# NAVAL POSTGRADUATE SCHOOL

## Monterey, California




# THESIS

A FAULT-TOLERANT SOFTWARE ALGORITHM
FOR A NETWORK OF TRANSPUTERS

by

William Fred Benage, Jr.

June 1989

Thesis Advisor: Uno R. Kodres

Unclassified

Security Classification of this page

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a Report Security Classification: Unclassified | 1b Restrictive Markings |
| 2a Security Classification Authority | 3 Distribution Availability of Report: Approved for public release; distribution is unlimited. |
| 2b Declassification/Downgrading Schedule | |
| 4 Performing Organization Report Number(s) | 5 Monitoring Organization Report Number(s) |
| 6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (If Applicable): 52 | 7a Name of Monitoring Organization: Naval Postgraduate School |
| 6c Address (city, state, and ZIP code): Monterey, CA 93943-5000 | 7b Address (city, state, and ZIP code): Monterey, CA 93943-5000 |
| 8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable) | 9 Procurement Instrument Identification Number |
| 8c Address (city, state, and ZIP code) | 10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No |

11 Title (Include Security Classification): A Fault-Tolerant Software Algorithm for a Network of Transputers

12 Personal Author(s): William F. Benage, Jr.

| 13a Type of Report | 13b Time Covered | 14 Date of Report (year, month,day) | 15 Page Count |
|---|---|---|---|
| Master's Thesis | From To | June 1989 | 60 |

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 17 Cosati Codes | | | 18 Subject Terms (continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| Field | Group | Subgroup | Fault-Tolerance, Fault-Tolerant Computing, Transputers, Multiprocessors |
| | | | |
| | | | |

19 Abstract (continue on reverse if necessary and identify by block number)

This thesis presents a software algorithm that resends work packages to processors when one or more of the worker processors fails or when the link with one or more processors fails. There are two resend criteria used in this algorithm: "resend at end of initial assignment" and "resend at time out." The work, divided into several packages in order to run on several processors in parallel, will be completed as long as at least one worker processor remains working and communicating with the main processor. This algorithm could add some fault-tolerance to computer processing equipment in embedded systems.

| 20 Distribution/Availability of Abstract | 21 Abstract Security Classification | |
|---|---|---|
| [X] unclassified/unlimited [ ] same as report [ ] DTIC users | Unclassified | |
| 22a Name of Responsible Individual: Uno R. Kodres | 22b Telephone (Include Area code): (408) 646 2197 | 22c Office Symbol: 52Kr |

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

security classification of this page: Unclassified

Approved for public release; distribution is unlimited.

**A Fault-Tolerant Software Algorithm for a Network of Transputers**

by

William F. Benage, Jr.
Lieutenant, United States Navy
B.S., California State University, Long Beach, 1976
M.B.A., University of Southern California, 1977

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

Author: William F. Benage, Jr.
William F. Benage, Jr.

Approved by: Uno R. Kodres
Uno R. Kodres, Thesis Advisor

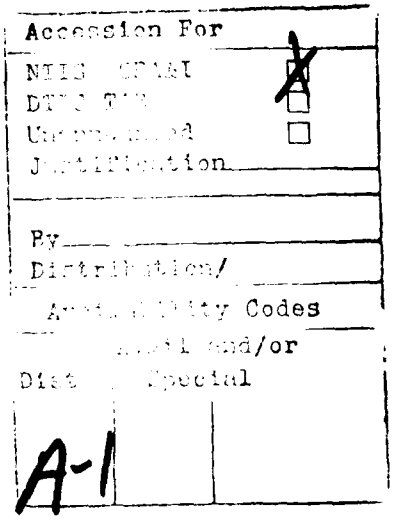
John Yurchak, Second Reader

Robert B. McGhee, Chairman
Department of Computer Science

K.T. Marshall
Kneale T. Marshall,
Dean of Information and Policy Sciences

## ABSTRACT

This thesis presents a software algorithm that resends work packages to processors when one or more of the worker processors fails or when the link with one or more processors fails. There are two resend criteria used in this algorithm: "resend at end of initial assignment" and "resend at time out." The work, divided into several packages in order to run on several processors in parallel, will be completed as long as at least one worker processor remains working and communicating with the main processor. This algorithm could add some fault-tolerance to computer processing equipment in embedded systems.



## THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made within the time available to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempting to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis are listed below the firm holding the trademark:

INMOS Limited, Bristol, United Kingdom:
- Transputer
- OCCAM
- T414
- T800
- Transputer Development System (TDS)

3L Limited, Livingston, United Kingdom:
- Parallel C
- Parallel Fortran

United States Government, Ada Joint Program Office, Washington, DC:
- Ada

ParaSoft Corporation, Mission Viejo, CA:
- Express

Microsoft Corporation, Redmond, WA:
- MS

Bell Laboratories, New York, NY:
- UNIX

Digital Equipment Corporation, Maynard, MA:
    VMS

Apple Computer, Cupertino, CA:
    Macintosh

## TABLE OF CONTENTS

## DEDICATION

I dedicate this thesis to my wife, Elizabeth, who gave me encouragement and patiently read this thesis also to my stepchidren, Robert and Erin, who were patient and understanding.

# I. INTRODUCTION

## A. BACKGROUND

### 1. Shipboard Systems

In today's combat systems, the ability to continue to operate with hardware failures is becoming more and more important. Today's shipboard computer systems require at best an automatic reload of the surviving system when a hardware failure occurs; at worst the system is completely inoperable. Some systems require a manual reload of the surviving system. All of this affects the ability to "fight hurt." A computer system that would automatically stop using processors that are not processing (for whatever reason), but still continue to process all needed functions and operations would greatly increase the ability of a ship to "fight hurt."

Current U.S. Navy systems using multiprocessors include: Aegis and all shipboard NTDS systems. In the Aegis system, there are four processors in the computer system. When one of these fails, the system automatically reloads the remaining three processors with software that has a reduced capability (primarily due to reduced available memory). This reload occurs in a very short time and thus the most important functions are restored with little user inconvenience. The major draw back is that it depends on the reload system to

function properly. Thus some ability of the computer system to "fight hurt" is there if the reload capability is still functioning when a processor quits.

Most shipboard NTDS systems have even less capability in this area. Most have three processors. When one of these quits, the remaining two must be manually reloaded with a reduced capable system. The time required for this manual reload is much longer than for the automatic reload used in the Aegis system. Thus the users are unable to function during the reload. This means that any ability to "fight hurt", where these NTDS systems are concerned, is dependent on the manual reload time as well as the proper functioning of the reload system.

Many U.S. Navy systems use only one processor in their computer systems. These have no ability to continue functioning when the processor fails. The ability of a ship to "fight hurt" would be enhanced if these systems had the ability to function when a processor quits.

A link failure between processors or between systems, in any of the current shipboard systems, means the loss of the functional use of at least one processor.

### 2. Fault-Tolerant Literature

The computer literature that discusses computer systems that can stop using processors refers to fault-tolerant computing or graceful degradation. Fault-tolerance deals with handling faults by restoring either full or reduced

capability. Graceful degradation deals generally with reduced capability. When a computer system has a graceful degradation capability, its:

- downtime for repair is short
- uninterrupted operation is longer
- unavailability periods are short
- overall processing power is not seriously affected by failures [Ref. 1].

In a multiprocessor system graceful degradation can be achieved by:

- Reconfiguring the system: switching out faulty hardware (processors, etc.) or software (modules, tasks, etc.) and switching in assumed good hardware or software.
- Masking the failed item: not using faulty hardware or software.

Reconfiguration can restore full or degraded capability. Masking will usually result in degraded capability in some way. [Ref. 2]

Reconfiguration of hardware can be done with a dedicated reconfiguration processor or each processor can preform part of the overall reconfiguration work. Another approach is to use a reconfiguring process running on each processor. This process runs in turn on each functioning processor using a "check list" to step through the recovery process. These approaches require normal processing to be stopped during the reconfiguring process.[Ref. 3]

The processor network used by the above approaches is not specified. The network may or may not have spare

processors that can be switched to when a processor fails. A matrix approach with one spare row and one spare column of processors gives a great deal of flexibility for replacing failed processor [Ref. 4].

## B. PROBLEM STATEMENT

Any computer system using more than one processor can have a fault-tolerant feature. To have a computer system continue to process when one or more of its processors stops (given some processors continue to process), is the problem this thesis looks at.

The objective of this thesis:

- While a Mandelbrot Program is running on a multiprocessor system, disconnect the link between some processors and have the system complete the Mandelbrot picture, even when only one processor capable of running.

## C. THESIS OVERVIEW

This thesis is presented in four Chapters and five Appendixes.

Chapter I was the introduction to the problem. Chapter II describes the system and the language. Chapter III addresses the method used in reaching the objective, the relative speeds of processing with different numbers of processing processors. Chapter IV states the conclusions and suggests further research.

# II. TRANSPUTERS: PROGRAMMING LANGUAGES AND CAPABILITIES

## A. PROGRAMMING LANGUAGES

### 1. OCCAM

OCCAM is a high level programming language that is designed to run concurrent processes on a network of transputers. There are two prime concepts in OCCAM, they are: concurrency and communication. These allow processes to run simultaneously and transfer information, via channels, from process to process. It is based on concepts founded by David May in Experimental Programming Language and by Tony Hoare in Communicating Sequential Processes. [Ref. 5]

It allows processes running on a transputer system to communicate only through channels. These channels are asynchronous, but require the send and receive processes to be ready to send and receive at the same time. This idea of being ready to send and receive simultaneously is known as rendezvous. (Ada also has a rendezvous feature.)

OCCAM has six kinds of constructions that are used to build a process from smaller processes (primitive or other). These constructions are:

- IF: This construction guards a number of processes by a boolean expression. It is similar to other languages' IF statement.

- CASE: This construction is used to select one of a number of options. It is similar to the Pascal CASE statement.

- WHILE: This construction is used for loops. Again it is similar to the Pascal WHILE statement.

- PAR: This construction is used to parallel processes. Processes under this construction are done at the same time.

- ALT: This construction is used to combine processes that are guarded. It selects one of the processes whose guard is in the true state.

This language allows the programmer to concentrate on a small, manageable set of processes which can then be connected with other sets of processes. In OCCAM a set of processes or a set of interconnected processes can be regarded as a single process. [Ref. 5]

The above features make OCCAM a powerful and versatile language. It hasn't gained wide acceptance thus far probably due to the limited use of multiprocessor (transputer) systems and due to the development of parallel versions of other widely used languages.

2. Parallel C

Parallel C is another high level programming language that was developed for use on a system of transputers [Ref. 6, pp. xi]. It is based on the abstract model that requires communication of sequential processes via communication channels or ports [Ref. 6, pp.32].

A program can consist of one or more tasks with all tasks executing simultaneously. Each task has its own area of

memory for instructions and data along with a vector of input ports and a vector of output ports. This allows a task to be on a transputer by itself or with other tasks. (Each task is compiled and linked separately.) The tasks communicate via the ports which make each task able to function as a software black box. [Ref. 6, pp.32-33]

This allows processes that can be done in parallel to be written in different tasks and run on separate transputers simultaneously. This means that if a problem can be done in many pieces simultaneously, the tasks to do these pieces can be placed on several transputers and executed in parallel. This reduces the overall computation time.

The software black box (tasks) and the hardware (transputers) are tied together with a program called the configurer. This program connects the ports of the various tasks and assigns each task to a transputer in a designated system. [Ref. 6, pp.35-36]

The Parallel C described above provides a convenient way of using transputers without the need to learn OCCAM and its associated editor and Transputer Development System.

Other Concurrent C languages have been developed for various multiprocessor systems [Ref. 7]. Unfortunately, since all concurrent or parallel C languages are a super set of C, the only commonality between these concurrent or parallel C languages is the underlying C language.

### 3. Parallel Fortran

Parallel Fortran, like Parallel C, was developed for use on a system of transputers [Ref. 8, pp. xviii]. There are some differences, however, between Parallel Fortran and Parallel C beyond the obvious differences between Fortran and C. Parallel Fortran has an ALT function that is similar in operation to the ALT in OCCAM [Ref. 8, pp.60]. Furthermore, Parallel Fortran has a set of Bit-Manipulation Functions [Ref. 8, pp.426]. It also has a set of ANSI Standard Intrinsic Functions for Fortran 77 that give Fortran its abilities for use in mathematic and scientific applications.

Parallel Fortran is used much like Parallel C in that it uses processes that communicate only over channels and the processes can be run on one or more transputers [Ref. 8, pp. 33-35]. It uses tasks in much the same way as Parallel C, and even its configurer is nearly identical to Parallel C's [Ref. 8, pp.33-40].

### 4. Ada

Ada is DOD's high level programming language that is required for all imbedded systems contracted for after June 1984 [Ref. 9]. When an Ada compiler becomes available for the Inmos transputer, research using the combination of Ada and the transputer can begin. With Ada on a transputer system, a true parallel processing use of the rendezvous feature can be incorporated in applications.

An Ada compiler for the Inmos transputer is under development by Alsys, Inc. [Ref. 10]. It is expected to be available in October 1989 [Ref. 11]. This will be the first implementation of Ada for a multiprocessor system that does not have a shared memory system.

## B. EXPRESS

Express is an operating system that runs under another operating system. This system allows a program to be parallelized and executed provided the proper compiler is on the host's operating system. Any compiler for use on the host operating system will work. The Express system is available for MS-DOS, UNIX, VMS and Macintosh operating systems. It provides facilities for:

- generating runtime parameters that allows programs to adapt at runtime to the transputer system environment
- automatic mapping of a large image into smaller images for individual processors
- interface of the I/O system with the various processors
- allowing global accumulation of data in the transputer system. [Ref. 12, pp.3-7]

## C. CAPABILITIES

The Inmos transputers are a family of microprocessors measuring about 30mm by 30mm by 4mm [Ref. 13, pp. 47]. The major components are: processor, four link engines, timer, 2K bytes RAM and external memory interface. (The T800 has

4K bytes RAM vice 2K bytes and has a floating point unit.) These components are connected by an internal bus. Both the T800 and the T414 are 32 bit devices. The T212 is a 16 bit device. [Ref. 14, pp. 18, 194] These are in essence a computer on a chip with limited memory. The on chip memory has a very short access time. The external memory interface can address up to 4G bytes of off chip memory allowing the transputer to access more memory than many large computer systems in use today.

Prior theses [Ref 14, 15] document a detailed description of the major transputer components and some detailed tests and analysis of transputer performance. Of these performance test results, the affects that communication links have on processor performance and visa versa are remarkable. The packet size effects the processor performance greatly. Continually communicating with small packets (less than 100 bytes per packet) reduces processor performance greatly. Conversely, using large packets (1000 to 100,000 bytes) only reduce the processor performance by 25% at most. This test was done while communicating on all four links as quickly as the link system will send/receive the packets. The affect that the processor has on communications link performance was not significant. [Ref. 15, pp.38-44] This seems to indicate that links have priority over the processor on the internal bus. This would keep processors from waiting on information to process.

## D. CONCURRENT FAULT-TOLERANT TRANSPUTER RESEARCH

A concurrent research project is in progress involving fault-tolerant transputer systems. This concurrent research involves the use of a transputer system whose links are connected through two crossbar switches. The links can be removed and/or inserted while the system is running. When a link is removed, the system looks for another link between the transputers that need to communicate and sets the crossbar switches to facilitate the needed communication. Thus, the fixed communication links are able to be replaced with dynamically assigned links for direct communications between transputers. These crossbar switches are controlled by a single transputer that takes communication requirements from each of the other transputers in the system. [Ref. 16]

# III. FAULT-TOLERANT SOFTWARE ALGORITHM

## A. BACKGROUND

### 1. Fault-Tolerant Literature Leading to the Algorithm

The fault-tolerance literature discusses many approaches. The masking technique mentioned earlier is user transparent and quick. The masking technique here implies static (no hardware switching) fault-recovery. However, a static technique can be employed while a dynamic technique is switching hardware or software. [Ref. 2]

### 2. The Processes Without the Algorithm

The Parallel C compiler contained a demonstration program (Mandelbrot) that simply sent packages of work out to worker transputers. the worker transputers sent back the assigned package results as they completed the work on each package, see Figure 1. The order of sending started with package number 00 and proceeded in order through 99 as shown in Figure 2. The order of receiving depends on the amount of work to complete the assigned package calculations. [Ref. 6]

## B. THE ALGORITHM

### 1. The Idea

While experimenting with the Parallel C compiler and the Mandelbrot demonstration program, two questions came to mind:

- what happens when communication is lost with one or more worker transputer
- can the remaining worker transputers pick up the work for those that have lost communication?

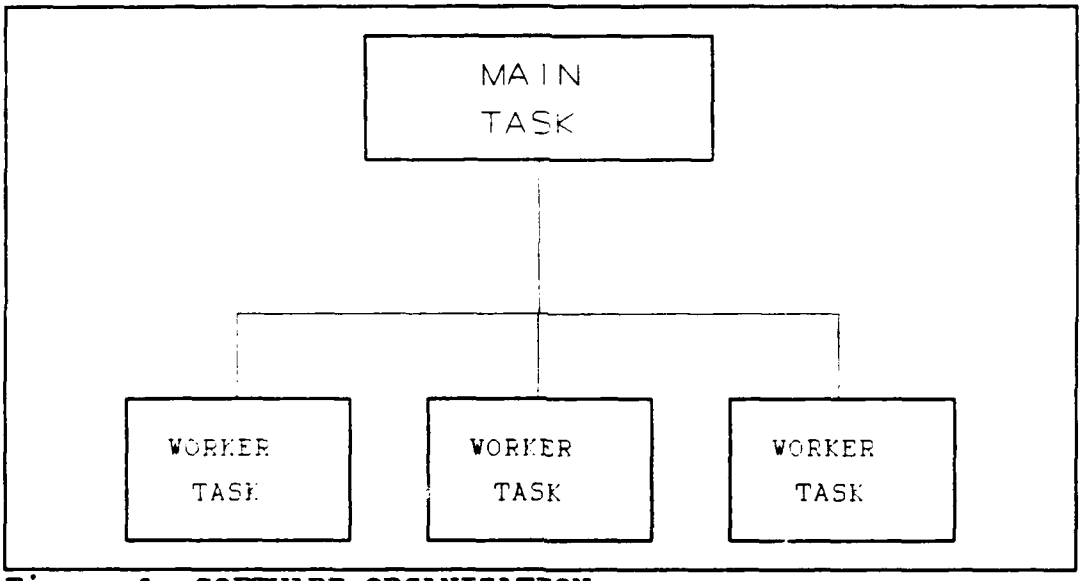



**Figure 1. SOFTWARE ORGANIZATION**

This is one method of masking as the article "Redundant Parts Keep Systems Running" [Ref. 2] discusses.

If the work can be done in spite of lost workers, imbedded systems could use this algorithm as a part of a fault-tolerant system.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| 01 | 11 | 21 | 31 | 41 | 51 | 61 | 71 | 81 | 91 |
| 02 | 12 | 22 | 32 | 42 | 52 | 62 | 72 | 82 | 92 |
| 03 | 13 | 23 | 33 | 43 | 53 | 63 | 73 | 83 | 93 |
| 04 | 14 | 24 | 34 | 44 | 54 | 64 | 74 | 84 | 94 |
| 05 | 15 | 25 | 35 | 45 | 55 | 65 | 75 | 85 | 95 |
| 06 | 16 | 26 | 36 | 46 | 56 | 66 | 76 | 86 | 96 |
| 07 | 17 | 27 | 37 | 47 | 57 | 67 | 77 | 87 | 97 |
| 08 | 18 | 28 | 38 | 48 | 58 | 68 | 78 | 88 | 98 |
| 09 | 19 | 29 | 39 | 49 | 59 | 69 | 79 | 89 | 99 |

**Figure 2. PATTERN DIVISION AND NUMBERING**

### 2. The Algorithm Details

When communication is lost with one or more worker transputers, the results from the work packages already assigned to them is lost. Each of the worker transputers will have up to three work packages assigned to it at any time. The remaining worker transputers continued to process their previously assigned work while receiving and processing new work. Thus, even with only one working transputer the problem will be completed; however, there is a considerable time penalty.

Thus, the main task was sending work packages and receiving results. All that needs to be done is to determine what results had not come back under some criteria. Two criteria are used in this thesis:

- resend at end of initial assignment: this criteria waits to resend any work package until all packages have been

sent once and then resends those packages whose results are not back yet

- resend at time out: this criteria resends a work package when a preset time has elapsed since it was first sent.

The code in Appendix A has the Mandelbrot data structure with annotated modifications for the "resend at end of initial assignment" criteria. The code in Appendix B uses the data structure in Appendix A to implement the "resend at end of initial assignment" criteria.

The "resend at time out" criteria data structure code is in Appendix C. It has additions to the code in Appendix A which allows the use of both criteria. Both criteria are implemented in the code in Appendix D.

## C. ALGORITHM TEST RESULTS

### 1. Original Code

The Mandelbrot code as supplied with the compiler, see Appendix E, was run on a system of 12 worker transputers (T414's). Figure 3 shows the system connections including the host PC and main transputer (a T414). The time to complete a Mandelbrot set averaged 108.6 seconds.

### 2. Adding the "resend at end of initial assignment" criteria

Next the Mandelbrot code was run with the "resend at end of initial assignment" criteria. Running the same Mandelbrot set as previously run on the system shown in Figure 3, the average time to complete the set was 109.6

seconds. Analysis of variance between the times with and without the "resend at end of initial assignment" criteria shows no significant difference at the .05 level of significance. Thus the new code doesn't significantly impact the computing time of this process.

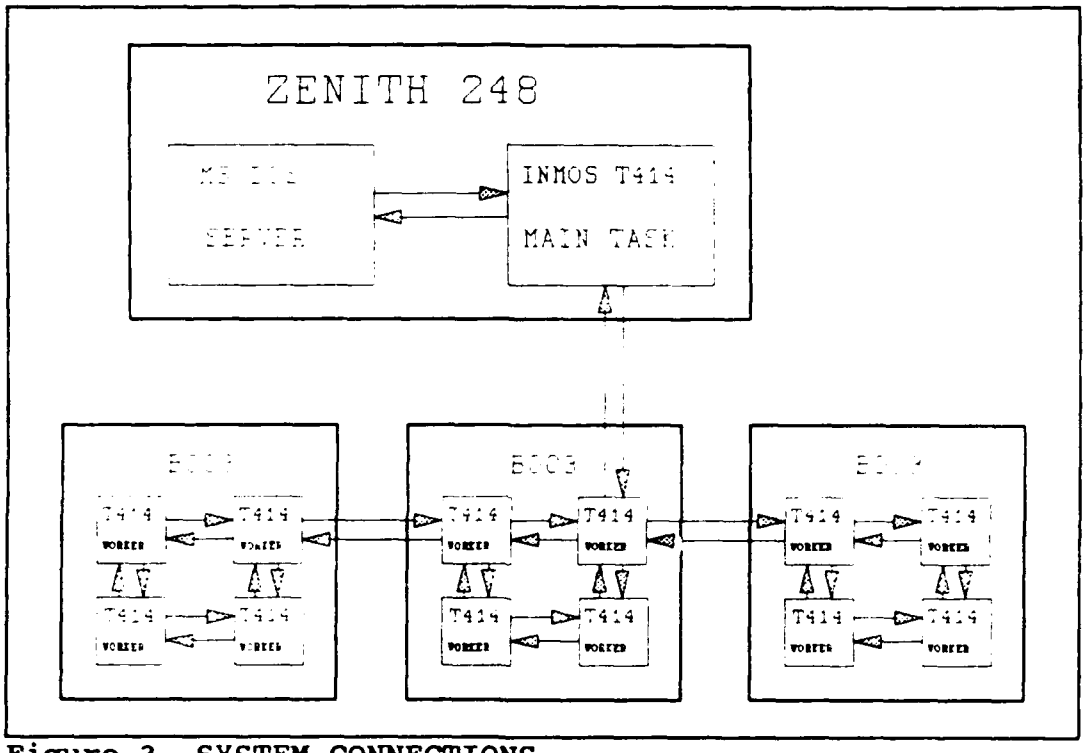



Figure 3. SYSTEM CONNECTIONS

The fault-tolerant feature of this code was next timed. The system started out with the connections shown in Figure 3. While the Mandelbrot set was running, the link to four of the worker transputers was disconnected as shown in Figure 4. Table I shows the average time to complete when the link was disconnected at times shown.

### 3. Adding the "resend at time out" criteria

Here, the "resend at time out" criteria was added to the original code with the "resend at end of initial assignment" criteria (running both criteria). The average time to complete the same Mandelbrot set as previously run, using the system shown in Figure 3, with different timer delays is as shown in Table II.

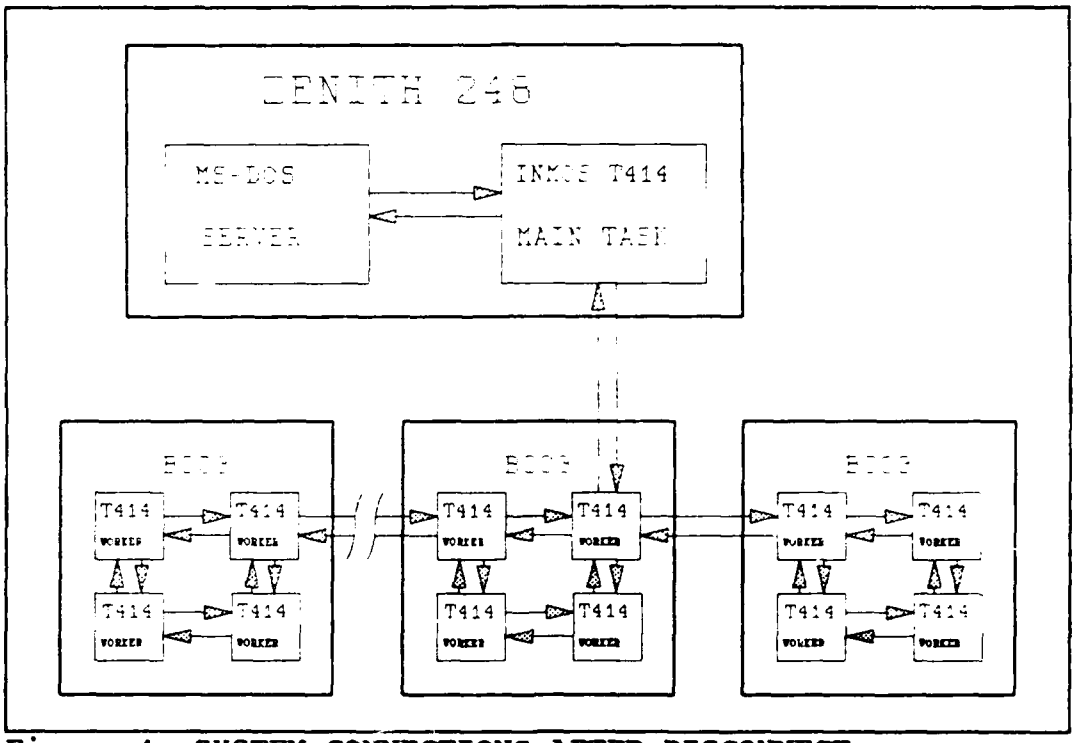



**Figure 4. SYSTEM CONNECTIONS AFTER DISCONNECT**

Analysis of variance between the original code time and the code with both criteria shows that all are significantly different at the .05 level of significance.

Since the 21 and 41 second delay times take longer to complete the problem than the original code, they are of less use than

**Table I. DISCONNECT TIME VS TIME TO COMPLETE**

| LINK DISCONNECT TIME TO IN SECONDS AFTER START SECONDS | TOTAL TIME TO COMPLETE IN |
|---|---|
| 20 | 142.0 |
| 40 | 143.0 |
| 60 | 128.0 |
| 80 | 128.0 |

Table II. DELAY TIME VS TIME TO COMPLETE

| DELAY TIME IN SECONDS | AVERAGE TIME TO COMPLETE IN SECONDS |
|---|---|
| 21 | 185.4 |
| 41 | 130.2 |
| 50 | 99.4 |

the 50 second delay. The 50 second delay took less time to complete than the original code. This would, with the 50 second delay, indicate better use of the processors for completing the work.

The fault-tolerant feature of this code was next timed. As with the test of the "resend at end of initial assignment"

The fault-tolerant feature of this code was next timed. As with the test of the "resend at end of initial assignment" criteria, the link to four of the worker transputers was disconnected approximately 20 seconds after the Mandelbrot set was started. The average times to complete are shown in Table III. The average time to complete for 40, 60 and 80 second link disconnect times are shown in Tables IV, V and VI respectively.

Table III. DELAY TIME VS TIME TO COMPLETE USING FAULT-TOLERANT FEATURE WITH 20 SECOND LINK DISCONNECT TIME

| DELAY TIME IN SECONDS | AVERAGE TIME TO COMPLETE IN SECONDS |
|---|---|
| 21 | 232.4 |
| 41 | 171.0 |
| 50 | 166.2 |

Table IV. DELAY TIME VS TIME TO COMPLETE USING FAULT-TOLERANT FEATURE WITH 40 SECOND LINK DISCONNECT TIME

| DELAY TIME IN SECONDS | AVERAGE TIME TO COMPLETE IN SECONDS |
|---|---|
| 21 | 301.6 |
| 41 | 222.2 |
| 50 | 208.0 |

Table V. DELAY TIME VS TIME TO COMPLETE
USING FAULT-TOLERANT FEATURE
WITH 60 SECOND LINK DISCONNECT TIME

| DELAY TIME IN SECONDS | AVERAGE TIME TO COMPLETE IN SECONDS |
|---|---|
| 21 | 293.6 |
| 41 | 203.6 |
| 50 | 164.4 |

Table VI. DELAY TIME VS TIME TO COMPLETE
USING FAULT-TOLERANT FEATURE
WITH 80 SECOND LINK DISCONNECT TIME

| DELAY TIME IN SECONDS | AVERAGE TIME TO COMPLETE IN SECONDS |
|---|---|
| 21 | 274.2 |
| 41 | 182.4 |
| 50 | 148.0 |

4. Results Summary

The "resend at end of initial assignment" criteria had no significant effect on the running of the program when all transputers remained connected. The program will complete the problem even though communication is lost with some of the worker transputers. When both criteria are used, the 21 and 41 second delays take longer than the original code; however, the 50 second delay takes less time than the original code. The fault-tolerant feature takes longer than the "resend at end of initial assignment" criteria. Further time tests are needed to find the best time to use.

# IV. CONCLUSIONS AND FURTHER RESEARCH PROPOSALS

## A. CONCLUSIONS

The INMOS transputers lend themselves readily to a fault-tolerant system. Here an application was made fault-tolerant by adding code to existing code. The two criteria for determining when an assumed fault has occurred can be used together. For practical use the individual application may need both or only one or the other. For example a system that never completes initial assignment, because it continually has work to assign, would not be able to use the "resend at end of initial assignment" criteria.

This algorithm could be useful in any embedded system. These systems have several functions or tasks (herein called worker) that are done repeatedly. By using more than one processor to do a task, many tasks can become fault-tolerant using this algorithm along with appropriate resend criteria. Thus, the loss of communication with a processor or processors (either processor failure or link failure or both) would not stop a task from functioning. By making tasks fault-tolerant, the entire system becomes more fault-tolerant.

Deadlines are very important in many embedded systems. Multiprocessors might be used in some areas in order to meet the required deadlines. If this is the case, adding additional

processors for fault-tolerance may be necessary to ensure that the deadlines can be met even with a failed processor.

## B. FURTHER RESEARCH

Research in the area of fault-tolerance on transputer systems is a broad area. The software approach taken in this thesis is one of several possible approaches. The algorithm presented herein is simple and appears not to increase problem completion time. Further research is needed in this area and on this algorithm to be able to apply it to a practical system.

Some recommended areas for further research are:

- develop an algorithm for determining the best time delay for use in the "resend at time out" criteria that can be used for any process using a similar system
- determine the affect of changing the time delay--used in the "resend at time out" criteria--to time from when a packet was last sent vice from when a packet was first sent
- develop an algorithm that will use reconnected transputers that were disconnected during processing, thus allowing recovery while the process is running (dynamic recovery)
- integrate this thesis with the one entitled Dynamic Reconfiguration and Link Fault-Tolerance in a Network of Transputers [Ref. 16], implementing dynamic recovery.

Continued research involving fault-tolerant systems of transputers should yield computer system that are especially useful in embedded systems.

## APPENDIX A

DATA STRUCTURE HEADER FILE FOR MANDELM2.C CODE

```
/***    MANDELTY.H[1]
 ***
 ***    Parallel Mandelbrots
 ***
 ***    These are the formats of the packets used to
 ***    communicate between the master task and the
 ***    computation tasks.
 ***
 ***    Rev 000   6-Dec-87  JF    Created
 ***
 ***/

typedef struct command_structure {
   float  x_coord, y_coord, gap;
   int    tlx, tly, brx, bry;
} COMMAND;

typedef struct results_structure {
   int    tlx, tly, brx, bry;
   char   counts[1008];
} RESULTS;

/* Modified by W. Benage  on 21 Mar 1989 */
/* This SAVE structure is used with MANDELM2.C code for */
/* fault tolerance. The fault tolerance for this code is */
/* for a link failure connecting work processors or */
/* connecting the work processors with the root processor. */
/* This alogrithm is not based on time but on what work */
/* has not returned complete.  */

typedef struct save_structure {
   int    tlx, tly, brx, bry, returned;
} SAVE;
```

[1]Reprinted with permission of 3L Ltd.

## APPENDIX B

## CODE FOR "RESEND AT END OF INITIAL ASSIGNMENT" CRITERIA

```
/***   MANDELM.C[2]
 ***
 ***   Copyright (c) 1988 3L Ltd
 ***
 ***   Example program: Mandelbrot set evaluation and
 ***   display. NB: This application requires a Colour
 ***   Graphics Adaptor.
 ***
 ***   The application
 ***   ---------------
 ***
 ***   The application consists of two tasks:
 ***
 ***   (1) MANDELM (this file). This is the master task, and
 ***   runs in the root transputer.
 ***   (2) MANDELW. This is the worker task, and runs in all
 ***   the other transputers of the net.
 ***
 ***   The flood configurer, FCONFIG, can be used to produce
 ***   an executable file which will automatically
 ***   distribute the worker tasks across an arbitrary
 ***   network and route work packets from the master to the
 ***   workers.
 ***
 ***   It is also possible to run the application in a
 ***   single transputer. This will work automatically if
 ***   the application is configured using FCONFIG.
 ***   Alternatively, a static single-transputer
 ***   configuration could be built by hand, using CONFIG. A
 ***   suitable configuration file may be found in
 ***   MANDEL.CFG.
 ***
 ***   As well as various routines from the Parallel C
 ***   run-time library, MANDELM must be linked with the CGA
 ***   primitives module, CGA.BIN. A file MANDELM.LNK is
 ***   supplied, which may be used to link MANDELM, like
 ***   this:
 ***
 ***      LINKT @MANDELM.LNK,MANDELM.B4
 ***
 ***   Functions of the tasks
```

[2]Reprinted with permission.

```
***   ----------------------
***
***   MANDELM is told by the user which part of the
***   Mandelbrot set to evaluate. It then breaks this up
***   into 100 packets, and sends them to the network of
***   MANDELW's. As the results from each return, they
***   are displayed on the PC's screen.
***
***   Internals of MANDELM
***   --------------------
***
***   The task contains three threads.
***
***   (1) The MAIN thread.
***   This runs in the function main(). It intialises the
***   other two threads and then goes into a loop, once
***   round for each Mandelbrot display. For each, it gets
***   instructions from the user, and then signals the SEND
***   thread to start work by using the
***   parameters_are_ready semaphore. It keeps track of
***   completed work by examining tally_done, which is
***   incremented by RECEIVE everytime a RESULTS packet is
***   displayed; whenever it notices that tally_done has
***   changed, it updates the PC's display; and when
***   tally_done reaches 100, MAIN knows that the display
***   is complete.
***
***   (2) The SEND thread.
***   This knows when to start work by examining the
***   parameters_are_ready semaphore. It then breaks the
***   job into 100 small jobs, places the details into a
***   COMMAND structure (defined in file MANDEL.H) and uses
***   the net_send function to send it off to the network
***   of MANDELW's. Notice the SEND does not specify WHICH
***   worker task is to do any particular job; this is
***   decided by the network of router tasks.
***
***   (3) The RECEIVE thread.
***   This simply waits till a packet arrives from the
***   network of MANDELW's and then displays it. Each
***   packet contains all the necessary information to
***   display it, so RECEIVE does not need to keep track of
***   which packet is which. Every time it does a display,
***   RECEIVE increments tally_done, so that MAIN can tell
***   when the whole display is complete.
***
***   Ver. 1.1  16-Dec-87   JF
***
***/

/* Modified for fault tolerance by W. Benage 21 Mar 1989 */
```

```
 /* All code not identified as fault-tolerant */
 /* modifications is 3L Ltd. original code. */

#include <stdio.h>
#include <dos.h>
#include <thread.h>
#include <sema.h>
#include <par.h>
#include <net.h>
#include <timer.h> /*Used in restart logic only */
#include "cga.h"
#include "mandel.h"  /* Modified for resend using
                        fault-tolerance. */
                     /* SAVE data type added. */

/* Interface to SEND thread */
static SEMA parameters_are_ready;

/* Fault-tolerance data structure. */
static SAVE s[10][10]; /* These integers in [] must match */
                          the divisors in #define         */
                          X_INCREMENT and #define         */
                          Y_INCREMENT lines.          */

/* Interface to RECEIVE thread */
static int tally_done;

/* Fault-tolerance variable */
static int resend;

/* Current Mandelbrot and display parameters */
static float x_coord, y_coord, gap;
static int thresh1, thresh2, thresh3;

/* Define the way the job is broken into packets */
#define X_INCREMENT ((CGA_LORES_XMAX+1)/10)
#define Y_INCREMENT ((CGA_YMAX+1)/10)
#define PACKETS 100

#define D 15625  /* Number of low pri ticks in 1 sec. */
                 /* Used in fault-tolerant restart only */

/*
 * This function is invoked by MAIN using thread_create to
 * create the SEND thread.
 *
 */

send ()
{
   int           i, j, x, y;   /* i, j varibles are added */
                                /* for fault-tolerance. */
```

```
COMMAND             c;

for (;;) {

 /* Wait here until MAIN signals it's okay to go ahead */
   sema_wait (&parameters_are_ready);

   /* Fill in the fixed parts of the command */
   c.x_coord = x_coord;
   c.y_coord = y_coord;
   c.gap = gap;

   /* Send off the packets to be done. Each includes the
      coordinates of the top-left and bottom-right
      corners of the area to do. This both tells the
      worker task what values to generate and identifies
      the RESULTS packet when it arrives in the RECEIVE
      thread (since there's no guarantee that the results
      will arrive in the same orderthe commands were sent
      out)
    */

   i = 0;
   for (x = 0; x < CGA_LORES_XMAX; x += X_INCREMENT) {
      c.tlx = x; c.brx = x + X_INCREMENT - 1;
      j = 0;
      for (y = 0; y <= CGA_YMAX; y += Y_INCREMENT) {
         c.tly = y; c.bry = y + Y_INCREMENT - 1;
         /* Send off the next packet */
         net_send (sizeof(COMMAND), &c, 1);
         /* Save coordinates of next packet */
         s[i][j].tlx = c.tlx;  s[i][j].brx = c.brx;
         s[i][j].tly = c.tly;  s[i][j].bry = c.bry;
         j += 1;
      }
      i += 1;
   }
   /* resend any packet not yet received back */
   resend = 0;
   while (tally_done < PACKETS) {
      for (i = 0; i < 10; i += 1) {
         for (j = 0; j < 10; j += 1) {
            if (s[i][j].returned == 0) {
               c.tlx = s[i][j].tlx; c.tly = s[i][j].tly;
               c.brx = s[i][j].brx; c.bry = s[i][j].bry;
               resend += 1;
               net_send (sizeof(COMMAND), &c, 1);
            }
         }
      }
   }
```

```
    }
}


/*
 * This function is invoked by MAIN using thread_create to
 * create the RECEIVE thread.
 *
 */

receive ()
{
    RESULTS     r;
    int         len, ready, x, y, i, j, n, colour, rtn, count;
              /* j, rtn, count are added for fault-tolerance */

    count = 0;
    for (;;) {

       /* Thread will wait here till a packet arrives */
       len=net_receive (&r, &ready);

       count += 1;
       if (tally_done < PACKETS) {

 /* Lets the fault-tolerant SAVE system know when a packet
    has been received. The i is also used here for fault-
    tolerance. */

          rtn = 0;
          for (i = 0; i < 10; i += 1) {
             for (j = 0; j < 10; j += 1) {
                if (s[i][j].tlx == r.tlx && s[i][j].tly ==
                    r.tly && s[i][j].brx == r.brx &&
                    s[i][j].bry == r.bry)
                   s[i][j].returned = 1;
                if (s[i][j].returned == 1)
                   rtn += 1;
             }
          }

          i = 0;

      /* The results packet includes the coordinates of the
         top-left and bottom-right corners of the data, so we
         know where to display it.
      */

          for (y=r.tly; y<=r.bry; y++) {
             for (x=r.tlx; x<=r.brx; x++) {
                n = r.counts[i++];
             /* Received 0 means 1, etc. */
```

```
                n += 1;
            /* Decide on the colour <- thresholds, and display */
                colour = (n>=thresh1) + (n>=thresh2) +
                         (n>=thresh3);
                cga_lores_plot (x, y, colour);
              }
            }

         /* Update the tally of packets displayed. */
            tally_done = rtn;
         }
/* The fault-tolerant system accounts for all packets
   sent.*/
         else {
            tally_done = count;
         /* Reset count when all packets are returned. */
         if ((count - PACKETS) == resend)
               count = 0;
         }
      }
}


/*
 * The MAIN thread runs here
 *
 */

main ()
{

   float range;
   int previous_tally, i, j;
                  /* i, j are added for fault-tolerance. */

/* Make sure we have text mode (and clear screen), then sign
   on */

   video_mode (MONO_80COL_TEXT_MODE);
   printf ("\nCopyright (c) 1988 3L Ltd\n\n");
   printf ("Example program: Mandelbrot set evaluation and
display\n");
   printf ("NB: This program requires a Colour Graphics
Adaptor\n\n");

 /* Initialise this SEMA to 0 BEFORE we start the SEND
    thread. This means it will wait until we tell it it's
    safe to go ahead  */

   sema_init (&parameters_are_ready, 0);
   /* Now start the other two threads */
   thread_create (send, 20000, 2,0,0);
```

```
thread_create (receive, 20000, 2,0,0);

for (;;) {

   /* Initialize s matrix for detecting packets not */
   returned for fault-tolerance. */
   for (i = 0; i < 10; i += 1) {
      for (j = 0; j < 10; j += 1) {
         s[i][j].returned = 0;
      }
   }

   /* This will ensure that no other threads are using
      the C run-time library (in fact, in this case they
      won't be, but I have done it here as an example...)
   */
   sema_wait (&par_sema);
   printf ("\nInput X coordinate: ");
   scanf ("%f", &x_coord);
   printf ("Input Y coordinate: ");
   scanf ("%f", &y_coord);
   printf ("Input Y range:      ");
   scanf ("%f", &range);
   gap = range / (float)(CGA_YMAX+1);
   y_coord = y_coord + range;

   printf ("Threshold 1: "); scanf ("%d", &thresh1);
   printf ("Threshold 2: "); scanf ("%d", &thresh2);
   printf ("Threshold 3: "); scanf ("%d", &thresh3);
   getchar ();  /* Consume the final NL */

   /* We have finished with the C RTL - release it */
   sema_signal (&par_sema);

   /* Into graphics (CGA low resolution) mode */
   video_mode (CGA_LORES_GRAPHICS_MODE);

   /* Before we set SEND going, reset the count of
      finished packets to zero - RECEIVE will count it
      back up
    */
   tally_done = 0;
   /* All ready - set it going! */
   sema_signal (&parameters_are_ready);

   previous_tally = 0;

   /* Until all the packets have been done, just keep
      updating the display when necessary
   */
   while (tally_done < PACKETS) {
      while (tally_done==previous_tally) {
```

```
            /* Wait here till something happens. Use
               thread_deschedule to save CPU time  */
            thread_deschedule ();
         }
      /* Send the picture up to the PC's display memory */
         cga_update ();
         previous_tally = tally_done;
      }

      /* This ensures that the PC display is up-to date. */
      cga_update ();

      /* Once again, wait for the RTL to be free; then beep
         and wait till the user strikes any key  */
      sema_wait(&par_sema);
      putchar ('\007');
      getchar ();
      sema_signal(&par_sema);

      /* Clear the screen and set text mode again */
      video_mode (MONO_80COL_TEXT_MODE);

   /* Print the number of packets resent to monitor fault-
      tolerant system. */
      printf ("Resend = ");
      printf ("%d",resend);
      printf ("\n");

      /* Wait for all resent packets to return or */
      /* if failure occured during processing, */
      /* wait 2 min.   */
      i = 0;
      while ((tally_done < (PACKETS + resend)) && (i < 120))
{
         timer_delay (D);
         i += 1;
         thread_deschedule ();
      }
   }

}
```

## APPENDIX C

### DATA STRUCTURE HEADER FILE FOR MANDELM3.C CODE

```
/***    MANDELTY.H[3]
 ***
 ***    Parallel Mandelbrots
 ***
 ***    These are the formats of the packets used to
 ***    communicate between the master task and the
 ***    computation tasks.
 ***
 ***    Rev 000   6-Dec-87  JF    Created
 ***
 ***/

typedef struct command_structure {
   float  x_coord, y_coord, gap;
   int    tlx, tly, brx, bry;
} COMMAND;

typedef struct results_structure {
   int    tlx, tly, brx, bry;
   char   counts[1008];
} RESULTS;

/* Modified by W. Benage  on 29 Mar 1989 */
/* This SAVE structure is used with MANDELM3.C code as */
/* modified for fault tolerance. The fault tolerance for */
/* this code is for a link failure connecting work */
/* processors  or connecting the work processors with the */
/* root processor. This alogrithm is based on time and on */
/* what work has not returned complete. */

typedef struct save_structure {
   int    tlx, tly, brx, bry, returned, time;
} SAVE;
```

[3]Reprinted with permission.

## APPENDIX D

## CODE FOR "RESEND AT TIME OUT" AND "RESEND AT END OF INITIAL ASSIGNMENT CRITERIA

```
/***   MANDELM.C[4]
 ***
 ***   Copyright (c) 1988 3L Ltd
 ***
 ***   Example program: Mandelbrot set evaluation and display.
 ***   NB: This application requires a Colour Graphics
 ***   Adaptor.
 ***
 ***   The application
 ***   ---------------
 ***
 ***   The application consists of two tasks:
 ***
 ***   (1) MANDELM (this file).   This is the master task,
 ***   and runs in the root transputer.
 ***   (2) MANDELW.   This is the worker task, and runs in
 ***   all the other transputers of the net.
 ***
 ***   The flood configurer, FCONFIG, can be used to produce
 ***   an executable file which will automatically distribute
 ***   the worker tasks across an arbitrary network and route
 ***   work packets from the master to the workers.
 ***
 ***   It is also possible to run the application in a single
 ***   transputer. This will work automatically if the
 ***   application is configured using FCONFIG. Alternatively,
 ***   a static single-transputer configuration could be built
 ***   by hand, using CONFIG. A suitable configuration file
 ***   may be found in MANDEL.CFG.
 ***
 ***   As well as various routines from the Parallel C
 ***   run-time library, MANDELM must be linked with the CGA
 ***   primitives module, CGA.BIN. A file MANDELM.LNK is
 ***   supplied, which may be used to link MANDELM, like this:
 ***
 ***      LINKT @MANDELM.LNK,MANDELM.B4
 ***
 ***   Functions of the tasks
 ***   ----------------------
 ***
```

[4]Reprinted with permission.

```
***   MANDELM is told by the user which part of the
***   Mandelbrot set to evaluate. It then breaks this up into
***   100 packets, and sends them to the network of
***   MANDELW's. As the results from each return, they
***   are displayed on the PC's screen.
***
***   Internals of MANDELM
***   --------------------
***
***   The task contains three threads.
***
***   (1) The MAIN thread.
***   This runs in the function main(). It intialises the
***   other two threads and then goes into a loop, once round
***   for each Mandelbrot display. For each, it gets
***   instructions from the user, and then signals the SEND
***   thread to start work by using the parameters_are_ready
***   semaphore. It keeps track of completed work by
***   examining tally_done, which is incremented by RECEIVE
***   everytime a RESULTS packet is displayed; when-
***   ever it notices that tally_done has changed, it updates
***   the PC's display; and when tally_done reaches 100, MAIN
***   knows that the display is complete.
***
***   (2) The SEND thread.
***   This knows when to start work by examining the
***   parameters_are_ready semaphore. It then breaks the job
***   into 100 small jobs, places the details into a COMMAND
***   structure (defined in file MANDEL.H) and uses the
***   net_send function to send it off to the network of
***   MANDELW's. Notice the SEND does not specify WHICH
***   worker task is to do any particular job;  this is
***   decided by the network of router tasks.
***
***   (3) The RECEIVE thread.
***   This simply waits till a packet arrives from the
***   network of MANDELW's and then displays it. Each packet
***   contains all the necessary information to display it,
***   so RECEIVE does not need to keep track of which packet
***   is which. Every time it does a display, RECEIVE
***   increments tally_done, so that MAIN can tell when the
***   whole display is complete.
***
***   Ver. 1.1  16-Dec-87   JF
***
***/

/* Modified for fault tolerance by W. Benage 29 Mar 1989 */
/* Lines not identified as added for fault-tolerance are */
/* original 3L Ltd. code. */

#include <stdio.h>
```

```
#include <dos.h>
#include <thread.h>
#include <sema.h>
#include <par.h>
#include <net.h>
#include <timer.h>
#include "cga.h"
#include "mandel.h"  /* Modified for fault-tolerance.*/
                     /* SAVE data type added . */

/* Interface to SEND thread */
static SEMA parameters_are_ready;

/* Fault-tolerance data structure (SAVE). */
static SAVE s[10][10];
           /* These integers in [] must match the */
           /* divisors in #define X_INCREMENT and */
           /* #define Y_INCREMENT lines.          */

/* Interface to RECEIVE thread */
static int tally_done;

/*Fault-tolerant variables. */
static int resend;
static int no_updated, last_i, last_j;

/* Current Mandelbrot and display parameters */
static float x_coord, y_coord, gap;
static int thresh1, thresh2, thresh3;

/* Define the way the job is broken into packets */
#define X_INCREMENT ((CGA_LORES_XMAX+1)/10)
#define Y_INCREMENT ((CGA_YMAX+1)/10)
#define PACKETS 100

#define D 15625 /* Number of low pri ticks in 1 sec. */
                /* Part of fault-tolerant timer system. */


/*
 * This function is invoked by MAIN using thread_create to
 * create the SEND thread.
 *
 */

send ()
{
/* k, l, i, j, and delay are integer variables added */
/* for fault-tolerance. */
   int                 k, l, i, j, x, y, delay;
   COMMAND             c;
```

```
delay = 50 * D;
    /* Sets length of fault-tolerant delay in sec.*/
for (;;) {

 /* Wait here until MAIN signals it's okay to go ahead */

   sema_wait (&parameters_are_ready);

   /* Fill in the fixed parts of the command */
   c.x_coord = x_coord;
   c.y_coord = y_coord;
   c.gap = gap;

/* Send off the packets to be done. Each includes the
   coordinates of the top-left and bottom-right corners of
   the area to do. This both tells the worker task what values
   to generate and identifies the RESULTS packet when it
   arrives in the RECEIVE thread (since there's no guarantee
   that the results will arrive in the same order the commands
   were sent out)                */

   resend = 0;
          /* Initialized for fault-tolerant system. */
   i = 0; /* Initialized for fault-tolerant system. */
   for (x = 0; x < CGA_LORES_XMAX; x += X_INCREMENT) {
      c.tlx = x; c.brx = x + X_INCREMENT - 1;
      j = 0; /* Initialized for fault tolerant system. */

      for (y = 0; y <= CGA_YMAX; y += Y_INCREMENT) {
         c.tly = y; c.bry = y + Y_INCREMENT - 1;
         /* Send off the next packet */
         net_send (sizeof(COMMAND), &c, 1);
      /* Save coordinates of next packet and time sent */

         /* for fault-tolerant system. */
         s[i][j].time = timer_now ();
         s[i][j].tlx = c.tlx;  s[i][j].brx = c.brx;
         s[i][j].tly = c.tly;  s[i][j].bry = c.bry;
        /* Resend any packet not received back in the */
         /* "delay" time for fault-tolerance.          */

         for (k = 0; k <= i; k += 1) {
            for (l = 0; l <= j; l += 1) {
               if ((s[k][l].returned == 0) &&
                  ((timer_now () - s[k][l].time) > delay)){
                  c.tlx = s[k][l].tlx; c.tly = s[k][l].tly;
                  c.brx = s[k][l].brx; c.bry = s[k][l].bry;
                  resend += 1;
                  net_send (sizeof(COMMAND), &c, 1);
               }
            }
         }
```

```
            c.tlx = s[i][j].tlx;
            c.brx = s[i][j].brx;
            j += 1;
         }
         i += 1;
      }
/* resend any packet not yet received back for */
/* fault-tolerance */
      while (tally_done < PACKETS) {
         for (i = 0; i < 10; i += 1) {
            for (j = 0; j < 10; j += 1) {
               if ((s[i][j].returned == 0) &&
                  ((timer_now () - s[i][j].time) > delay)) {
                  c.tlx = s[i][j].tlx; c.tly = s[i][j].tly;
                  c.brx = s[i][j].brx; c.bry = s[i][j].bry;
                  resend += 1;
                  net_send (sizeof(COMMAND), &c, 1);
               }
               else
                  thread_deschedule ();
            }
         }
      }

   }
}


/*
 * This function is invoked by MAIN using thread_create to
 * create the RECEIVE thread.
 *
 */

receive ()
{
   RESULTS     r;
/* j, n_inc, rtn, and count are added for the */
/* fault-tolerant system.*/
   int        len, ready, x, y, i, j, n, n_inc, colour, rtn,
count;

   count = 0; /* Initialized for fault-tolerant system. */
   for (;;) {

      /* Thread will wait here till a packet arrives */
      len=net_receive (&r, &ready);

      count += 1;
        /* Counts number of packets received back for */
                  /* for the fault-tolerant system. */
```

```
      /* "if" statement rejects all packets received after */
      /* number 100. Part of fault-tolerant system.  */
      if (tally_done < PACKETS) {

      /* The results packet includes the coordinates of the
         top-left and bottom-right corners of the data, so we
         know where to display it.
      */

         rtn = 0; /* Initialized for fault-tolerant system */

/* Fault-tolerant code for determining what packets have */
/* returned. */
         for (i = 0; i < 10; i += 1) {
            for (j = 0; j < 10; j += 1) {
               if (s[i][j].tlx == r.tlx && s[i][j].tly ==
                r.tly && s[i][j].brx == r.brx &&
       s[i][j].bry == r.bry){
                  if (s[i][j].returned == 0) {
                     n_inc = 0;
          /* "for" loops using y and x are 3L Ltd. code */
                     for (y=r.tly; y<=r.bry; y++) {
                        for (x=r.tlx; x<=r.brx; x++) {
                           n = r.counts[n_inc++];
                        /* Received 0 means 1, etc. */
                           n += 1;
                           colour = (n>=thresh1) +
                          (n>=thresh2) +
                          (n>=thresh3);
                           cga_lores_plot (x, y, colour);
                        }
                     }
    /* Remaining code in this thread is part of fault- */
    /* tolerant system. */
                     no_updated += 1;
                  }
     /* Let SAVE sys know when a packet has been received */
                  s[i][j].returned = 1;
                  last_i = i; last_j = j;
               }
               if (s[i][j].returned == 1) {
                  rtn += 1;
               }
            }
         }

      /*  Update the tally of packets displayed */
         tally_done = rtn;

 /* Set i to zero for the 2 min. timer when tally_done is */
 /* equal to or greater than PACKETS.                     */
      i = 0;
```

```
        }

        /* Account for all packets sent or wait 2 min. */
        else {
           if (((count - PACKETS) <= resend) && (i < 120)) {
              tally_done = count;
              i += 1;
           }
        /* Reset count when all packets are returned or 2 min */
        /* have elapsed.                                      */
           if (((count - PACKETS) == resend) || (i >= 120)) {
              count = 0;
           }
        }

     }
}


/*
 * The MAIN thread runs here
 *
 */

main ()
{

   float range;
    /* i and j are added for fault-tolerant system. */
   int previous_tally, i, j;

/* Make sure we have text mode (and clear screen), then
                  sign on */

   video_mode (MONO_80COL_TEXT_MODE);
   printf ("\nCopyright (c) 1988 3L Ltd\n\n");
   printf ("Example program: Mandelbrot set evaluation and
display\n");
   printf ("NB: This program requires a Colour Graphics
Adaptor\n\n");

  /* Initialise this SEMA to 0 BEFORE we start the SEND
     thread. This means it will wait until we tell it it's
     safe to go ahead   */

   sema_init (&parameters_are_ready, 0);

   /* Now start the other two threads */
   thread_create (send, 20000, 2,0,0);
   thread_create (receive, 20000, 2,0,0);

   for (;;) {
```

```
      /* Initialize s matrix for detecting packets not */
      /* returned for fault-tolerant system. */
      for (i = 0; i < 10; i += 1) {
         for (j = 0; j < 10; j += 1) {
            s[i][j].returned = 0;
         }
      }
      no_updated = 0; /* Initialized for the */
                      /* fault-tolerant system. */

      /* This will ensure that no other threads are using
         the C run-time library (in fact, in this case they
         won't be, but I have done it here as an example...)

      */
      sema_wait (&par_sema);
      printf ("\nInput X coordinate: "); scanf ("%f",
&x_coord);
      printf ("Input Y coordinate: "); scanf ("%f", &y_coord);
      printf ("Input Y range:      "); scanf ("%f", &range);
      gap = range / (float)(CGA_YMAX+1);
      y_coord = y_coord + range;

      printf ("Threshold 1: "); scanf ("%d", &thresh1);
      printf ("Threshold 2: "); scanf ("%d", &thresh2);
      printf ("Threshold 3: "); scanf ("%d", &thresh3);
      getchar ();  /* Consume the final NL */

      /* We have finished with the C RTL - release it */
      sema_signal (&par_sema);

      /* Into graphics (CGA low resolution) mode */
      video_mode (CGA_LORES_GRAPHICS_MODE);

      /* Before we set SEND going, reset the count of finished
         packets to zero - RECEIVE will count it back up
        */
      tally_done = 0;
      /* All ready - set it going! */
      sema_signal (&parameters_are_ready);

      previous_tally = 0;

   /* Until all the packets have been done, just keep updating
      the display when necessary
   */
      while (tally_done < PACKETS) {
         while (tally_done==previous_tally) {
        /* Wait here till something happens. Use
           thread_deschedule to save cpu time
        */
```

```
            thread_deschedule ();
         }
         /* Send the picture up to the PC's display memory */
         cga_update ();
         previous_tally = tally_done;
      }

      /* This ensures that the PC display is up-to date. */
      cga_update ();

    /* Once again, wait for the RTL to be free; then beep and
       wait till the user strikes any key
    */
      sema_wait(&par_sema);
      putchar ('\007');
      getchar ();
      sema_signal(&par_sema);

      /* Clear the screen and set text mode again */
      video_mode (MONO_80COL_TEXT_MODE);

     /* Print out items monitoring fault-tolerant system. */

      printf ("Number updated = ");
      printf ("%d",no_updated);
      printf ("\n");
      printf ("Last i = ");
      printf ("%d",last_i);
      printf ("   Last j = ");
      printf ("%d",last_j);
      printf ("\n");
      printf ("Resend = ");
      printf ("%d",resend);
      printf ("\n");

  /* Wait for all resent packets to return or */
  /* if failure occured during processing, wait 2 min. */
  /* This insures all packets resent by fault-tolerant */
  /* system have returned. */
      i = 0;
      while ((tally_done < (PACKETS + resend)) && (i < 120))
{
         timer_delay (D);
         i += 1;
         thread_deschedule ();
      }
   }

}
```

## APPENDIX E

## ORIGINAL CODE

```
/***   MANDELM.C[5]
 ***
 ***   Copyright (c) 1988 3L Ltd
 ***
 ***   Example program: Mandelbrot set evaluation and display.

 ***   NB: This application requires a Colour Graphics
 ***   Adaptor.
 ***
 ***   The application
 ***   ---------------
 ***
 ***   The application consists of two tasks:
 ***
 ***   (1) MANDELM (this file). This is the master task, and
 ***   runs in the root transputer.
 ***   (2) MANDELW.   This is the worker task, and runs in
 ***   all the other transputers of the net.
 ***
 ***   The flood configurer, FCONFIG, can be used to produce
 ***   an executable file which will automatically distribute
 ***   the worker tasks across an arbitrary network and route
 ***   work packets from the master to the workers.
 ***
 ***   It is also possible to run the application in a single
 ***   transputer. This will work automatically if the
 ***   application is configured using FCONFIG. Alternatively,
 ***   a static single-transputer configuration could be built
 ***   by hand, using CONFIG. A suitable configuration file
 ***   may be found in MANDEL.CFG.
 ***
 ***   As well as various routines from the Parallel C
 ***   run-time library, MANDELM must be linked with the CGA
 ***   primitives module, CGA.BIN. A file MANDELM.LNK is
 ***   supplied, which may be used to link MANDELM, like this:
 ***
 ***      LINKT @MANDELM.LNK,MANDELM.B4
 ***
 ***   Functions of the tasks
 ***   ----------------------
 ***
 ***   MANDELM is told by the user which part of the
```

[5]Reprinted with permission.

```
***   MANDELM is told by the user which part of the
***   Mandelbrot set to evaluate. It then breaks this up into
***   100 packets, and sends them to the network of
***   MANDELW's. As the results from each return, they
***   are displayed on the PC's screen.
***
***   Internals of MANDELM
***   --------------------
***
***   The task contains three threads.
***
***   (1) The MAIN thread.
***   This runs in the function main(). It intialises the
***   other two threads and then goes into a loop, once round
***   for each Mandelbrot display. For each, it gets
***   instructions from the user, and then signals the SEND
***   thread to start work by using the parameters_are_ready
***   semaphore. It keeps track of completed work by
***   examining tally_done, which is incremented by RECEIVE
***   everytime a RESULTS packet is displayed;  when-
***   ever it notices that tally_done has changed, it updates
***   the PC's display;  and when tally_done reaches 100,
***   MAIN knows that the display is complete.
***
***   (2) The SEND thread.
***   This knows when to start work by examining the
***   parameters_are_ready semaphore. It then breaks the job
***   into 100 small jobs, places the details into a COMMAND
***   structure (defined in file MANDEL.H) and uses the
***   net_send function to send it off to the network of
***   MANDELW's. Notice the SEND does not specify WHICH
***   worker task is to do any particular job;  this is
***   decided by the network of router tasks.
***
***   (3) The RECEIVE thread.
***   This simply waits till a packet arrives from the
***   network of MANDELW's and then displays it. Each packet
***   contains all the necessary information to display it,
***   so RECEIVE does not need to keep track of which packet
***   is which. Every time it does a display, RECEIVE
***   increments tally_done, so that MAIN can tell when the
***   whole display is complete.
***
***   Ver. 1.1  16-Dec-87   JF
***
***/

#include <stdio.h>
#include <dos.h>
#include <thread.h>
#include <sema.h>
#include <par.h>
```

```
#include "cga.h"
#include "mandel.h"

/* Interface to SEND thread */
static SEMA parameters_are_ready;

/* Interface to RECEIVE thread */
static int tally_done;

/* Current Mandelbrot and display parameters */
static float x_coord, y_coord, gap;
static int thresh1, thresh2, thresh3;

/* Define the way the job is broken into packets */
#define X_INCREMENT ((CGA_LORES_XMAX+1)/10)
#define Y_INCREMENT ((CGA_YMAX+1)/10)
#define PACKETS 100


/*
 * This function is invoked by MAIN using thread_create to
 * create the SEND thread.
 *
 */

send ()
{
   int                  x, y;
   COMMAND              c;

   for (;;) {

     /* Wait here until MAIN signals it's okay to go ahead */
      sema_wait (&parameters_are_ready);

      /* Fill in the fixed parts of the command */
      c.x_coord = x_coord;
      c.y_coord = y_coord;
      c.gap = gap;

      /* Send off the packets to be done. Each includes the
         coordinates of the top-left and bottom-right corners
         of the area to do. This both tells the worker task
         what values to generate and identifies the RESULTS
         packet when it arrives in the RECEIVE thread (since
         there's no guarantee that the results will arrive in
         the same order the commands are sent out)
     */
      for (x = 0; x < CGA_LORES_XMAX; x += X_INCREMENT) {
         c.tlx = x; c.brx = x + X_INCREMENT - 1;
         for (y = 0; y <= CGA_YMAX; y += Y_INCREMENT) {
            c.tly = y; c.bry = y + Y_INCREMENT - 1;
```

```
            /* Send off the next packet */
            net_send (sizeof(COMMAND), &c, 1);
         }
      }

   }
}


/*
 * This function is invoked by MAIN using thread_create to
 * create the RECEIVE thread.
 *
 */

receive ()
{
   RESULTS              r;
   int                  len, ready, x, y, i, n, colour;

   for (;;) {

      /* Thread will wait here till a packet arrives */
      len=net_receive (&r, &ready);
      i = 0;

      /* The results packet includes the coordinates of the
         top-left and bottom-right corners of the data, so we
         know where to display it.
      */
      for (y=r.tly; y<=r.bry; y++) {
         for (x=r.tlx; x<=r.brx; x++) {
            n = r.counts[i++];
            /* Received 0 means 1;  received 255 means 256 */
            n += 1;
            /* Decide on the colour <- thresholds, and
               display...*/
            colour = (n>=thresh1) + (n>=thresh2) +
           (n>=thresh3);
            cga_lores_plot (x, y, colour);
         }
      }

      /*  Increment the tally of packets displayed */
      tally_done += 1;

   }
}


/*
 * The MAIN thread runs here
```

```
 *
 */

main ()
{

   float range;
   int previous_tally;

/* Make sure we have text mode (and clear screen), then sign
   on */
   video_mode (MONO_80COL_TEXT_MODE);
   printf ("\nCopyright (c) 1988 3L Ltd\n\n");
   printf ("Example program: Mandelbrot set evaluation and
display\n");
   printf ("NB: This program requires a Colour Graphics
Adaptor\n\n");

   /* Initialise this SEMA to 0 BEFORE we start the SEND
      thread. This means it will wait until we tell it it's
      safe to go ahead */

   sema_init (&parameters_are_ready, 0);
   /* Now start the other two threads */
   thread_create (send, 10000, 2,0,0);
   thread_create (receive, 10000, 2,0,0);

   for (;;) {

   /* This will ensure that no other threads are using the C

      run-time library (in fact, in this case they won't be,

      but I have done it here as an example...)          */

      sema_wait (&par_sema);
      printf ("\nInput X coordinate: ");
                                        scanf ("%f", &x_coord);
      printf ("Input Y coordinate: "); scanf ("%f", &y_coord);
      printf ("Input Y range:      "); scanf ("%f", &range);
      gap = range / (float)(CGA_YMAX+1);
      y_coord = y_coord + range;

      printf ("Threshold 1: "); scanf ("%d", &thresh1);
      printf ("Threshold 2: "); scanf ("%d", &thresh2);
      printf ("Threshold 3: "); scanf ("%d", &thresh3);
      getchar ();  /* Consume the final NL */

      /* We have finished with the C RTL - release it */
      sema_signal (&par_sema);

      /* Into graphics (CGA low resolution) mode */
```

```
        video_mode (CGA_LORES_GRAPHICS_MODE);

        /* Before we set SEND going, reset the count of finished
           packets to zero - RECEIVE will count it back up  */

        tally_done = 0;
        /* All ready - set it going! */
        sema_signal (&parameters_are_ready);

        previous_tally = 0;

        /* Until all the packets have been done, just keep
           updating the display when necessary              */
        while (tally_done < PACKETS) {
           while (tally_done==previous_tally) {
              /* Wait here till something happens. Use
                 thread_deschedule to save cpu time      */
              thread_deschedule ();
           }
           /* Send the picture up to the PC's display memory */
           cga_update ();
           previous_tally = tally_done;
        }

        /* In case tally_done was updated to = PACKETS AFTER
           the last cga_update, do another one to ensure the
           PC's display is up-to-date */

        cga_update ();

        /* One again, wait for the RTL to be free; then beep
           and wait till the user strikes any key      */

        sema_wait(&par_sema);
        putchar ('\007');
        getchar ();
        sema_signal(&par_sema);

        /* Clear the screen and set text mode again */
        video_mode (MONO_80COL_TEXT_MODE);
    }

}
```

## LIST OF REFERENCES


1. Losq, J., "Effects of Failures on Gracefully Degradable Systems," a paper presented at the International Conference on Fault-Tolerant Computing, 7th Annual, Los Angeles, California, 28-30 June 1977.

2. Kravetz, G., "Redundant Parts Keep Systems Running," Computer Design, v. 27, pp. 80-84, 15 May 1988.

3. Antola, A. and Scarabottolo, N., "Reconfiguration Policies in a Multimicroprocessor Environment," a paper presented at the Proceedings of the ISMM International Symposium, Bari, Italy, 5-8 June 1984.

4. Popli, S. and Bayoumi, M., "Faut Diagnosis and Reconfiguration for Reliable VLSI Arrays," a paper presented at the Seventh Annual International Phoenix Conference on Computers and Communications, Scottsdale, Arizona, 16-18 March 1988.

5. OCCAM 2 Reference Manual, Prentice Hall International (UK) Limited, 1988, Hertfordshire, United Kingdom.

6. Parallel C User Guide, 1988, 3L Limited, Livingston, United Kingdom.

7. Cmelik, R., Gehani, N. and Roome, W., "Fault-Tolerant Concurrent C: a Tool for Writing Fault Tolerant Distribution Programs," a paper presented at The Eighteenth International Symposium on Fault-Tolerant Computing, Tokyo, Japan, 27-30 June 1988.

8. Parallel Fortran User Guide, 1988, 3L Limited, Livingston, United Kingdom.

9. Wolverton, R., "Scftware Costing", a paper in Handbook of Software Engineering, edited by Vick, C. and Ramamoorthy, C., Van Nostrand Reinhold Company Inc., New York, New York, 1985.

10. "Ada on the Inmos Transputer," a pamphlet from Alsys, Alsys Incorporated, Waltham, Massachusetts.

11. E mail, Subject: Ada for Inmos Transputer, 3 April 1989.

12. Express, 1988, ParaSoft Corporation, Mission Viejo, California.

13. Preliminary Data IMS T414 Transputer, December 1986, INMOS Limited, Bristol, United Kingdom.

14. Filho, J., Test and Evaluation of the Transputer in a Multi-Transputer System, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1987.

15. Bryant, G., Design, Implementation and Evaluation of an Abstract Programming and Communications Interface for a Network of Transputers, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1988.

16. Pikelis, W., Dynamic Link Switching and Link Fault-Tolerance in a Network of Transputers, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1989.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 2 |
| 3. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943 | 1 |
| 4. | Dr. Uno R. Kodres, Code 52Kr<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943 | 3 |
| 5. | Lieutenant Commander J. Yurchak, USN, Code 52Yu<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943 | 3 |
| 6. | Commanding Officer<br>Naval Surface Weapons Center, Code 20<br>Dahlgren, VA 22449 | 1 |
| 7. | Commander, Naval Sea Systems Command<br>PMS 400B5<br>Washington, DC 20362 | 1 |
| 8. | Randal Larsen<br>Digital Systems Integrators<br>P.O. Box 405<br>Alamo, CA 94507 | 1 |
| 9. | RCA AEGIS Repository<br>RCA Corporation<br>Government Systems Division<br>Mail Stop 127-327<br>Moorestown, NJ 08057 | 1 |

10. Library (Code E33-05) 1
Naval Surface Weapons Center
Dahlgren, VA 22449

11. Office of the Chief of Naval Operations (OP-35) 1
Department of the Navy
Washington, DC 20350-2000

12. Captain John Gauss 1
Commander,
Space and Naval Warfare Systems Command
PMW-162
Washington, DC 20363-5100

13. Captain F. Williamson 1
Cruise Missile Project (PMA282)
Naval Air Systems Command HQ
Washington, DC 20361-1014

14. Andre M. vanTilborg 1
Computer Science Division, Code 1133
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

15. Dr. Roy Lee 1
Sandia National Laboratory
Mail Stop 4233
Livermore, CA 94551

16. Dr. Tony Degroot 1
(L 795)
Lawrence Livermore National Laboratory
Livermore, CA 94550

17. Lieutenant W.F. Benage, USN 2
1961 Lincoln Street
Seaside, CA 93955

18. AEGIS Modeling Laboratory, Code 52 6
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943