AD-A051 855 CALIFORNIA UNIV LOS ANGELES F/G 9/2
PERFORMANCE EVALUATION OF A PARALLEL SYSTEM PROCESSING FAULT-TO--ETC(U)
1977 K H KIM, M J JENSON F44620-76-C-0061
UNCLASSIFIED AFOSR-TR-78-0454 NL

| OF |
AD
A051 855

END
DATE FILMED
4-78
DDC

Proc. 1977 Int'l Conf. on Parallel Processing
pp. 118-127.

PERFORMANCE EVALUATION OF A PARALLEL SYSTEM PROCESSING FAULT-TOLERANT PROGRAMS†

K. H. Kim and M. J. Jenson
Department of Electrical Engineering
and Computer Science
University of Southern California
Los Angeles, California 90007

Abstract. A parallel (multiprocessor) system processing fault-tolerant programs was developed in [4, 5]. The system performance is evaluated in this paper, using an analytic approach based on stochastic models. The analysis confirms the high effectiveness of a parallel system, under all practical circumstances, in reducing the program execution time increase due to run-time validation and system state saving. It also shows how the system performance is affected by various program characteristics.

1. Introduction

A system architecture for parallel execution of fault-tolerant programs (i.e., programs containing redundancy for the tolerance of residual program errors and/or hardware faults [7]) was developed in [4, 5]. The system was designed to execute block-structured fault-tolerant programs developed by Horning et al. [3]. A fault-tolerant block or recovery block is the basic component containing redundancy in these programs and has the following structure: ensure T by O_1 else-by O_2 else-by ... else-by O_n else-error, where T denotes the validation test, O_1 the primary object block, and O_k ($1 < k \leq n$) the alternate object blocks. All of the object blocks in a fault-tolerant block F compute the same or approximately the same objective function. The validation test T is executed on exit from an object block to confirm that the object block has performed acceptably. The execution of a validation test results in either an acceptance (i.e., confirmation) or a rejection. If accepted, control exits from the fault-tolerant block. If the result produced by an object block is rejected, the next alternate is entered. After the alternate object block finishes its computation, the validation test is repeated. Before an alternate object block is entered, the system state is restored to the state that existed just before entry to the primary object block [1, 2, 3]. To enable this, a state vector that contains the values of all the variables (that may be changed by the object blocks) is saved on entry to a fault-tolerant block.

The goal of the parallel execution is to overlap, as much as possible, execution of object blocks with the validation and system state saving. In this paper, we evaluate the performance of the parallel system. The approach used in this paper for performance evaluation is of an analytic nature and is based on stochastic models for both the parallel system and the sequential system (i.e., one in which the execution of an object block is not overlapped with the execution of a validation test). The evaluation shows the performance gain by parallel execution over sequential execution.

In the next section major characteristics of both an efficient sequential system and a parallel system are compared. Section 3.1 deals with the evaluation of the sequential system. Performance of the parallel system is evaluated in Section 3.2 and compared with the performance of the sequential system in Section 3.3.

2. Distinguishing Characteristics of a Sequential System and a Parallel System

In this section two systems, a sequential system using a memory organization called a recovery cache [1, 3] and a parallel system using a duplex memory [4, 5], are briefly sketched.

The essence of the recovery cache scheme is to save the "original value" of each non-local variable W together with its logical address right before the variable is modified for the first time in a new object block. The original values are thus saved in a compact

† This work was supported in part by the Joint Service Electronics Program under Air Force Contract F44620-76-c-0061.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR-78-0454	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): PERFORMANCE EVALUATION OF A PARALLEL SYSTEM PROCESSING FAULT-TOLERANT PROGRAMS	5. TYPE OF REPORT & PERIOD COVERED: Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): K. H. Kim, M. J. Jenson	8. CONTRACT OR GRANT NUMBER(s): F44620-76-C-0061
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Electrical Engineering and Computer Science, University of Southern California, Los Angeles, CA 90007	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 2305/A9, 61102F, A9
11. CONTROLLING OFFICE NAME AND ADDRESS: AFOSR /NE, Bldg 410 Bolling AFB DC, 20332	12. REPORT DATE: Proc. 1977 Int'l Conf. of Parallel Processing 13. NUMBER OF PAGES: 10
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 11 1977, 12 11 P.	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

BOOK REPRINT:
Proc. 1977 Int'l Cof. on Parallel Processing p.118-127

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

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

A parallel (multiprocessor) system processing fault-tolerant programs was previously developed, ~~in (4,5)~~. The system performance is evaluated in this paper, using an analytic approach based on stochastic models. The analysis confirms the high effectiveness of a parallel system, under all practical circumstances, in reducing the program execution time increase due to run-time validation and system state saving. It also shows how the system performance is affected by various program characteristics.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

072 250

FOB

table structure. For illustration, the fault-tolerant program in Figure 1a is used.

Figure 1b shows a snapshot of the recovery cache taken when primary object block $O_{2.1}$ is in execution. As shown, there is a stack, called the cache stack, used for saving the original values. Similar to the main stack, the cache stack is also divided into regions, one region for each nested fault-tolerant block in the "active" state (i.e., a fault-tolerant block that has been entered but not exited). The top region of the cache stack in Figure 1b contains previous values of non-local variables together with their names (representing logical addresses), i.e., Y2, X1, X2, which have been modified during execution of the current object block $O_{2.1}$. Similarly, the bottom region of the cache stack contains the previous value of non-local variable X1 which had been modified by execution of object block $O_{1.1}$ before $O_{2.1}$ was entered. Figure 1b also shows a flag field in the main stack. The flag attached to a variable indicates whether the original value of the variable has already been saved since the current object block was entered. Thus the flags attached to Y2, X1, X2 in the main stack are currently set.

If the result produced by execution of $O_{2.1}$ fails the validation test V_2, then the top region C_2 of the cache stack can be used to reset the main stack to the state that existed on entry to fault-tolerant block F_2. If it passes the test, execution of F_2 is complete and C_2 is merged into C_1 so that the result will contain previous values of those variables which are non-local to $O_{1.1}$ and have been modified since $O_{1.1}$ was entered. Thus the result will be a single region containing (X1, 9) and (X2, 2). Flags in the main stack are also adjusted such that only flags of X1 and X2 are set. Therefore, the combination of the main and cache stacks usually contains information with which several old state vectors can be reconstructed.

In the case of parallel execution at least two processors are used, a main processor for object block execution and a VR-(validation and recovery) processor or audit processor for execution related to validation and recovery. It is necessary to save a state vector on exit from an object block since the state vector is used by both the main processor and the VR-processor. This is accomplished by simultaneously storing the operand of each WRITE operation into two locations, one in the main stack and the other in the VR-store. When the main processor enters a fault-tolerant block F, a VR-store-segment is created to keep an execution image which consists of records of assignments made by an object block in F. A VR-store-segment consists of two sections, the L-(local variable) section for keeping records of assignments to variables local to the object block in execution and the N-(non-local variable) section for assignment records of non-local variables. A variable local to the object block being entered is allocated one location in the main stack and one location in the L-section of a VR-store-segment. New values assigned to variables that are non-local to the object block in execution are recorded together with the logical addresses (of the variables) in a table structure in the N-section of a VR-store-segment.

For illustration, Figure 1c shows the content of the VR-store at an instant during execution of the program in Figure 1a by a parallel system using a duplex memory. When the main processor entered the program (i.e., the outermost block), VR-store-segment S_0 was created to keep assignment records of local variables X1 and X2. Since there are no variables non-local to the outermost block, S_0 does not contain a N-section. When the main processor entered F_1, VR-store-segment S_1 was created. When non-local variable X1 was assigned the value "8" during execution of object block $O_{1.1}$, a table entry (X1, 8) was made in S_{1N}. Similarly, S_2 was created when the main processor entered F_2 and was filled by execution of object block $O_{2.1}$. The content of the main stack in a duplex memory is that in a recovery cache minus the flag field.

On completion of $O_{2.1}$, the main processor proceeds to the execution of F_3 (which will be imaged in a new VR-store-segment S_3) while the VR-processor starts examining the execution image in S_2 by execution of V_2. If the result produced by execution of $O_{2.1}$ (kept in S_2) fails the validation test V_2, then the non-local variables recorded in S_{2N} (and S_{3N}, if not empty) are those which need to be reset. Segments S_0 and S_1 contain the values of the variables that existed when the main processor entered fault-tolerant block F_2 and their values may be used to reset the main stack. A duplex memory may be implemented such that the previous value can be obtained in a single content-addressable memory (CAM) cycle [4, 5]. If the result of $O_{2.1}$ passes V_2, S_{2L} is discarded and S_{2N} is merged into S_1 so that the result contains the assignment records, of the variables addressable in $O_{1.1}$, made since $O_{1.1}$ was entered. This will result in S_{1L} containing "1","5" and "3" for Y1, Y2, Y3, respectively and S_{1N} containing (X1, 7) and (X2, 8).

Let us now compare the characteristics of the recovery cache scheme for sequential execution with the characteristics of the duplex memory scheme for parallel execution.

1. In both schemes, content-addressable memory modules are needed to obtain an acceptable level of performance in program execution and in the rest of this paper, the use of CAM modules is assumed.

2. The duplex memory takes more space than the recovery cache.

3. The WRITE operation into a non-local variable W involves two steps with the recovery cache, the first step being used for fetching the original value or the flag, while the WRITE operation takes one step (CAM cycle) with the duplex memory. Therefore, the execution of an object block is slower with the recovery cache than with the duplex memory.

4. Overall, it is expected that the recovery cache takes less merging time than the duplex memory. During the execution of a program in which no fault-tolerant block is nested within another fault-tolerant block, there is no merging involved with the recovery cache.

5. The parallel system is slower in recovery because (a) recovery of a variable takes more steps with the duplex memory than with the recovery cache and (b) there are more variables that need to be recovered in the parallel system because while an execution image is being validated, the main processor normally proceeds to the successor block(s).

In summary, the parallel system largely trades recovery time increase for the reduction of total program execution time. There are cases, though highly impractical, where the performance of the parallel system is inferior to the performance of the sequential system. Let α denote the reliability of an object block, i.e., the probability of an average object block producing an accepted execution image. Then there is a lower bound α_L for α such that when $\alpha > \alpha_L$, the parallel system performs more efficiently than the sequential system. This lower bound is one of the values of interest examined in subsequent sections.

3. Performance Evaluation

Given a fault-tolerant program, the average execution time of a fault-tolerant block is defined as the execution time of the program divided by the number of fault-tolerant blocks executed during the program execution. T_s and T_p denote the average execution time of a fault-tolerant block by the sequential system and by the parallel system, respectively. The system throughput is defined as the number of fault-tolerant blocks completed per unit time and is given by the inverse of the average execution time of a fault-tolerant block. We denote the sequential system throughput and the parallel system throughput by THR_s and THR_p, respectively. Throughputs are used in this section as measures of the performance of the sequential system and of the parallel system.

For mathematical tractibility, the following set of global assumptions have been adopted throughout the performance evaluation.

Assumption G

G.1 The programs considered in this analysis are of the type in which no fault-tolerant block is nested within another fault-tolerant block and whose execution becomes a sequential chain of fault-tolerant block executions (Figure 2).

G.2 Primary and alternate object blocks take the same average execution time.

G.3 Each fault-tolerant block contains an unlimited number of alternate object blocks (to eliminate the case of program failure).

In executing a program satisfying assumption G.1, the sequential system does not involve assignment record merging, as mentioned in Section 2. This assumption G.1 is adopted because of the difficulties in (1) dealing with a large spectrum of legitimate program structures, (2) keeping accounts of various execution times during execution of a general program (i.e., a program in which fault-tolerant blocks are nested one within another), etc. However, it is conjectured that results in this paper of performance comparison between two systems for programs satisfying G.1 will not be far different from the results for general programs.

3.1 Throughput Evaluation for the Sequential System

The behavior of the sequential system during execution of a fault-tolerant block is depicted in Figure 3a. The system first enters the "object block execution" state s_o in which the processor executes an object block within the current fault-tolerant block. On completion of an object block, the system enters the "validation" state s_v in which the processor executes the validation test. If the validation results in a rejection, the system enters the "recovery" state s_r, and on completion of the recovery, the system again enters s_o in which the processor executes an alternate object block. If the validation results in an acceptance, the system proceeds to the execution of the successor fault-tolerant block and repeats the above behavior.

During execution of fault-tolerant programs satisfying assumption G, the sequential system continuously repeats the process depicted in Figure 3a. We thus model the system behavior by the following stochastic process for the purpose of evaluating THR_s.

Model S

S.1 There are three states which the sequential system may enter: s_o - object block execution, s_v - validation, and s_r - recovery. (Due to assumption G.1 there is no merging state.)

S.2 The time during which the system is in any state is exponentially distributed.

S.2.1 When the system is in state s_o, the rate gs of generating an execution image (i.e., the probability of the system completing the execution of an object block within an infinitesimal time interval Δt is $gs^*\Delta t$), is $gs = 1/t_{os}$ where t_{os} denotes the mean object block execution time in the sequential system. gs is called the generation rate.

S.2.2 When the system is in state s_v, the rate v of completing the validation, called the validation rate, is $v = 1/t_v$ where t_v denotes the mean validation time.

S.2.3 When the system is in state s_r, the rate rs of completing the recovery, called the recovery rate, is $rs = 1/t_{rs}$ where t_{rs} denotes the mean recovery time in the sequential system.

S.3 The probability of the system entering state s_o after leaving state s_v is α, while the probability of entering state s_r is $\alpha' = 1 - \alpha$.

Figure 3b depicts Model S. Let p_o, p_v, p_r denote the equilibrium probabilities [6] of the system being in s_o, s_v, s_r, respectively. The steady-state behavior of the system is expressed by the following equilibrium equations.

$$p_o \cdot gs = p_r \cdot rs + p_v \cdot v \cdot \alpha$$

$$p_v \cdot v = p_o \cdot gs \qquad (1)$$

$$p_o + p_v + p_r = 1 \text{ (normalizing equation).}$$

Solving Eq. 1, we obtain

$$p_o = rs \cdot v/(gs \cdot v \cdot \alpha' + rs \cdot v + gs \cdot rs)$$

$$p_v = rs \cdot gs/(gs \cdot v \cdot \alpha' + rs \cdot v + gs \cdot rs) \qquad (2)$$

$$p_r = gs \cdot v \cdot \alpha'/(gs \cdot v \cdot \alpha' + rs \cdot v + gs \cdot rs).$$

By definition system throughput is equal to the number of execution images accepted per unit time. Throughput THR_s and its inverse T_s can thus be obtained as follows.

$$THR_s = p_v \cdot v \cdot \alpha$$
$$= rs \cdot gs \cdot v \cdot \alpha/(gs \cdot v \cdot \alpha' + rs \cdot v + gs \cdot rs) \qquad (3a)$$

$$T_s = 1/THR_s$$
$$= (gs \cdot v \cdot \alpha' + rs \cdot v + gs \cdot rs)/(rs \cdot gs \cdot v \cdot \alpha)$$
$$= (1/\alpha) \cdot (t_{os} + t_v) + (\alpha'/\alpha) \cdot t_{rs}. \qquad (3b)$$

3.2 Throughput Evaluation for the Parallel System

In most cases the main processor need not be synchronized with the VR-processor. However, when the next fault-tolerant block to be executed specifies irreversible actions of critical nature, the main processor waits until the VR-processor accepts all the execution images in the queue (i.e., the execution images of the predecessor fault-tolerant blocks) [4, 5]. An execution image generated immediately before a block specifying an irreversible action is entered, is a "synchronizing" execution image (or for short, S-image). The other execution images are "normal" execution images (or N-images).

An abstract representation of the parallel system with unbounded queue is shown in Figure 4. The main processor continuously constructs execution images and puts the completed execution images into the queue of execution images except when (1) the VR-processor stops it on rejection of an execution image and enters the recovery state, or (2) the main processor has generated a synchronizing execution image and put it into the queue. The VR-processor validates execution images in the order of their arrival. When it accepts an execution image, it enters the "merging" state. On completion of merging, it checks if another execution image is waiting in the queue. If an execution image is rejected, the main processor is stopped and recovery is initiated. Recovery involves a sequence of assignment reversals using the assignment records in the execution images and thus can be thought of as a process of "erasing" the execution images in the queue. On completion of the recovery, the queue is empty and the main processor is restarted. The parallel system is thus modeled by the following stochastic process.

Model P

P.1 The state of the system at any instant is characterized by (1) the state of the VR-processor which may be in wait, validation, merging or recovery, and (2) the number and types of execution images in the queue. The state of the main processor is busy or waiting

and is determined by the state of the VR-processor and the state of the queue. Thus each system state is denoted by

$$s_{\text{VR-processor state, queue state}}$$

where (1) VR-processor state = w (wait), v (validation), m (merging), or r (recovery), and (2) queue state = $\emptyset$ (empty), N (one normal execution image), S (one synchronizing execution image), \$ (= N or S), NN, NS, \$N, \$S, NNN, NNS, \$NN, \$NS, ...

Some possible states of the system are shown in Figure 5, where some possible state transitions are also indicated. For example, $s_{v,N}$ is the state where the queue contains one normal execution image which the VR-processor is validating. There are four states which the system may enter from $s_{v,N}$: $s_{v,NN}$ which is entered if the main processor generates another normal execution image; $s_{v,NS}$ which is entered if the main processor generates a synchronizing execution image; $s_{m,\$}$ which is entered if the VR-processor accepts the normal execution image in the queue; and $s_{r,N}$ which is entered if the VR-processor rejects the normal execution image in the queue. In $s_{r,N}$ the system erases the normal execution image in the queue and thereafter enters state $s_{r,\emptyset}$ in which the system erases the partially constructed execution image contained within the main processor. Note that the type of the first image in the queue is not distinguished in some states (*e.g.*, $S_{m,\$N}$). This is because once an execution image is accepted, the system's future behavior is independent of the type of the execution image just accepted.

P. 2 The time during which either processor is in a particular state is exponentially distributed.

P. 2.1 When the main processor is in a busy state, the generation rate gp is $gp = 1/t_{op}$, where t_{op} represents the mean object block execution time (which is different from t_{os}).

P. 2.2 When the VR-processor is in a validation state, the validation rate v is $v = 1/t_v$, where t_v represents the mean validation time.

P. 2.3 When the VR-processor is in a merging state, the rate mp of completing the merging, called the merging rate, is $mp = 1/t_{mp}$ where t_{mp} represents the mean merging time.

P. 2.4 When the system is in a recovery state other than $s_{r,\emptyset}$, the rate rp of erasing an execution image, called the recovery rate, is $rp = 1/t_{rp}$ where t_{rp} represents the mean time for erasing an execution image.

P. 2.5 The size of the partially constructed execution image remaining within the main processor when the system enters a recovery state is assumed to be proportional to the amount of time that the main processor has spent in construction of that execution image. Borrowing a result in the renewal theory, the mean size of the execution image partially constructed (when the system enters a recovery state from a state where the main processor is busy), is the same as the mean size of a completed execution image [6]. Thus when the system is in $s_{r,\emptyset}$, the rate of moving from $s_{r,\emptyset}$ to $s_{w,\emptyset}$ is also rp.

P. 3 The probability of a validation resulting in an acceptance is α as before, while the probability of a rejection is $\alpha' = 1 - \alpha$.

P. 4 The probability of a newly generated execution image being an N-image is η, while that for being an S-image is $\eta' = 1 - \eta$.

Figure 5 depicts Model P. It also shows the notation for the equilibrium probability of the system being in each state $s_{i,j}$. The probabilities are denoted by I (for $s_{w,\emptyset}$), J (for $s_{m,\$}$), z_k, y_k, x_k, w_k, u_0 (for $s_{r,\emptyset}$), u_k, and q_k, where $k = 1, 2, \ldots$ except that there does not exist y_1 nor x_1. The subscript k indicates the number of execution images present in the queue. The steady-state behavior of the system is then expressed by the following equilibrium equations.

(a) $I \cdot gp = J \cdot mp + u_0 \cdot rp + q_1 \cdot rp$

(b) $J \cdot (gp + mp) = (z_1 + w_1) \cdot v \cdot \alpha$

(c) $z_1 \cdot (gp + v) = I \cdot gp \cdot \eta + y_2 \cdot mp$

(d) $z_k \cdot (gp + v) = z_{k-1} \cdot gp \cdot \eta + y_{k+1} \cdot mp$ for $k = 2, 3, \ldots$

(e) $y_2 \cdot (gp + mp) = J \cdot gp \cdot \eta + z_2 \cdot v \cdot \alpha$

(f) $y_k \cdot (gp + mp) = y_{k-1} \cdot gp \cdot \eta + z_k \cdot v \cdot \alpha$ for $k = 3, 4, \ldots$

(g) $x_2 \cdot mp = J \cdot gp \cdot \eta' + w_2 \cdot v \cdot \alpha$

(h) $x_k \cdot mp = y_{k-1} \cdot gp \cdot \eta' + w_k \cdot v \cdot \alpha$ for $k = 3, 4, \ldots$

(i) $w_1 \cdot v = I \cdot gp \cdot \eta' + x_2 \cdot mp$

(j) $w_k \cdot v = z_{k-1} \cdot gp \cdot \eta' + x_{k+1} \cdot mp$ for $k = 2, 3, \ldots$

(k) $u_0 \cdot rp = u_1 \cdot rp$

(l) $u_k \cdot rp = z_k \cdot v \cdot \alpha' + u_{k+1} \cdot rp$ for $k = 1, 2, \ldots$

(m) $q_k \cdot rp = w_k \cdot v \cdot \alpha' + q_{k+1} \cdot rp$ for $k = 1, 2, \ldots$

(n) $I + J + u_0 + \sum_{k=1}^{\infty} (z_k + y_k + x_k + w_k + u_k + q_k) = 1$
(normalizing equation) (4)

Solving this system of equations, we can obtain the quilibrium probabilities. This system can be solved in closed form, but the solution pro-

cedure is not described here. Since the system throughput THR_p was defined as the number of acceptances made per unit time, THR_p and T_p can be obtained by

$$THR_p = v \cdot \alpha \cdot \left(\sum_{k=1}^{\infty} z_k + \sum_{k=1}^{\infty} w_k \right)$$

$$T_p = 1/\left(v \cdot \alpha \cdot \left(\sum_{k=1}^{\infty} z_k + \sum_{k=1}^{\infty} w_k \right) \right). \qquad (5)$$

Another measure of interest is the expected queue-length E(QL).

$$E(QL) = J + \sum_{k=1}^{\infty} (k \cdot (z_k + y_k + x_k + w_k + u_k + q_k))$$

$$\text{where } y_1 = x_1 = 0 . \qquad (6)$$

Figure 6 depicts the expected queue-length E(QL) for various values of $\alpha, \eta, t_v/t_{op}, t_{mp}/t_{op}, t_{rp}/t_{mp}$. In examining Figures 6 and 7 we are mostly interested in the cases where α is greater than 0.9. Since fault-tolerant programs dealt with here are supposed to have undergone a testing phase before being put into operation, one or more erroneous object blocks out of ten seems highly improbable. On the other hand, η is application-dependent and may not be very close to 1. For example, $\eta = 0.999$ implies that only one among 1000 execution images generated is an S-image. In this evaluation, η is set mostly within the range of 0.9 - 0.95 and the most frequently used values are 0.9 for η and 0.95 for α. The following practical constraints were also adopted.

$$t_v < t_{op}$$

$$t_{mp} < t_{op}$$

$$1 < t_{rp}/t_{mp} \leq 1.5 . \qquad (7)$$

As expected, E(QL) becomes larger as α or η increases. Furthermore, comparison of curve 3 in Figure 6a (which is a result of changing α when $\eta = 0.95$) with curve 2' (a result of changing η when $\alpha = 0.95$) indicates that E(QL) is more sensitive to the change of η than to the change of α. This is also shown by a comparison of curve 2 (a result of changing α when $\eta = 0.9$) with curve 1' (a result of changing η when $\alpha = 0.9$). Figure 6b shows that E(QL) increases as mean validation time t_v or mean merging time t_{mp} increases. When $t_v + t_{mp} < t_{op}$, E(QL) is generally smaller than 5. The data obtained but not plotted in Figure 6 indicated that mean recovery time t_{rp} affects E(QL) to a negligible extent. This is because (1) when α is large, the system rarely enters a recovery state and (2) when α is small, the system rarely enters a state where the queue-length is large.

3.3 Performance Comparison Between the Sequential System and the Parallel System

A simple way of assessing the performance of the parallel system is to compare the throughput THR_p with the throughput THR_s of the sequential system. THR_p/THR_s is then the throughput ratio and is a function of $\alpha, \eta, t_v/t_{op}, t_{mp}/t_{op}, t_{rp}/t_{mp}, t_{os}/t_{op}$, and t_{rp}/t_{rs}. Here t_{os}/t_{op} represents the object block execution time ratio while t_{rp}/t_{rs} represents the recovery time ratio. These parameters are within the following ranges (cf. Section 2 or [5] for more details).

$$1 < t_{os}/t_{op} \ll 2$$

$$1 < t_{rp}/t_{rs} < 1.5 . \qquad (8)$$

Figure 7 depicts the throughput ratio for various values of parameters subject to the constraints in Eqs. (7) and (8). First, Figure 7a discloses that variation of recovery time ratio t_{rp}/t_{rs} within a practical range has little effect on the throughput ratio. This is again because (1) when α is large, the system rarely gets into a recovery state, and (2) when α is small, E(QL) becomes small and thus a recovery involves mostly a small number of execution images. Figure 7b indicates that the throughput ratio is not much affected by the change of t_{rp}/t_{mp} for α within a practical range, while it is significantly affected by object block execution time ratio t_{os}/t_{op}. Object block execution time ratio t_{os}/t_{op}, recovery time ratio t_{rp}/t_{rs} and t_{rp}/t_{mp} are machine characteristics while other parameters represent program characteristics.

Figure 7c shows that the throughput ratio decreases as merging time t_{mp} (more precisely t_{mp}/t_{op}) increases. The obvious reason is because under assumption G.1 merging is involved only in parallel execution. It also shows that increase of t_v causes a throughput ratio increase approximately until $t_v + t_{mp}$ surpasses t_{op} but further increase of t_v does not change (actually slightly decreases) the throughput ratio. This can be explained as follows. As $t_v + t_{mp}$ becomes larger than t_{op}, E(QL) becomes large and thus, each time a synchronizing execution image is generated, the queue contains a large number of execution images. The validation and merging of these are not overlapped with object block execution. Figure 7d confirms the expectation that as η increases, the throughput ratio increases.

In summary, (1) for a practical α, the performance improvement by parallel execution is most sensitive to object block execution time ratio t_{os}/t_{op} and t_{mp}/t_{op}, less sensitive to t_v/t_{op} and the least sensitive to t_{rp}/t_{mp} and recovery time ratio t_{rp}/t_{rs}, and (2) the throughput ratio ranged over 1.02 - 1.65 (or 2 - 65% gain) for $\alpha = 0.95$ and for the values of other parameters plotted in Figure 7.

Figure 7a also displays the existence of α_L (defined in Section 2 as the lower bound of α to make the performance of the parallel system superior to that of the sequential system). The data obtained but not fully plotted in Figure 7 showed that in all the cases depicted in Figure 7, α_L did not exceed 0.87 and rarely went above 0.6. It can conservatively be said that the practical range of α is far above α_L.

4. Summary

The analysis made in this paper confirmed that parallel execution can reduce the execution time increase inherent in fault-tolerant programs. The analysis demonstrated largely two points. First, under all practical circumstances the parallel system showed good performance. The performance was particularly good when α was above 0.9 or 0.95. It is believed that α would always be in such a range for programs which have undergone a reasonable degree of testing before being put into operation. Second, it showed how the effectiveness of parallel execution was affected by various program characteristics. Although no real statistics on various program characteristics are available, it is believed that our examination covered a broad range of reasonable values for each parameter. Availability of a parallel system may influence the program characteristics to some extent.

In short, the parallel execution approach allows the incorporation of extensive validation and recovery facilities without associated expensive execution time overhead. The price paid is the increased hardware requirement.

Acknowledgements

The authors would like to thank Drs. David L. Russell, C. V. Ramamoorthy, and L. R. Welch for helpful discussions. The authors also wish to acknowledge the help of Messrs. M. Naghibzadeh, M. Olumi, and B. Shah.

References

[1] Anderson, T. and Kerr, R., "Recovery Blocks in Action: A System Supporting High Reliability," Proc. 2nd Int'l Conf. on Software Engineering, 1976, pp. 447-457.

[2] Chandy, K.M., "A Survey of Analytic Models of Rollback and Recovery Strategies," Computer, May 1975, pp. 40-48.

[3] Horning, J.J., Lauer, H.C., Melliar-Smith, P.M. and Randell, B., "A Program Structure for Error Detection and Recovery," Lecture Notes in Computer Science, Vol. 16, Springer-Verlag, 1974, pp. 171-187.

[4] Kim, K.H. and Ramamoorthy, C.V., "Failure-tolerant Parallel Programming and Its Supporting System Architecture," Proc. AFIPS Nat'l Comp. Conf., 1976, pp. 413-423.

[5] Kim, K.H., "A Parallel System Processing Fault-Tolerant Programs - I. System Architecture," submitted for publication. (Also Tech. Memo. PETP-2, Electronics Science Laboratory, University of Southern California.)

[6] Kleinrock, L., Queueing Systems Volume 1: Theory, Wiley-Interscience, 1975.

[7] Computing Surveys, Vol. 8, No. 4, December 1976 (Special Issue on Fault-Tolerant Software edited by R. T. Yeh).

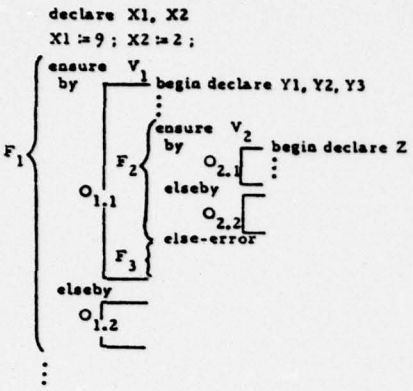


Figure 1a. A block-structured fault-tolerant program.

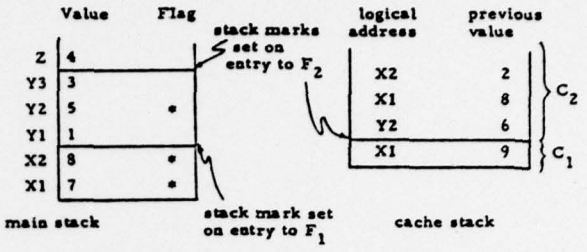


Figure 1b. Recovery cache during execution of 1a.

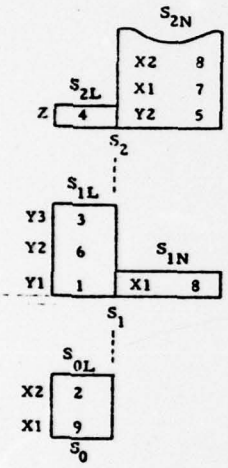


Figure 1c. VR-store during execution of 1a.

F_k: Fault-tolerant block k

Figure 2. Execution of a fault-tolerant program of the type assumed.

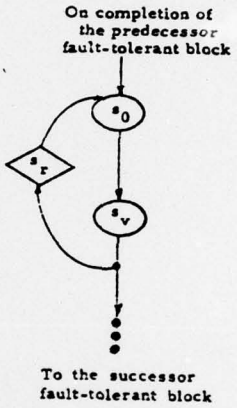


Figure 3a. Behavior of the sequential system during execution.

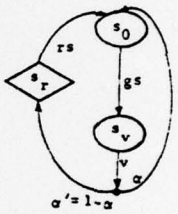


Figure 3b. Model S.

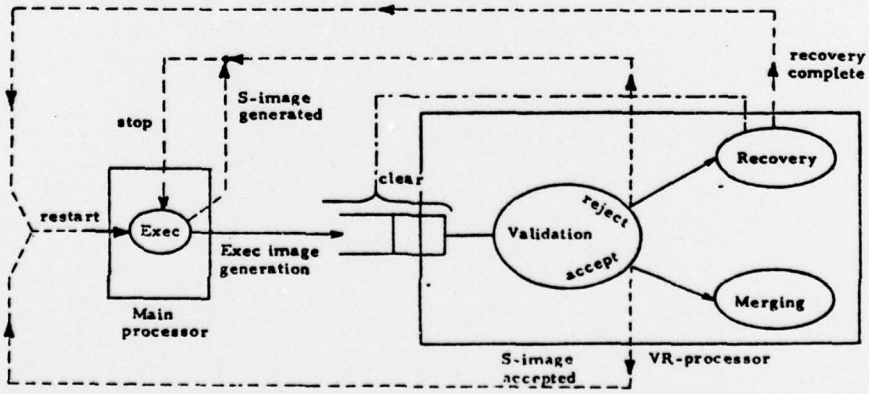


Figure 4. Parallel system modeled in a queueing system.

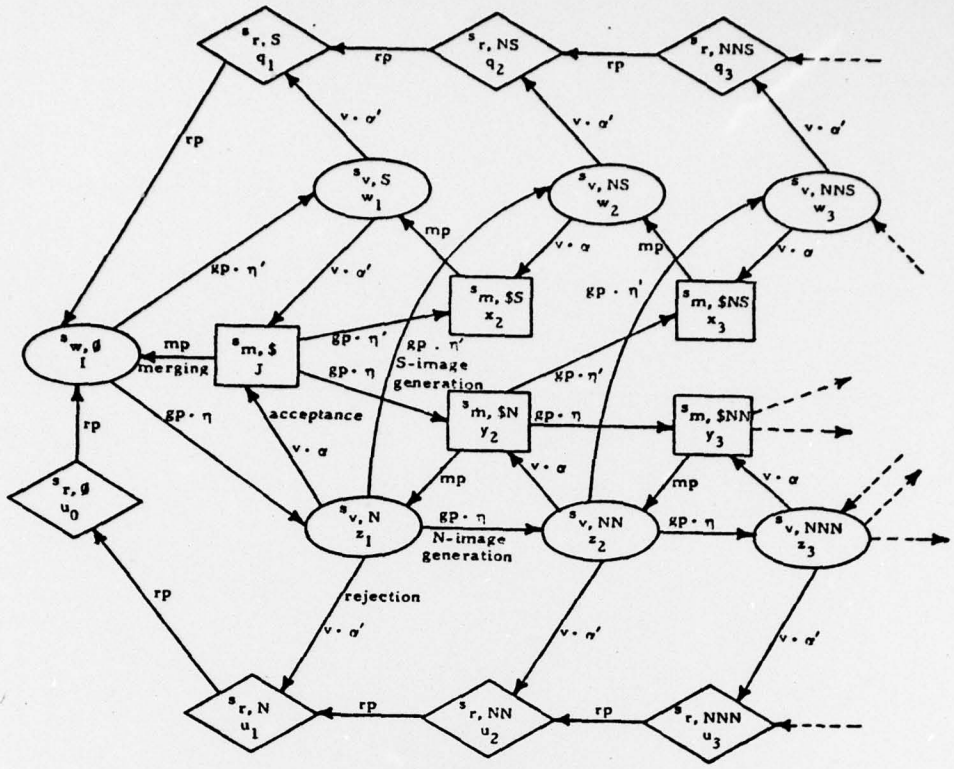


Figure 5. Model P

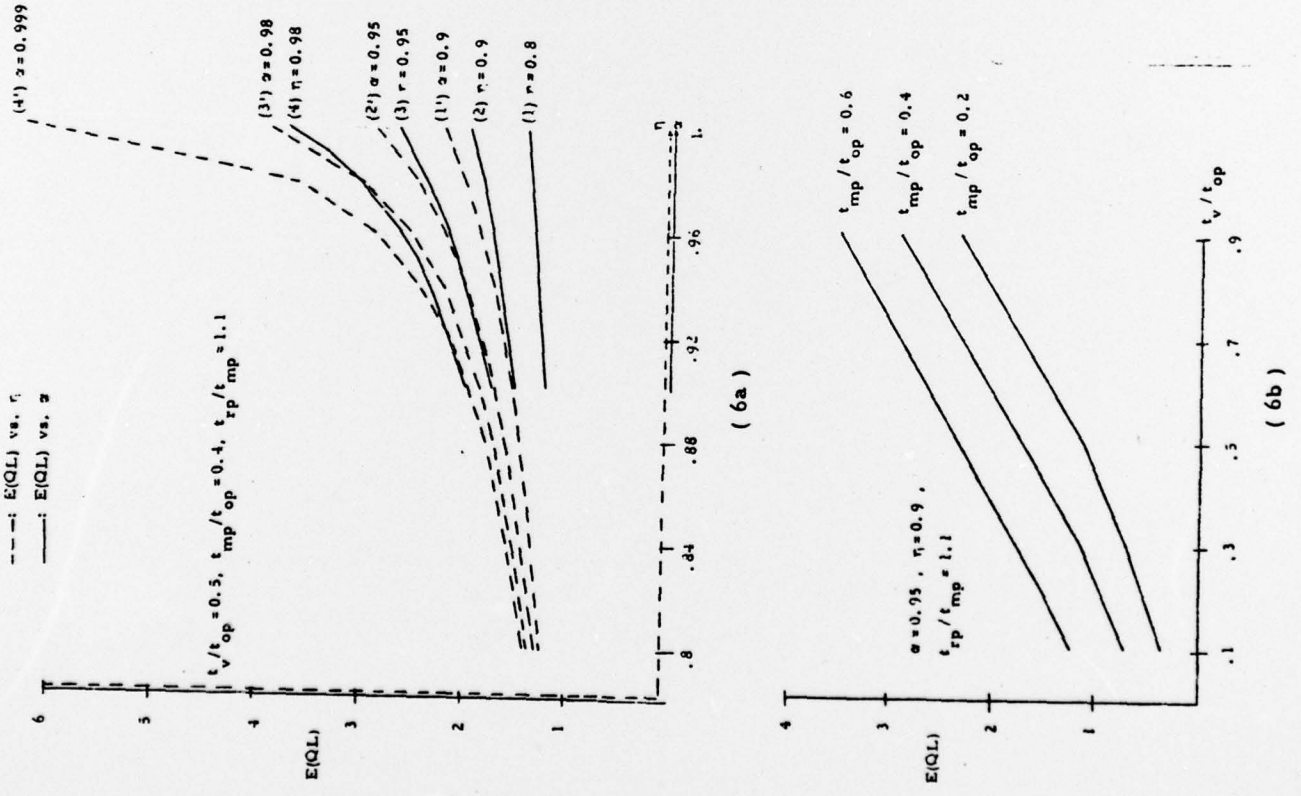
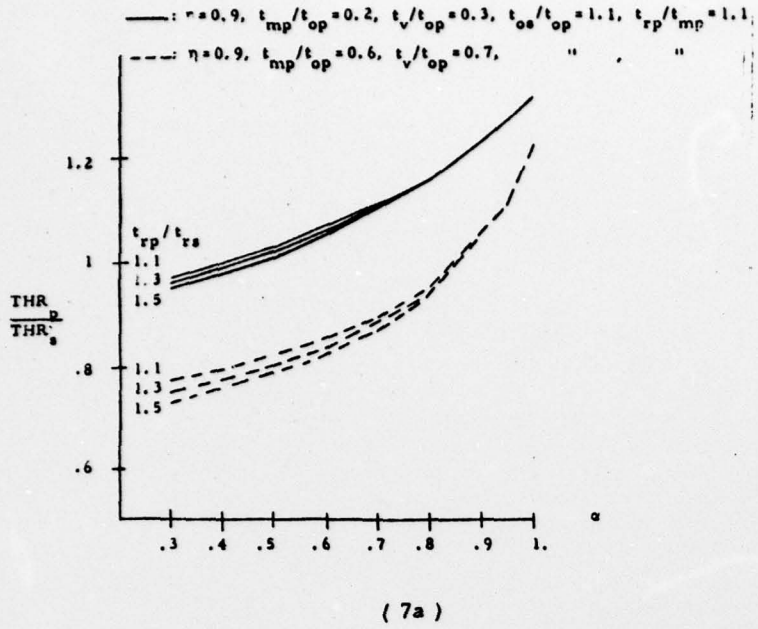
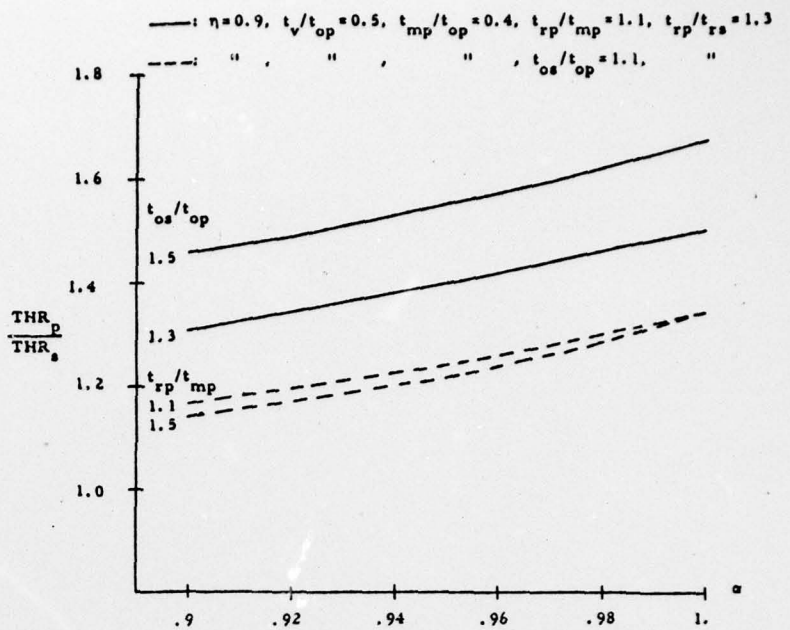


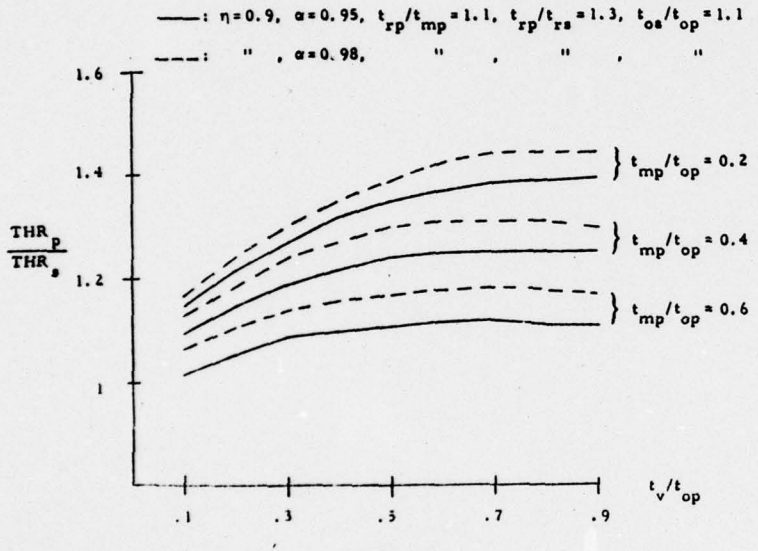
Figure 6. Expected queue-length E(QL).



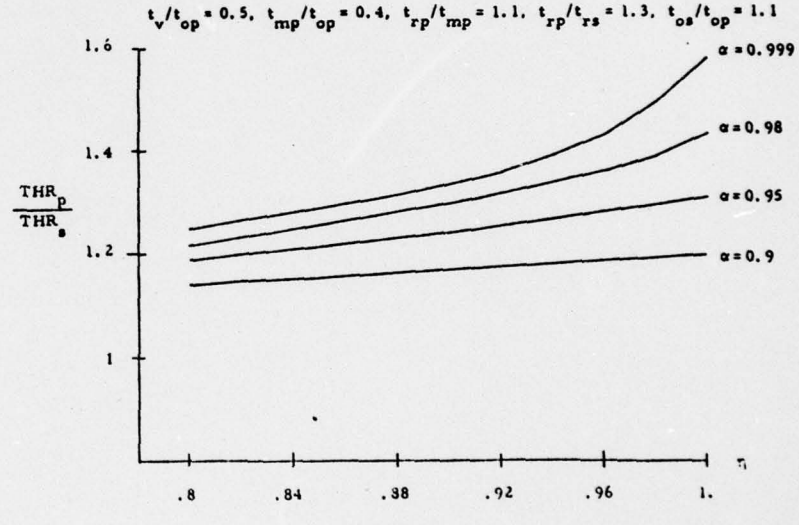
(7a)



(7b)



(7c)



(7d)

Figure 7. Throughput ratio THR_p / THR_s.