chung, W. Kent Fuchs

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1992 May 22	32

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	error detection latency performance degradation scheduling, diagnosis, error pattern analysis

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

Periodic application of time-redundant error checking provides the trade-off between error detection latency and performance degradation. The goal is to achieve high error coverage while satisfying performance requirements. In this paper, we derive the optimal scheduling of checking patterns in order to uniformly distribute the available checking capability and maximize the error coverage. Synchronous buffering designs using data forwarding and dynamic reconfiguration are described. Efficient single-cycle diagnosis is implemented by error pattern analysis and direct-mapped recovery cache. A rollback recovery scheme using start-up control for local recovery is also presented.

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

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED