NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

LARGE GRAIN DATA FLOW GRAPH CONSTRUCTION AND RESTRUCTURING UTILIZING THE ECOS WORKSTATION SYSTEM

by

Richard Toney Keys

September 1994

Thesis Co-Advisors: Amr Zaky
S. B. Shukla

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE LARGE GRAIN DATA FLOW GRAPH CONSTRUCTION AND RESTRUCTURING UTILIZING THE ECOS WORKSTATION SYSTEM	5. FUNDING NUMBERS
6. AUTHOR(S) Keys, Richard T.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
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 United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The U.S. Navy's new multiprocessor, the AN/UYS-2 Enhanced Modular Signal Processor (EMSP) utilizes a First-Come-First-Serve (FCFS) algorithm to transfer data. This algorithm is simple to implement but provides no mechanism to control execution of a specific application on the AN/UYS-2 which prevents performance predictions. A Large Grain Data Flow (LGDF) representation of a specific application is utilized to predict performance, with the introduction of trigger queues (dependency arcs) into the graphs to control execution.

I utilized the EMSP Common Operational Software (ECOS) Workstation to execute graph representations of specific applications used by the U.S. Navy in the Anti-Submarine Warfare (ASW) arena. A complete description of the ECOS workstation, and the process of transforming specific applications into graph representations to be executed on the ECOS Workstation is demonstrated. Specifically, the Correlator Graph which represents a real-time ASW process is examined

To control and improve performance, the technique of implementing trigger queues using the ECOS Workstation is demonstrated. A basic graph is executed and referenced as a benchmark, with two reconstructed graphs executed demonstrating how trigger queues effect graph execution. The node execution times statistics indicate trigger queues control execution and will provide a mechanism to predict node performance.

14. SUBJECT TERMS Data Flow Processing, Processing Graph Methodology, Signal Processing, EMSP, First-Come-First-Serve (FCFS), Revolving Cylinder Algorithm, Trigger Queues	15. NUMBER OF PAGES 90
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

LARGE GRAIN DATA FLOW GRAPH CONSTRUCTION AND RESTRUCTURING UTILIZING THE ECOS WORKSTATION

Richard T. Keys
Lieutenant, United States Navy
B.S., University of Mississippi, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: Richard T. Keys
Richard T. Keys

Approved By: Amr Zaky
Amr Zaky, Thesis Co-Advisor

[illegible]
Shridhar B. Shukla, Thesis Co-Advisor

for [illegible]
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The U.S. Navy's new multiprocessor, the AN/UYS-2 Enhanced Modular Signal Processor (EMSP) utilizes a First-Come-First-Serve (FCFS) algorithm to transfer data. This algorithm is simple to implement but provides no mechanism to control execution of a specific application on the AN/UYS-2 which prevents performance predictions. A Large Grain Data Flow (LGDF) representation of a specific application is utilized to predict performance, with the introduction of trigger queues (dependency arcs) into the graphs to control execution.

I utilized the EMSP Common Operational Software (ECOS) Workstation to execute graph representations of specific applications used by the U.S. Navy in the Anti-Submarine Warfare (ASW) arena. A complete description of the ECOS workstation, and the process of transforming specific applications into graph representations to be executed on the ECOS Workstation is demonstrated. Specifically, the Correlator Graph which represents a real-time ASW process is examined

To control and improve performance, the technique of implementing trigger queues using the ECOS Workstation is demonstrated. A basic graph is executed and referenced as a benchmark, with two reconstructed graphs executed demonstrating how trigger queues effect graph execution. The node execution times statistics indicate trigger queues control execution and will provide a mechanism to predict node performance.

TABLE OF CONTENTS

I. INTRODUCTION

The Twentieth Century has produced two great trends in Naval Warfare, the Weaponry Revolution and the Sensory Revolution. The advances in the weapons field has produced fire control systems which can deliver ordnance on target quickly and accurately. The aircraft carrier and the embarked air wing, nuclear powered fast attack submarines, and missile equipped surface combatants provides the US Navy with platforms which can deliver offensive firepower throughout the world. The range and speed of today's offensive weapons; jet aircraft, torpedoes, surface to air missiles and cruise missiles have greatly increased. This increase in range and speed has resulted in a sensor revolution which attempts to detect and locate the electronic devices used in these weapons systems. The area of the earth's surface which must be monitored to detect the enemy has increased at the same ratio as the weapons range. These surveillance systems must be able to detect and process video data onboard ships of the fleet before a coordinated, concentrated attack can be carried out.

Capt. Wayne Hughes (RET), in his book "Fleet Tactics" discusses six Measures required for effective attack; Strategic detection, Tactical detection, Tracking, Targeting, Attacking, and Damage Assessment [HUG 86]. The quick, accurate, and coordinated processing of data during the tactical detection, tracking and targeting phases used for an effective attack, must be improved to counter the ever increasing speed of today's modern offensive weapons. Two specific areas, the processing of radar (radio detection and ranging) data, and passive and active acoustic data requires millions of computations per second. The fusion of this data with other passive sensors such as Electronic Surveillance Measures (ESM), Cryptologic Support Measures, satellite imagery, and satellite data links provides the Warfare Commander with the data needed to organize and coordinate a concentrated attack on the enemy. To disperse the same tactical picture to all units in a task group, the signal processing methods must be improved to provide an accurate real-time picture of events. A delay in signal processing and fusion of available data inputs will only provide a history of what happened, in other words, information which is worthless to the tactical commander during the heat of battle in today's fast paced environment.

Signal processing requirements for the Navy in 1990 ranged from 300 million floating point operations per second (MFLOP) for small airborne sensors, to 2.4 million MFLOPs for submarine sonar arrays [RIC 90]. But as the speed of offensive weapons increase, and current traditional von Neumann architectures reach physical limits of technology, an increase in signal processing rates must be accomplished to compete in the twenty-first century. The problem encountered here is a multi-disciplinary one. Network theorists try to find optimal transformation methods and transparent and clear representation domains for the algorithms; mathematicians provide methodologies and algorithms for solving the related mathematical problems; VLSI designers have to bridge the gap to silicon when designing the microprocessor [VAN 91]. To meet these needs, the United States Navy must utilize a system which can test and optimize new signal processing methods, and test the techniques on new microprocessors before being built in the lab and introduced into the fleet.

To meet the needs of today's smaller Department of Defense's budgets, the U.S. Navy must maximize simulation in the lab, analyzing how the new system would perform with existing systems. The ECOS Workstation (EWS) will be used to test signal processing methods by using Processing Graph Methods (PGM), the algorithm being implemented to process the data flow, and simulation of the proposed the multiprocessor providing statistical data which can be used to optimize the system.

Specifically, the EWS was developed to test the Enhanced Modular Signal Processor (EMSP) AN/UYS-2, which will have applications to the AN/SQS-89 sonar, Surveillance Towed Array Sensor System (SURTASS), and Airborne Low Frequency Sonar (ALFS) programs. The AN/UYS-2, attempts to maximize the concurrency inherent in data-flow architectures, while minimizing the associated overhead, using a hybrid architecture. Such an architecture would take advantage of control-flow for the task level, and data-flow at the functional level [ATT 93].

A. BACKGROUND

1. Data Flow and the EMSP

Data flow representation of digital signal processing algorithms provides a natural exploitation of concurrence [LIT 91]. In a signal processing graph the execution of a particular node is controlled by the amount of information on each of its input queues. For each node, each incoming queue has an associated minimum amount, or threshold, of information needed for the operation's execution. When all of a node's input queues contain sufficient information to meet the respective thresholds, the node is ready for execution. This node represents the head of the queue, while the source node represents the tail. Information is deposited on, and removed from, queues in different methods. One method is the First In, First Out (FIFO) algorithm, which schedules by maintaining a database consisting of PGM graph node and Arithmetic Processor information. The simplicity of the FIFO algorithm earns it the designation as the most attractive scheduling algorithm [LIT 91].

2. The Multiprocessor

Once the signal processing application and the Data Flow Algorithm have been optimized, the results must be tested on the proposed multiprocessor. An analysis of the hardware configuration and its major subsystems; Functional Elements, Data Transfer Network, Functional Element Control Bus and how these operate together, is important in optimizing the entire process. The EWS provides the statistics to examine each Node, queue, and architecture of the specific multiprocessor.

The AN/UYS-2 is meant to provide the United States Navy with a standard, programmable, modular, multi-processor capable of meeting the digital signal processing requirements into the twenty-first century. All previous signal processors in the U.S. Navy utilized time-line control-flow architectures. Here a series of instructions and corresponding data are processed sequentially and initiated by a single control signal. This narrow connection between instructions and data held in memory and the single processing unit forms von Neumann's bottlenecks. The bottleneck can be avoided by removing the assumptions implicit in a von Neumann design [LEW 92].

3. Processing Graph Method (PGM)

The PGM was developed to provide the signal processing engineer a convenient means to write applications software without having to be concerned with the architecture of the machine on which it would be run [NRL 90]. PGM allows a signal processing application to be described as a collection of signal processing graphs in a manner similar to the common use of block diagrams [ATT 88]. The implementation of PGM allows the automated translation of a representation of a signal processing graph into a collection of software modules which carry out the application functions.

In signal processing, as in any high data rate, processor-intensive computer application, the name of the game is throughput. The more data the computer can process per second, the better. The simplest method is to speed up the processors. Until recently, computer processors executed programs one operation at a time. Consequently, the algorithms written to solve problems were designed to perform one step at a time; such algorithms are called serial. However, many computationally intense problems cannot be solved in a reasonable amount of time using serial operations [ROS 91].

A second method to increase the throughput of data is by the use of several processors running in parallel. Parallel processing, which uses computers made up of many separate processors, each with its own memory, helps overcome the limitations of serial computers. Parallel algorithms, which break a problem into a number of subproblems that can be solved concurrently, can then be devised to rapidly solve problems using a computer with multiple processors. In a parallel algorithm, a single instruction stream controls the execution of the algorithm, sending subproblems to different processors, and directs the input and output of these subproblems to the appropriate processors.

When parallel processing is used, one processor may need output generated by another processor. Consequently, these processors need to be interconnected. However, an imprudent assignment of tasks to processors can cause unnecessary interprocessor communications. The communication time - the amount of time spent moving data between processors - can dominate the computation time and thus limit the realized performance [HAM 92]. The use of PGM, allows the analysis of the interconnection network for the parallel processors, thus optimizing the system by using the most appropriate graph for the problem presented. The type of interconnection network used to implement a particular parallel algorithm depends on the requirements for exchange of data between processors, the desired speed, and the available hardware.

B. OBJECTIVE

The ECOS Workstation provides the tools for an analysis of each phase involved from graph creation to multiprocessor development. This thesis will examine the procedures required to implement and develop a graph, a scheduling algorithm, and the AN/UYS-2 multiprocessor. This system produces two outputs, graphical analysis static (gas) and event time simulator (ets++), which will be analyzed to provide insight into what each output file means towards the development of an optimized, efficient signal processing system. A case study utilizing the Correlator Graph model to demonstrate this systems' capabilities from design to implementation will be examined. The implementation of trigger queues will be demonstrated to show how a graphs execution can be modified with this technique.

C. SCOPE, LIMITATIONS, AND ASSUMPTIONS

The scope of this investigation is limited to performing testing and analysis of an example graph, input/output file, and machine configuration file, explaining how these files are integrated together to produce the gas and ets++ output. The EWS system will then be executed using a correlator graph to provide insight into the statistics provided for optimization of a graph. Trigger queues will then be demonstrated, and their effect on a graph's performance.

D. ORGANIZATION OF THESIS

This thesis is organized into five chapters. This chapter provides the introduction and scope of work to be performed. Chapters II provides a background on the ECOS Workstation (EWS) and the specifics of each phase of development, and the files involved in this investigation. Chapter III will explain the Correlator Graph and associated files, which will provide an example of the system in use. Chapters IV examines the implementation of trigger queues into a graph and the methods utilized with the ECOS simulator. Chapter V covers what conclusions can be derived from these results.

II. ECOS FUNDAMENTALS

A. BACKGROUND

The ECOS Workstation (EWS) is a suite of tools used to develop, analyze, test and optimize AN/UYS-2 signal processing applications. The system allows the implementation of signal processing algorithms using the Processing Graph Method (PGM) and the testing of the signal flow characteristics with different configurations of the AN/UYS-2 architecture prior to actually building the hardware in the lab. EWS may be run on a variety of platforms and is used independently of the actual AN/UYS-2 machine. ECOS (EMSP Common Operational Support Software) refers to the implementation of the Processing Graph Method (PGM) on the AN/UYS-2 previously known as the EMSP (Enhanced Modular Signal Processor). [ATT 92]

PGM provides a convenient way of specifying the signal processing algorithms to be executed by application software programs by allowing the description of the algorithms in terms of signal processing graphs. These graphs are analogous to be the block diagrams commonly employed as high level summaries of the signal processing algorithms.

The AN/UYS-2A architecture is determined by the needs of the application. Each configuration must contain one Data Transfer Network (DTN), one Scheduler (SCH), and one Command Program Processor (CPP). The architecture accommodates various combinations of Input Output Processors (IOP's), Input Signal Conditioner's (ISC's), Global Memories (GM's), and Arithmetic Processor's (AP's). [NRL 1]

B. THE ECOS PROCESS

The phases of the ECOS process, utilized to create a graph, implement a data processing algorithm, optimize the graph, and configure an AN/UYS-2 multiprocessor are the key aspects of this system. A graph file, command program file, Input/Output procedure file, and Machine Configuration file are needed to run the system. The following information provides an in-depth explanation of how the ECOS simulator works and how the files are integrated to produce the data flow characteristics used to optimize the graph. The best way to explain this system is to use an example and provide a detailed explanation of each integral part of the system and it's relevance during ECOS simulator operation.The sample graph file, command program file, Input/Output procedure file, and Machine Configuration file demonstrates the process from graph development to system optimization.

1. Graph File

The EWS visual graph editor, **gred**, speeds up application development by allowing the design of signal processing programs in the form of ECOS data-flow graphs as represented in Figure 1. These graphs automatically generate Signal Processing Graph Notation (SPGN) source code for the application, freeing the programmer from the task of programming in SPGN code. The system allows the graph objects, which are the building blocks of a gred graph, and represent one step in the flow of signal processing data to be created:

A signal processing graph consists of a set of nodes representing the primitive processing elements of the graph, a set of subgraphs representing more complex processes, a set of merges for use in controlling the flow of data, a set of queues representing the directed information flow through the graph, and a set of graph variables holding arbitrary information. The **INPUT QUEUE** provides the interface between an input processor and the graph. The **OUTPUT QUEUE** provides the interface between the graph and an output processor. The **GRAPH INSTANTIATION PARAMETER (GIP)**, is a start time constant which is passed to a graph by the command program at graph instantiation time. The **VAR (GRAPH VARIABLE)** is a variable which is passed to a graph by the command program, or passed to a subgraph by a parent graph. The value of the Var may be re-written during run-time, and the new value may be passed to the graph during run-time. A **LOCAL QUEUE** represents the directed flow of information from node to node, from node to subgraph, or from subgraph to node within the graph and carries information on a first-in, first-out basis into a node or subgraph. A **NODE** in a graph embodies a specific signal processing operation, called a primitive, and a Primitive Interface Procedure (PIP), which provides the interface between the node and the primitive. A **SUBGRAPH** allows hierarchical structures in a graph definition. A subgraph is originally defined as a graph and then used in the definition of a larger graph. A **CONNECTION** is the object used to connect NODES, Queues, Subgraphs, INPUTQ, OUTPUTQ, and CONSTRUCTOR Objects. An **INPUT CONSTRUCTOR** consolidates data from various graph objects and passes it to a node or subgraph as a single input. An **OUTPUT CONSTRUCTOR** takes output data from a node or subgraph and distributes it to various graph objects. These are the objects required to build and test the flow of information on a graph. [ATT 92]

The EWS provides a number of tools for graph creation. The graph topology is implemented using **gred**, which permits an object-oriented style of code creation. Once created, the graph is converted to SPGN using the **grail** translator. This converts the graphical representation into compilable code, which is checking for syntax and continuity errors. This new file is executed using **ggcc**, which is three programs which validate and compile the SPGN; **grasp**, **glitr,** and **cpcc**. **Grasp** translate the SPGN source code into graph tables. **Glitr** translates the tables in C language routines and **cpcc** compiles the C routines and command program, then links all the programs. Once the executable file is created, static and dynamic simulations may be accomplished using two additional EWS tools, Graph Analysis, Static (**gas**) and Event Time Simulator (**ets++**).

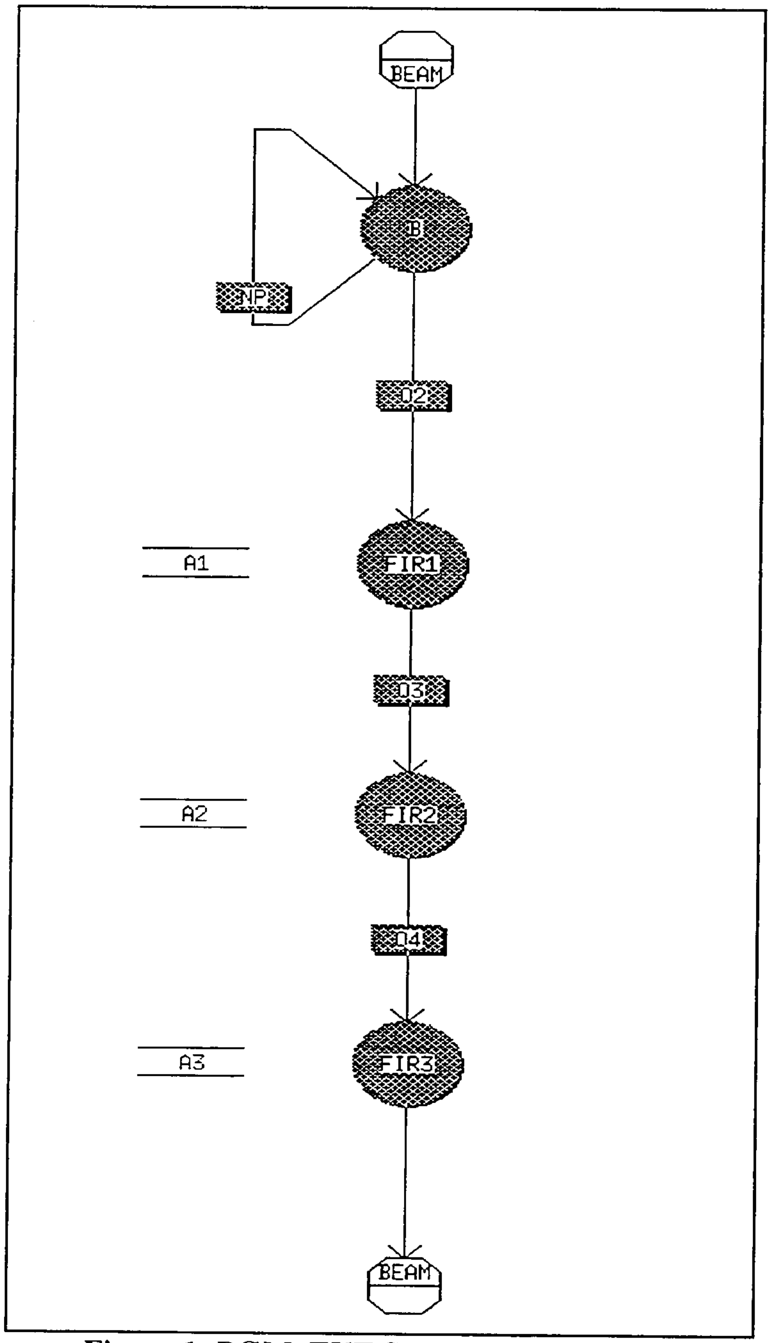


Figure 1: PGM_TUT Sample graph

2. Command Program File

The EWS command program defines how an application connects with the outside world via the input and output procedures, formal input and output queues, graph variables and graph instantiation parameters, and how various graph within an application connect to one another. The command program is used together with the SPGN source file produced by **grail** to compile the application.

PROGRAM	COMMENTS
`%INITCOMPROG()`	Opening statement of each Command program
`IO_PROC_ID      iop1,iop2;`	- I/O procedures
`QUEUE_ID        in1, ou1;`	-- Formal input/output queues
`GRAPH_ID        G;`	-- Graph's name
`in1 = %CREATEQ(FLOAT);`	-- Create the queue named in1
`ou1 = %CREATEQ(CFLOAT);`	-- Create the queue named ou1
`G = %START(PGM_TUT`	-- Start graph PGM_TUT
`INPUTQ=in1`	-- Match INPUTQ with in1
`OUTPUTQ=ou1`	-- Match OUTPUTQ with ou1
`PRIORITY = 2`	-- State the graph's priority
`);`	
`if (%SCODE) exit(1);`	-- Check for correct SPGN, stop if errors detected.
`iop1 = %INITIO(IOP1`	-- Initialize the I/O procedure
`INPUTQ=in1);`	-- Match IOP1 with INPUTQ in1
`iop2 = %INITIO(IOP2`	-- Initialize the I/O procedure
`OUTPUTQ=ou1);`	-- Match IOP2 - OUTPUTQ ou1
`%STARTIO(iop1);`	-- Start I/O procedures iop1
`%STARTIO(iop2);`	-- Start I/O procedures iop2
`%PRINT(%TERM, 1, G);`	-- Prepares the compiled executable file to generate an ets++ input file. There is 1 graph, G.
`%ENDPROGRAM`	-- Final statement of each command program.

Figure 2: PGM_TUT Command Program

Each command program must be able to create the input and output queues for the graph, initialize the input/output procedures (i.e., attach the queues to them), start the graph with its required parameters, and start the input/output procedures. The command program example in Figure 2, lists two Input/Output procedures iop1, iop2 which correlate with two names that will be listed in the Input/Output procedure file (**pgm_tut.io**), and are started by %INITIO. Two formal input/output queues, in1 and ou1 which map to Beam_In and Beam_Out of the graph PGM_TUT in figure 1, and are initialized by %CREATEQ. The graph G is started by %START(PGM_TUT) which correlates to the file created using **gred.**

3. I/O procedure file

The EWS I/O procedure file provides information to **gas** and **ets++** about characteristics of the graph's input and output nodes. Input and output nodes are external to the graph; they feed outside world input data into the graph and read output data from the graph. The I/O procedure file contains details about incoming and outgoing data rates. These rates are used by **gas** and **ets++** in their various timing analysis.

The I/O procedure file **pgm_tut.io** displayed in FIGURE 3, provides the input to the ECOS simulator example. The IOP1 variable, declared as an input for the graph with one port, corresponds to the iop1 variable in the command program. The input rate into the graph is 300000 words per second with the output rate producing 2048 words per second directed into the output interface. The IOP2 variable, is an output variable with one port, and corresponds to the variable iop2 in the command program. This program has a threshold of 822 words, reads 822 words, and consumes 822 words.

```
#The format of the i/o procedure file is tabular; the information content is
# simple, so this should cause no problems, and it reduces the amount of baggage in
# the simulator for interpreting this file.

# NAME is the symbolic identifier of the i/o procedure named in the %INITIO
# statement in the command program; it must match exactly; case is significant.

# TYPE is one of INPUT, BIDIRECTIONAL, or OUTPUT

# DATA RATES for graph inputs are expressed in words/second

# THRESH, READ, CONSUME, and PRODUCE amounts are in words

# comments look like this, everything on a line to the right of the sharp sign is discarded

#                                   number of
# name           type                  ports
#----------      ----------            -----------
IOP1             INPUT                 1

#                rate                  produce
#---------       -----------           -----------
                 300000                2048

#                                      number of
# name           type                  ports
#----------      -----------           -----------
IOP2             OUTPUT                1

# thresh         read                  consume
#----------      -----------           ------------
822              822                   822
```

Figure 3: PGM_TUT Input/Output File

4. AN/UYS-2 Machine Configuration file

The EWS machine configuration file defines an AN/UYS-2 hardware configuration for use by **ets++**, will execute an application as if it were running on the hardware specified in the configuration. The hardware consists of the Data Transfer Network (DTN), the Distributor/Concentrator ports on the DTN, and the Functional Elements (FE). The DTN's come in three sizes: 4, 8 or 16 ports. The designer may assign up to four FE's per port, however, to minimize DTN contention, assign no more than one FE per port. A concentrator transfers data from a FE to the DTN, while a distributor transfers data from the DTN to a FE. The Functional elements consist of Arithmetic Processors, Input/Output Processors, and Global Memory. Each Functional Element is associated with a specific port.

Figure 4 provides a visual display of the AN/UYS-2 which would be tested by the ECOS system as determined by the machine configuration file. The DTN has eight ports, two of which are not used. Each Functional Element has it's own port to reduce data transfer conflicts.

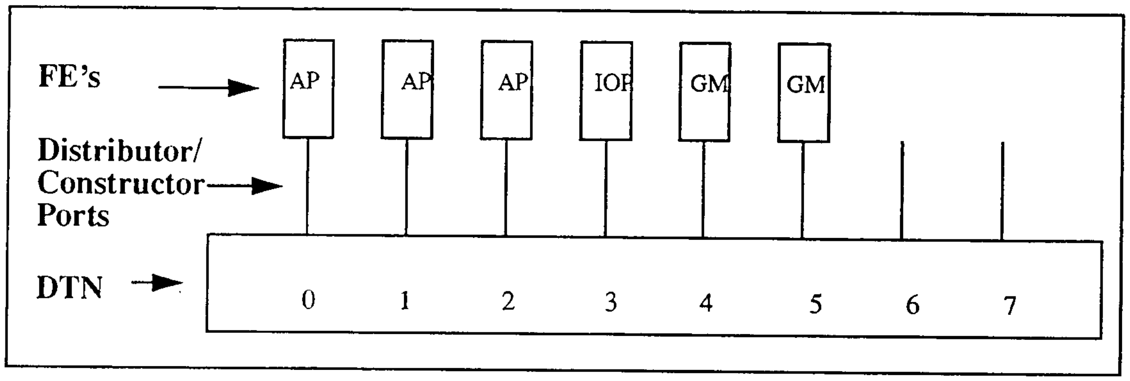


Figure 4: PGM_TUT Machine Hardware

Figure 5 is an example of a machine configuration file, **3ap2gm.cf** which is used in the ECOS example. This Configuration has three Arithmetic Processors, one Input/ Output Processor, and two Global Memories. The Data Transfer Network has eight distributors and eight concentrators. The Arithmetic Processors are Class one, meaning that the AP class is directed to use a particular AP load image for class one. AP 0 is connected to distributor 0 and concentrator 0, AP 1 to distributor 1/concentrator 1, AP 2 to distributor 2/concentrator 2, IOP 0 connected to distributor 3/concentrator 3, GM 0 to distributor 4/ concentrator 4, and GM1 to distributor 5/concentrator 5.

```
#ETS++ machine configuration file
# comment lines look like this, anything on a line after the sharp sign ("#") is discarded
# the format here is vaguely reminiscent of the actual configuration file
# keywords:

#CONFIGURATION
#HARDWARE
#                   AP (ARITHMETIC PROCESSOR)
#                   IOP (INPUT/OUTPUT PROCESSOR)
#                   GM (GLOBAL MEMORY)
#                   DISTRIBUTORS
#                   CONCENTRATORS

CONFIGURATION

# number of functional elements of each type
#                             FE
#type                         quantity
#-------                      --------
AP                            3
IOP                           1
GM                            2

# number of concentrators/distributors on DTN

DIST                          8
CONC                          8

HARDWARE

# FE                          FE             DIST           CONC           AP
#type                         name           ID             ID             CLASS
#----------                   -----------    -----------    ----------     ----------
AP                            AP:0           0              0              1
AP                            AP:1           1              1              1
AP                            AP:2           2              2              1
IOP                           IOP:0          3              3
GM                            GM:0           4              4
GM                            GM:1           5              5
```

Figure 5: PGM_TUT Machine Configuration File

C. ECOS EXECUTION PROCESS

The following sequence of events demonstrates the ECOS Workstation functions and how they are integrated together to produce the statistical data required to optimize the signal data flow represented by the graph. The PGM_TUT graph in Figure 1, the Command Program File **com_prog.cp,** I/O Procedure File **pgm_tut.io**, and Machine Configuration File **3ap2gm.cf** are utilized to demonstrate the execution of the system, with the **gas** and **ets++** output files produced, displayed and examined:

1. **gred**

The **gred** (graphical editor) is an executable file which invokes a graphical window which is used as a tool to create a graph which is saved as **pgm_tut.g.**

2. **grail -g pgm_tut.g > pgm_tut.src**

The **grail** -g **pgm_tut.g** > **pgm_tut.src** is the function which calls **grail** which converts the graph created in **gred** into SPGN compilable code. The **-g filename(pgm_tut.g)**, is the name of the graph file saved in **gred** and used by **grail** to identify the input. The >**pgm_tut.src**, is the new name of the file created by **grail.**

3. **ggcc pgm_tut.src com_prog.cp**

The **ggcc pgm_tut.src com_prog.cp** is the function call to **ggcc (grasp, glitr, cpcc, cpcc**), where **grasp** translates your SPGN source file into graph tables, **glitr** translates .int file into C routines, **cpcc** compiles the C files and **cpcc** compiles the command program and then links all the compiled files The **pgm_tut.src** file is the name of the file generated by **grail** and **com_prog.cp** is the name of the command program supplied to **cpcc.**

4. **com_prog > com_prog.ets**

The **com_prog > com_prog.ets** command, is the renaming of the file produced by **ggcc,** which is named **com_prog** and > **com_prog.ets** is used to generate the simulation input file for **gas** and **ets++**.

5. **gas -g com_prog.ets -i pgm_tut.io -n all -q all > pgm_tut.gout**

The **gas -g com_prog.ets -i pgm_tut.io -n all > pgm_tut.gout** command, is the call to the **gas** function, which is the graph analysis static, the analysis tool utilized first. It performs a quick, cursory check of the signal processing application based on the static properties of the graph. The **-g filename(com_prog.ets)** provides the name of the graph file to **gas**. The **-i filename(pgm_tut.io)** provides the name of the Input/Output file to **gas**. The

-n all provides the command which requests statistics on each node to **gas.** The > **pgm_tut.gout** is the name given to the output file produced by **gas.**

6. **ets++ -g com_prog.ets -i pgm_tut.io -m 3ap2gm.cf -n all -q all -t (time) > pgm_tut.eout**

The **ets++ -g com_prog.ets -i pgm_tut.io -m 3ap2gm.cf -n all -q all -t (time) > pgm_tut.eout** command, is the call to the **ets++** function, which is a dynamic 'event-time simulator' function used for modeling real-time graph performance for a given AN/UYS-2 configuration. The **-g filename(com_prog.ets)** provides the graph file name to **ets++.** The **-i filename(pgm_tut.io)** provides the Input/Output file name to **ets++.** The **-m filename(3ap2gm.cf)** provides the Machine Configuration file name to **ets++.**The **-n all** provides the command which requests all statistics on each node. The **-q all** provides the command which requests all statistics on each queue.The **-t** (***time***) - time may be in seconds, minutes or machine cycles. The default is seconds. To specify minutes, type -t timem and to specify machine cycles type -t timec. For example:

-t 10 = 10 seconds
-t 10m = 10 minutes
-t 10c = 10 machine cycles

The > **pgm_tut.eout** is the name given to the output file produced by **ets++.**

D. ECOS OUTPUT

1. gas analysis

The **gas** function does not simulate a graph execution and it has no information regarding the machine configuration. Rather, it takes a look at the graph's design and calculates execution statistics as if the graph were running in a perfect setting with unlimited hardware resources. **gas** performance statistics thus present a best case scenario. The analysis of the **gas** output file will check to see if the graph is structurally sound, report characteristics of the graph which allow identification of certain types of performance trouble spots, and report execution characteristics which allow the programmer to estimate the hardware configuration needed to run the application on an AN/UYS-2.

Three types of graph statistics appear in the **gas** output, scheduling rate and execution time statistics, System Language message statistics, and the Bandwidth statistics.The following acronyms are used in the **gas** output file:

- AIS - Accept Instruction Stream
- AP - Arithmetic Processor
- AU - Arithmetic Unit
- CQ - Consume Queue
- GM - Global Memory
- IOP - Input/Output Processor
- RGV - Read Graph Variable
- RQ - Read Queue
- SL - System Language
- WGV - Write Graph Variable
- WQ - Write Queue

a. Graph Statistics

Figure 6, which displays the graph statistics section, is the output file which has four categories. The first, SCHEDULING RATE AND EXECUTION TIME STATISTICS section provides four specific areas of information. The *total node scheduling rate* is the sum of the individual scheduling rates of all nodes. The *AP node scheduling rate* is the sum of the individual scheduling rates for all nodes which execute on an AP. The *total required AU cycles* is a weighted sum of the AP node execution times where the weights are the individual node execution rates. And the *mean AU cycles per node* is a weighted average of the per-node AU execution time.

b. System Language Statistics

The SYSTEM LANGUAGE MESSAGE STATISTICS provides thirteen different types of information for analysis. The *mean queue consumes/node* is the average number of CQ (consume queue) messages sent per node execution. The *mean queue consume amount* is the average amount consumed pr consume request. The *mean queue reads/node* is the average number of RQ (read queue) messages per node execution. The *mean queue read amount* is the average amount read per read request. The *mean queue writes/node* is the average number of WQ (write queue) messages per node execution. The *mean queue write amount* is the average size of a write request. The *mean graph variable reads* is the average number of RGV (read graph variable) messages per node execution. The *mean gv read amount* is the average size of an RGV request. The *mean graph variable writes* is the average number of WGV (write graph variable) messages per node execution. The *mean gv write amount* is the average size of a WGV request. The m*ean queue+gv read amount* is the average read amount (without distinguishing between a queue read and a GV read).The *mean queue+gv write amount* is the average write amount (without distinguishing between a queue write and a GV write). The *mean AIS size*, is derived from the following process; Every node execution involves the transfer of an instruction stream from a GM (global memory) to an AP (arithmetic processor) via an AIS (accept instruction stream) message.

c. Bandwidth Statistics

The BANDWIDTH STATISTICS section provides three specific areas of information for analysis. The *Bandwidth AP* is the rate at which data is moved between APs and GMs due to reading and writing queues and graph variables. The *Bandwidth IOP* is the rate at which data is moved between IOPs and GMs. The *Bandwidth GM* is the total rate at which data is moved into and out of GMs. This is the sum of *Bandwidth AP* and *Bandwidth IOP*.

```
Static Graph Analysis - EWS Release: 5.6
Invocation: gas -g com_prog.ets -i pgm_tut.io -n all

total graph objects: 4 nodes, 6 queues, 0 graphvars, 2 I/O procedures

graph: PGM_TUT
pids: /users/res/zakyclasses/EWS/sef/Pids/pidtoc
graph objects: 4 nodes[0,3], 6 queues[0,5], 0 graphvars, 2 I/O
procedures[0,1]

graph source: com_prog.ets
i/o procedure source: pgm_tut.io

  total node scheduling rate:      2434.99 / second
  AP node scheduling rate:         2343.75 / second
  total required AU cycles:       5.63e+06 cycles / second
  mean AU cycles per node:        2.40e+03 cycles
  mean queue consumes / node:         1.25 CQ    / node
  mean queue consume amount:        461.00 words / CQ
  mean queue reads / node:            1.25 RQ    / node
  mean queue read amount:           470.20 words / RQ
  mean queue writes / node:           1.25 WQ    / node
  mean queue write amount:          384.20 words / WQ
  mean graph variable reads:          0.00 RGV   / node
  mean gv read amount:                0.00 words / RGV
  mean graph variable writes:         0.00 WGV   / node
  mean gv write amount:               0.00 words / WGV
  mean queue+gv read amount:        470.20 words / (RQ,RGV)
  mean queue+gv write amount:       384.20 words / (WQ,WGV)
  mean AIS size:                    256.00 words / AIS
  Bandwidth AP:                    2.5e+06 words / second
  Bandwidth IOP:                  3.75e+05 words / second
  Bandwidth GM:                   2.88e+06 words / second

Individual node statistics
  Node Node           rate        rate       ------- number of -------    i/o rate
   id  name        (exec/sec)  (cycl/sec)  RQs  WQs  CQs  RGVs  WGVs  (word/sec)
  ---- ----        ----------  ----------  ---  ---  ---  ----  ----  ----------
   0  PGM_TUT>BANDSHIFT  586     932e6      2    2    2    0     0      901e3
   1  PGM_TUT>FIR1       586     2.14e6     1    1    1    0     0      906e3
   2  PGM_TUT>FIR2       586     1.69e6     1    1    1    0     0      461e3
   3  PGM_TUT>FIR3       586     865e3      1    1    1    0     0      236e3
```

Figure 6: PGM_TUT gas Output File

d. Node Statistics

The INDIVIDUAL NODE STATISTICS section provides six areas of information about the performance of each node in the graph. The *Node ID* is the unique numerical identifier for the node. The *Node name* is the hierarchical node name. The *rate (exec/sec)* is the number of node executions per second (the number of times per second that the node is called upon to process data). The *rate (cycle/sec)* is the number of AP cycles per second (processing speed) required to execute the node. This rate must be lower than the maximum processing speed of the AP. Otherwise, the node will become a bottleneck and the graph will not be executable on a machine of any size. The *number of RQs, WQs, CQs, RGVs, and WGVs* summarizes the number of SL messages per node execution. The *i/o rate (word/sec)* is the rate at which data is transferred between APs and GMs per node execution. This is calculated for each node by taking the number of words per execution due to RQ, WQ, RGV and WGV messages and multiplied by the number of executions per second. This gives a words per second result.

The results of the gas function is analyzed, providing the programmer with insight into how each node is operating in reference to the data flow rates. This allows the programmer to edit the application, or prepare the machine configuration file and perform a real-time simulation with **ets++**.

2. ets++ analysis

One of the central components of the ECOS Workstation is an EMSP system simulator which models the execution of a graph on the actual machine. **ets++** does not perform signal processing. It does simulate the data-flow scheduling of the functional elements in the AN/UYS-2 signal processor. Through this simulation, **ets++** computes timing and hardware usage statistics for the application. **ets++** models the data-flow scheduling between graph elements, the system language message traffic and the real time operations which take place in the functional elements (FEs) of an AN/UYS-2 during graph execution. It models the execution based on the AN/UYS-2 configuration specified in the machine configuration file.

ets++ provides four main types of information about your graph, overall system performance, individual functional element (FE) performance, graph execution characteristics, and individual graph object execution statistics.

The following is a list of acronyms used in the ets++ output file:

- RQ - Read queue
- AQ - Accept queue
- WQ - Write queue
- CQ - Consume queue
- RGV - Read graph variable
- AGV - Accept graph variable
- WGV - Write graph variable
- QOT - Queue over threshold
- QUT - Queue under threshold
- QOC - Queue over capacity
- QUC - Queue under capacity
- RFIS - Ready for instruction stream
- SIS - Send instruction stream
- AIS - Accept instruction stream
- EIS - Execute instruction stream
- SNDT - Suspend node data transfer
- CNDT - Continue node data transfer
- CTC - Change threshold and consume

3. The ets++ output file

The following is an analysis of the five sections of the output file produced by executing the **ets++** function. Utilizing the input files in the example with the gas function coupled with the machine configuration file **3ap3gm.cf, ets++** results are provided.

a. Header Section

The HEADER section provides general information about the **ets++** output file such as the EWS version being utilized, and the command line used to invoke **ets++**. The machine parameter file line, lists the directory and filename of the machine configuration file. The Configuration section lists the hardware makeup of the mircroprocessor being used in the simulation. The graph objects line, provides the number of nodes, the

```
EMSP Event Time Simulator - EWS Release: 5.6

Invocation: ets++ -g com_prog.ets -i pgm_tut.io -m 3ap2gm.cf -n all -q
all -t 1

Machine parameter file: '/users/res/zakyclasses/EWS/sef/lib/params.d'
of 06/22/92 15:58

Configuration
-------------
FE count:  7
AP count:  3
GM count:  2
IOP count: 1
concentrators: 8
distributors: 8

total graph objects: 4 nodes, 6 queues, 0 graphvars, 2 I/O procedures

graph: PGM_TUT
pids: /users/res/zakyclasses/EWS/sef/Pids/pidtoc
graph objects: 4 nodes[0,3], 6 queues[0,5], 0 graphvars, 2 I/O
procedures[0,1]


Statistics
----------

Simulation clock = 1.000 seconds
```

Figure 7: PGM_TUT ets++ Header

number of queues, the number of graph variables, and I/O procedures. The range of numerical identifiers for the graph objects and I/O procedures appears in square brackets.

Additionally, the graph name used in the simulation is listed, the location of the PID library, and the Statistics section states how long the simulation was executed.

b. Functional Element Statistics

An example of each type of Functional Element (arithmetic processor, global memory, input/output processor) and the scheduler, and the statistics provided follow:

```
FEID:  0  NAME: AP:0          TYPE: arithmetic processor
  conc id:  0   dist id:  0
 AP Class: 1
 CU idle =  50.70 %
 CU wait =   0.12 %
 CU busy =  49.18 %
      service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
      service wait per dtn  request:  avg = 7.71e-06 sec, max = 1.57e-04 sec
      cbus messages sent:  1808    received:    0
      dtn  messages sent:   646    received:  1163
 AG/AU nodes executed: 516  (CU only: 0)
 AG/AU busy: 17.62 %
 AG/AU pending: 51.27 %
AG/AU efficiency: 25.57 %
Free time (AP:0): 48.49 %
```

Figure 8: PGM_TUT ets++ AP Statistics

(1) The Arithmetic Processor Section:

This section is broken down into thirteen sections which provides vital information about the AP. The numerical ID for this FE comes from the machine configuration file. The *AP class* which is assigned during **gred**, can be used to group APs (Arithmetic Processors) into a particular class.The *CU idle* percentage, is the amount of the time that the Control Unit(CU) is not in use at all. The *CU wait* percentage, is time when the CU must wait for the completion of an event before it can do anything else. The *CU busy* percentage, is the time when the CU is busy when it is performing work such as SL message processing. The *service wait* is the amount of time a message arriving at the CBUS

(Control Bus) or DTN (Data Transfer Network) interfaces must wait before processing. The *cbus messages sent* is the number of SL (System Language) messages sent from this AP via the control bus. The *received* data, is the number of SL messages received by this AP via the control bus. The *dtn messages sent* information is the number of SL messages sent from this AP via the data transfer network. The *received* information, is the number of SL messages received by this AP via the data transfer network. The *AG/AU nodes executed*, is the number of AG/AU(Address Generator/Arithmetic Unit) nodes executed on this AP. The CU only section, is the number of nodes executed only on the Control Unit (CU), such as merge, flow control and AP replicate nodes. The *AG/AU busy* section is the percent of total simulation time that the AP's AU was executing primitives. The *AG/AU pending* section, is the percent of total simulation time that the AP was waiting to execute a primitive that has been assigned to it. The *AG/AU efficiency* section is the percent of the AP's 'in use' time that is actually used for doing productive work. The *Free time* data, is the percent of total simulation time that the AP was free to be assigned a node for execution.

(2) The Global Memory Section

The *GM Utilization (busy + dtn i/o)* statistics, is the percent of total simulation time that the GM was either processing SL messages (CU busy) or sending and receiving data to and from the DTN (dtn i/o). The *memory usage information*, where three types of memory usage statistics are produced; *high water mark, pointer block, and static*. *High water mark* is the maximum amount of memory used at any one time during the simulation. The *Pointer block* is the maximum memory used by the pointer block. The pointer block keeps track of the used and unused blocks of GM memory. Static, is the amount of fixed overhead memory, which remains constant throughout the simulation.

```
 FEID:  3  NAME: GM:0              TYPE: global memory
conc id:  4   dist id:  4
 CU idle =  53.91 %
 CU wait =   0.00 %
 CU busy =  46.09 %
 service wait per cbus request: avg = 4.29e-05 sec, max = 5.98e-04 sec
 service wait per dtn  request: avg = 1.97e-05 sec, max = 1.24e-04 sec
 cbus messages sent:  1226    received:  3227
 dtn  messages sent:  2001    received:  1309
GM Utilization (busy + dtn i/o):  60.40 %
memory usage,high water mark:108236; pointer block: 512, static: 4844
   Queue table:     128      GI table:   3000
    Node table:     128    Queue Hist:      0
      GV table:       0     Templates:   1040 (appr)
     GV memory:       0         delta:      0 (assm)
  Initial data:     544     CTC table:      4
Nodes: 2    Queues: 4    Graph Variables: 0
```

Figure 9: PGM_TUT ets++ Global Memory Statistics

(3) The I/O Processor Section

The Simple IOP model, provides the Basis information for the Control bus and Data Transfer Network. The *Number of WQs completed* is the number of Write Queue messages successfully sent from this IOP to a GM. The *cancelled by SNDT* section, is the number of WQ messages lost due to outstanding SNDTs (suspend node data transfer messages). The *Input queue* data, provides the Input queue # handled by this IOP. The *input rate (w/sec)* data, is the input data rate for that particular input queue #. The *Output nodes executed* provides the number of output nodes executed on this IOP. This allows the designer to analyze the amount of traffic on the two buses.

```
FEID:  6  NAME: SCH               TYPE: scheduler
CU idle =  84.99 %
CU wait =   0.01 %
CU busy =  15.00 %
service wait per cbus request:avg = 6.29e-06 sec, max = 1.54e-04 sec
service wait per dtn  request:avg = 0.00e+00 sec, max = 0.00e+00 sec
cbus messages sent:  1613    received:  3553
dtn  messages sent:     0    received:     0
```

Figure 10: PGM_TUT ets++ Scheduler Statistics

(4) The Scheduler Section

The SCHEDULER Section of the **ets++** output file provides statistical data about the control of the resources in the mircroprocessor. The *CU idle* data, is the percent of time that the Control Unit is not in use. The *CU wait* data, is the percent of control bus contention. The *CU busy* data, is the percent of time the scheduler is processing data, including System Language message processing and assignment of nodes to APs for execution.

c. Graph Execution Performance

The graph execution performance consists of the Control Bus, Data Transfer Network, the DTN Input FIFO,and Node Scheduling Statistics.

```
 CBUS Utilization Statistics

CBUS message count = 9165
idle = 96.20 %
busy = 3.80 %
```

Figure 11: PGM_TUT ets++ Control Bus Statistics

The Control Bus is a single access bus handling only one Functional Element at a time. The *CBUS message count*, conts the number of messages sent via the CBUS. The *idle* percentage data, is the percent of total simulation time that the CBUS was idle. The *busy* percentage data, is the percent of total simulation time that the CBUS is busy.

```
DTN Utilization Statistics
DTN message count = 5636

Conc 0: idle =  96.98 %  busy=  3.02 %  avg. wait= 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 1: idle =  97.03 %  busy =  2.97 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 2: idle =  96.92 %  busy =  3.08 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 3: idle =  97.59 %  busy =  2.41 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 4: idle =  90.96 %  busy =  9.04 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 5: idle =  91.29 %  busy =  8.71 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 6: idle = 100.00 %  busy =  0.00 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 7: idle = 100.00 %  busy =  0.00 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec

Dist 0: idle =  94.28 %  busy =  5.72 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 1: idle =  94.32 %  busy =  5.68 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 2: idle =  94.19 %  busy =  5.81 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 3: idle =  99.46 %  busy =  0.54 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 4: idle =  93.80 %  busy =  6.20 %  avg. wait = 2.79e-06 sec  max wait =  1.41e-04 sec
Dist 5: idle =  95.09 %  busy =  4.91 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 6: idle = 100.00 %  busy =  0.00 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 7: idle = 100.00 %  busy =  0.00 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
```

Figure 12: PGM_TUT ets++ DTN Statistics

The Data Transfer Network(DTN) Utilization Statistics reports four areas of information for each concentrator and distributor; idle, busy, average, wait and maximum wait. Data flows from an FE through a concentrator into the DTN. Data flows from the DTN through a distributor into the FE. The *idle* percentage, is the percent of total simulation time that the concentrator or distributor was not in use. The *busy* percentage, is the percent of total simulation time that the concentrator or distributor was in use, including resource contention delays. The *avg. wait* data is the average amount of time that an FE had to wait to use this resource. The *max wait* data is the maximum amount of time that any one FE had to wait to use this resource.

```
DTN input fifo statistics
    AP:0 (id =  0) PIP: max. messages =  2,  high water mark =  519
    AP:1 (id =  1) PIP: max. messages =  2,  high water mark =  519
    AP:2 (id =  2) PIP: max. messages =  2,  high water mark =  519
    GM:0 (id =  3) PIP: max. messages =  3,  high water mark = 2190
    GM:1 (id =  4) PIP: max. messages =  2,  high water mark = 1031
   IOP:0 (id =  5) PIP: max. messages =  1,  high water mark =  623
```

Figure 13: PGM_TUT ets++ DTN Input FIFO Statistics

For each AP, GM, and IOP Primitive Interface Procedure (PIP) First in, First out (FIFO) buffer, the DTN input statistics are provided. The *max messages* number is the highest number of messages that ever existed in the FIFO buffer at any one time. The *high water mark* number is the maximum number of 16-bit words waiting in the buffer at any one time. The FIFO buffer capacity is pre-set.

```
Node Scheduling Statistics
nodes scheduled = 1613
rate = 1613.0  nodes/sec
```

Figure 14: PGM_TUT Node Scheduling Statistics

The *nodes scheduled*, is the total number of nodes scheduled for execution for the length of the simulation. The *rate*, is the number of nodes scheduled divided by the simulated length of time.

d. System Language Statistics

This System Language messages list, provides the number of times each type of message was sent during the simulation. The two key categories, QOC (queue over capacity) and SNDT (suspend node data transfer) indicate that data has been backing up. If both values are greater than zero, there may be data loss.

Cumulative SL message statistics

```
RQ          1999
AQ          1999
WQ          2085
CQ          1999
RGV         0
AGV         0
WGV         0
QOT         2004
QUT         0
QOC         0
QUC         0
RFIS        1550
SIS         1553
AIS         1552
EIS/ESN     60
SNDT        0
CNDT        0
CTC         0

TOTAL       14801
```

Figure 15: PGM_TUT System Language Statistics

e. Individual Graph Object Execution Statistics

The *Node ID*, is the node's numerical identifier. The *Node name*, is the node's symbolic hierarchical name. The *GM id* is the numerical identifier of the GM that stores this node's instruction stream. The *times scheduled*, is the number of times the node was scheduled for execution. The *times waited for AP*, is the number of times the node was ready for execution but had to wait because there was no AP available. The *average wait for AP*, is the average amount of time, in seconds, that the node which is ready for execution had to wait for an AP to become available.The *average delay to exec*, is the average amount of time from when the node was scheduled for execution on an AP until execution actually begun. The *average breakdown delay*, is the average amount of time from the completion of microcode execution until the completion of output-data transfer to the GM's.

```
Graph Object Statistics

Node Statistics
----  ----------
                                   times     average     average     average
Node     Node          GM  times     waited    wait        delay       brkdown
 id      name          id  scheduled  for AP   for AP      to exec     delay
 ----    ----          ---- ---------  ------   --------    -------     -------
  0     >BANDSHIFT 3  389        197    1.19e-04    1.03e-03             6.49e-04
  1     >FIR1          4    389         0     0.00e+00  8.45e-04           3.64e-04
  2     >FIR2          3    388         0     0.00e+00  1.07e-03           3.85e-04
  3     >FIR3          4    387         0     0.00e+00  9.60e-04           7.70e-04
```

Figure 16: PGM_TUT Node Graph Object Statistics

(1) Queue Information

The *Queue id*, is the queue's numerical identifier. The *Queue name*, is the queue's symbolic hierarchical name. The *GM id*, is the numerical identifier of the GM that stores this queue's data. The *times written*, is the number of times the queue was written by it's source node. The *remaining words in queue*, is the number of 16 bit words remaining on the queue at the time this statistic was written.

```
Queue Statistics
----- ----------
                                          remaining
Queue Queue                 GM   times     words
 id   name                  id   written  in queue
----- -----                ---- -------- ---------
   0  >BEAM_IN               3      146     99840
   1  >BEAM_OUT              3      386        88
   2  >Q2                    4      389      1034
   3  >NP                    3      389         1
   4  >Q3                    3      388       530
   5  >Q4                    4      387       274
```

Figure 17: PGM_TUT Queue Statistics

The data provided by the **ets++** output provides the vital statistics for the complete analysis of the proposed graph, data-flow diagram, and algorithm used. The optimization of the new system can be performed prior to the actual building of a new multiprocessor.

III. PERFORMANCE ANALYSIS TOOLS: A CASE STUDY

A digital signal processor's performance can be measured by throughput. Unfortunately, the AN/UYS-2 's throughput is not based exclusively on its scheduling algorithm, the First-Come-First-Serve algorithm. The concurrent parts of the program or tasks, represented by nodes on the graph must be arranged in time and space to optimize performance. The analysis of a graph created at AT&T, the Correlator Graph, will provide actual hands on experience in the ECOS process and the optimization of a graph, data flow, and multiprocessor configuration. This process will highlight the features used to optimize a signal graph using the simulator.

A. CORRELATOR GRAPH

The Correlator Graph, constructed utilizing the **gred** function, (refer to Figure 18 and Figure 19), was created based on the task to be performed, the data flow topology required, and the available Primitive Interface Definitions (PIDs) accessible in the PID directory.

This Correlator Graph, named E006, consists of 27 nodes, 37 queues, and 4 input/ output procedures. Two input queues, named X and represented by the family input queue graph object feeds the graph. Data is directed to two separate nodes, FIXL1 and FIXL2, where two separate paths for data processing is accomplished. Upon completion of processing, Two output queues, named GRAMOUT and represented by the family output queue graph object provides the results to the Input/Output Processor (IOP) for distribution to an interface not associated with the Correlator Graph.

B. CORRELATOR GRAPH COMMAND PROGRAM

The command program for the Correlator Graph, named test.cp (refer to Appendix A), controls the graph's execution and interaction. The program starts the E006 graph, initializes four Graph Instantiation Parameters (GIPs) STI, F, SR, and TC, declares the

Input Queue will be a Family of Queues, consisting of two inputs designated as IN1 and IN2, and the Output Queue will be a Family of Queues, consisting of two outputs designated as OUT1 and OUT2. The program will then initialize the input and output queues with the %INITIO command, then start the processing with the %STARTIO command. From this brief description of the command program, it is obvious that the purpose of test.cp is control rather than signal processing. The command program basically initializes and starts the io-procedure and controls the flow of data to and from the outside world.

C. CORRELATOR GRAPH INPUT/OUTPUT PROCEDURE FILE

The I/O procedure file provides the characteristics of the Correlator Graph's input and output data rate. The Correlator Graph Input/Output file, named 1.io (refer to Appendix B), defines four I/O procedures, IN1 and IN2 which are for input, OUT1 and OUT2 which are for output and correspond to the input/output variables in the command program file test.cp. Each procedure has only one associated port. The data rate of IN1 and IN2 corresponds to 2048 words per second being generated by the I/O device onto the queue, which are expressed in 16-bit words/second. The produce amount of IN1 and IN2 is the size of the data blocks which the I/O input device feeds into the graph, which are expressed in 16 bit words. The data rate of OUT1 and OUT2 corresponds to a 513 words for the threshold level, 513 words for the read amount, and 513 consume amount, with each amount expressed in 16-bit words.

D. CORRELATOR GRAPH gas ANALYSIS

Now that the graph and command program have been created, compiled, and linked, the I/O procedure file prepared, and simulation input file generated, the **gas** analysis tool is utilized.

The Correlator Graph simulation file corr.ets (refer to Appendix C), and the I/O procedure file 1.io are utilized as inputs to **gas** which produces an output file, named corr.gout (refer to Appendix D). The gas environment simulates a perfect setting with unlimited hardware resources, from which the machine configuration needed for this graph can be determined. In particular, analysis of the AP node scheduling rate, the total required AU cycles, Bandwidth GM, Node rate (exec/sec), and Node rate (cycle/sec) provides the necessary information for AN/UYS-2 configuration.

The *AP node scheduling rate* of 3.19/second indicates this node scheduling rate achieves the estimated rate based on the nature of the processing task and the intended throughput rate. If the rate had been less, the graph may have run successfully for short bursts, but the machines' resources would become swamped in the long run. The maximum throughput can be computed by taking the number of processors and multiplying them by the clock speed, divide that number by the *total required AU cycle* field, and take the result and multiply by the input data rate.

$$Max\ Throughput = \frac{Num\_of\_Processors \times Clock\_Speed}{total\_required\_AU\_cycles} \times Gas\_Data\_Rate \qquad (Eq\ 3.1)$$

The *total required AU cycles* of 3.90e+04 cycles per second, provides a rough estimate of the minimum amount of AP processing power needed to meet the real-time scheduling requirements of the graph. A SEM-E AP with a 10 MHZ clock can handle 10e+6 cycles per second. The minimum number of processors needed for the graph can be computed by dividing the *total required AU cycles* rate by the processor speed.

$$Min\ Processors = total\ required\ AU\ cycles\ /\ processor\ speed \qquad (Eq\ 3.2)$$

The *Bandwidth GM* of 1.09e+5 words/second, is the sum of *Bandwidth AP* and *Bandwidth IOP*. This figure provides an estimate of the minimum amount of GMs needed for this machine. A SEM-E GM can transfer data over the DTN at a rate of 10e+6 32-bit wps or 20e+6 16-bit wps. The minimum required Global Memory can be computed by dividing the *Bandwidth GM by* 10e+6 or 20e+6.

Required GM = Bandwidth GM / Clock Speed (Eq 3.3)

The *Node rate (exec/sec)* provides the number of times per second that the node is called upon to process data. Verification that expected values are consistent with given input data rates for the graph are necessary to insure a smooth data flow.

The *Node rate (cycle/sec)* value provides the number of AP cycles per second (processing speed) required to execute the node. This rate must be lower than the maximum processing speeds of the AP or a bottleneck will occur at that node and the graph will not execute.

E. CORRELATOR GRAPH MACHINE CONFIGURATION FILE

The machine configuration file for the Correlator Graph, 4-6.cf (refer to Appendix E), lists in tabular form, the machine configuration of the AN/UYS-2, to be utilized as input to the ECOS analysis tool, ets++. The information provided by the analysis of the gas output for the Correlator Graph was utilized to construct the proposed hardware.

The 4-6.cf machine configuration file lists four Arithmetic Processors, six Global Memories, two Input/Output Processors, and One Scheduler Processor. The Data Transfer Network (DTN) has eight Distributor/Concentrator ports, the ports numbered 0-7. The four Arithmetic Processors are on ports 0-3, the six Global Memories are on ports 4-6, with two GMs per port, and the two Input/Output Processors on port 7.

F. CORRELATOR GRAPH ets++ ANALYSIS

The ets++ event time simulator performs dynamic graph simulation on the Correlator Graph using the machine configuration file created from analysis of the gas simulation. The output file of the Correlator Graph ets++ simulation, corr.eout (Refer to Appendix F), provides the vital statistics needed for analysis. Through this simulation, timing and hardware usage statistics are computed which are utilized for further debugging, configuration sizing and performance optimization of the Correlator Graph.

To fully understand the optimization of the Correlator Graph, an analysis of the Functional Elements; arithmetic processors, global memory, Input/output processor scheduler, the data transfer network, scheduler and system language messages must be performed.

The arithmetic processor statistic *Free Time* is the key field for analysis of the AP functional element. A Free Time Value of under 10% indicates insufficient AP resources. If an AP runs out of free time, the Correlator Graph will fail. Free Time for AP0, AP1, AP2, and AP3 is 99.83%, 99.83%, 99.84%, and 99.86% respectfully. The APs are providing the Correlator Graph with a sufficient amount of Free Time which guarantees the multiprocessor will have an AP available when needed.

The Global Memory Statistics *high water mark* is the key data field for analysis. This *high water mark* is the sum of the *static memory*, the *dynamic memory*, and the *pointer block* utilized simultaneously. The GM supports optimized performance through memory management of data.

The *cancelled by SNDT* (Suspend Node Data Transfer) is the key field for analysis for the input/output processor. If this value is not zero, it indicates data is backing up on the graph. Input/Output processor IOP0 and IOP1 for the Correlator Graph are both zero, indicating a free flow of data.

The *avg wait* field for the DTN utilization is the key field for analysis. If the wait statistic is too high for any concentrator or distributor, indications are that congestion exists, and a modification to the AN/UYS-2 must be accomplished for the system to executed smoothly. The data for the Correlator Graph and the machine configuration show little contention, thus indicating optimal performance.

The *high water mark* field of the *DTN input fifo statistics* is the key field for analysis. This field tracks the maximum number of 16-bit words waiting in the buffer at any one time. If the FIFO (First-In-First-Out) buffer capacity exceeds the 32K for SEM-B or 8K for SEM-E, a blocked message is substituted. The Correlator Graph performance is outstanding with GM0 and GM4 each having 16689 words in the buffer, well below the 32K maximum.

The System Language message field which provides key data are the *QOC* (queue over Capacity) and *SNDT* (Suspend Node Data Transfer). These messages indicate that data has been backing up. If both values are greater than zero, there is a risk of losing data. The Correlator Graph simulation performance has *QOC* equal to one, *SNDT* equal to zero, so performance is not degraded.

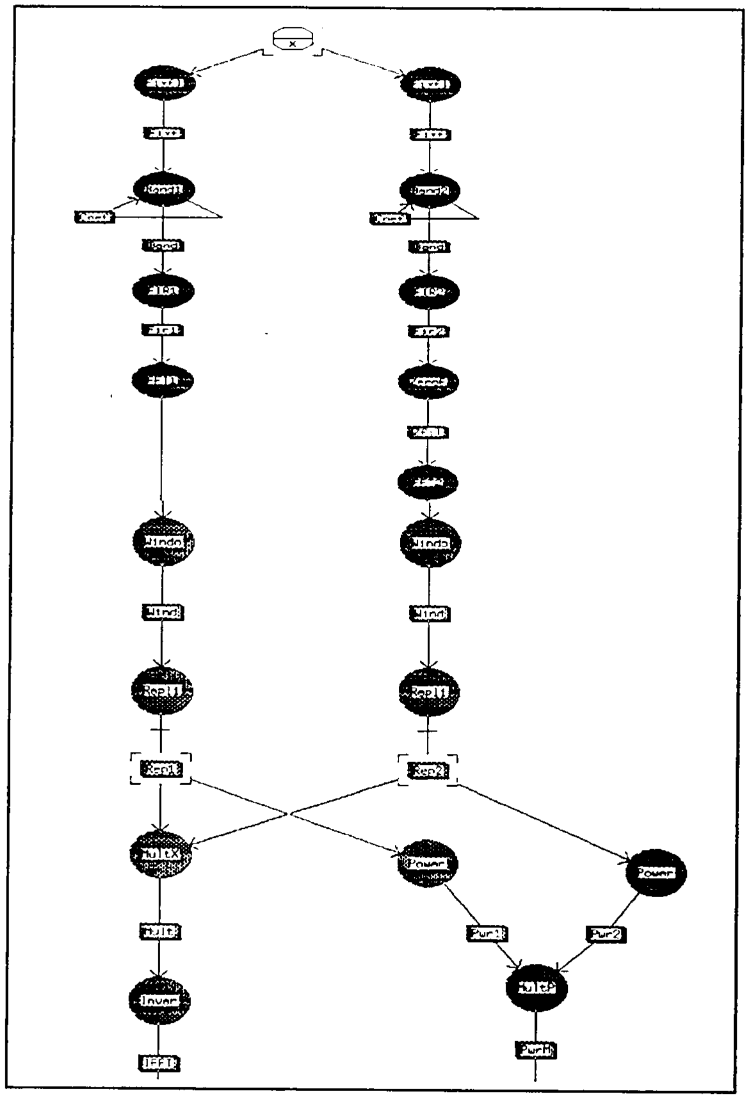


Figure 18: Correlator Graph Part 1

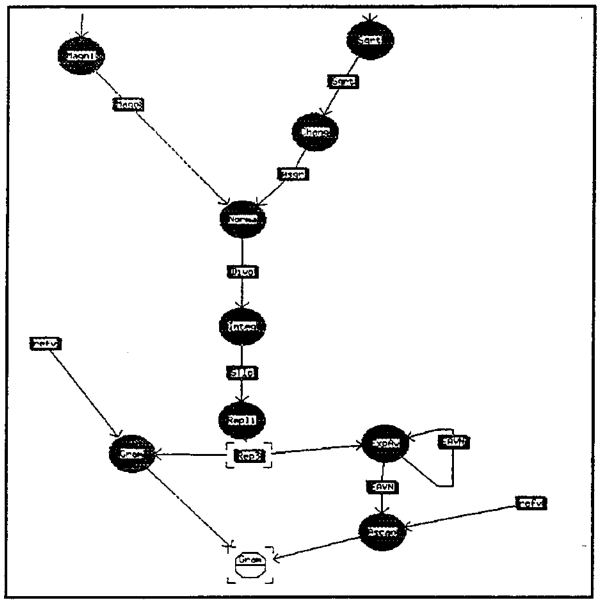


Figure 19: Correlator Graph Part 2

The simulation process is now complete for the Correlator Graph, with results indicating that this graph will function in an optimal capacity in its current configuration. The ECOS simulator has provided a data flow path for a specific hardware configuration which meets the requirements for the current data input rate as described in the input/output file.

IV. SYNCHRONIZATION TRIGGER QUEUES

A. INTRODUCTION

Analyzing and predicting the performance of the AN/UYS-2, the multiprocessor which the ECOS system simulates, is an extremely complicated process. The graph representation used by the ECOS simulator provides an excellent tool to perform analysis of how different processes integrate together to complete the execution of a specific graph. But, a second area which must be addressed, is the transfer of data between nodes during execution. Currently, the AN/UYS-2 utilizes a First-Come-First-Serve algorithm which does not provide any characteristics which can be utilized to predict performance. A possible solution is the implementation of the Revolving Cylinder algorithm by Graph Restructuring. This process, along with its implementation on the ECOS system and the simulation of several graph's to demonstrate how node execution can be controlled will follow.

B. GRAPH RESTRUCTURING

Graph Restructuring provides a means to control the graph's execution, while not modifying the basic purpose of the graph. As data rates increase, or the issue of scalability is applied to a graph, the execution of a graph and the data transfer between nodes must be computed to insure sufficient hardware configurations are available. An algorithm which overcomes these issues and which does not change the semantics of the graph, is the RC algorithm.

The key idea in the RC algorithm is based on inserting dependencies into the graph. These dependencies will be implemented into a graph by inserting trigger queues (dependency arcs) onto the graph which utilizes the FCFS algorithm. Essentially, the execution of nodes in the graph will be controlled by the trigger queues, and their associated parameters.

The advantages of the RC algorithm are that each execution of a graph will set up a system where performance can be predicted, reduce memory contention, and maximize communication and computation overlap. This will make the system have more predictable average response time and throughout. Thus memory contention can be reduced because the nodes and queues can be mapped to different memory modules. Another advantage of RC is the ability to achieve maximum overlap of communication time with computation time by placing the cylinders in appropriate order of execution.

C. ADDING TRIGGER QUEUES

The graph in Figure 20, the PGM_TUT sample graph, will be used as an example in demonstrating the implementation of the TRIGGER Queues onto a graph. The graph will be modified step by step, with an explanation of the transformation to the RC technique.

Trigger queues must consist of artificial data flows. Implementation is accomplished using the same structures seen previously for traditional data flow, node production sent to an assigned queue in memory, via an output port, then to the successor node via an input port. In the case of hierarchical structures, triggering pulses are passed using the actual input and output constructs [SWK 93].

To implement, changes must be made to the graph by using the graph editor **gred**. Four basic steps must be accomplished to implement the trigger queue. The local queue must be created and the characteristics declared. The queue must be connected to a node, and the threshold level declared. The node of interest must be modified to reflect the new input into the node, and possible output to a Revolving Cylinder queue. The output connection from the node to a trigger queue must be edited to reflect the change.

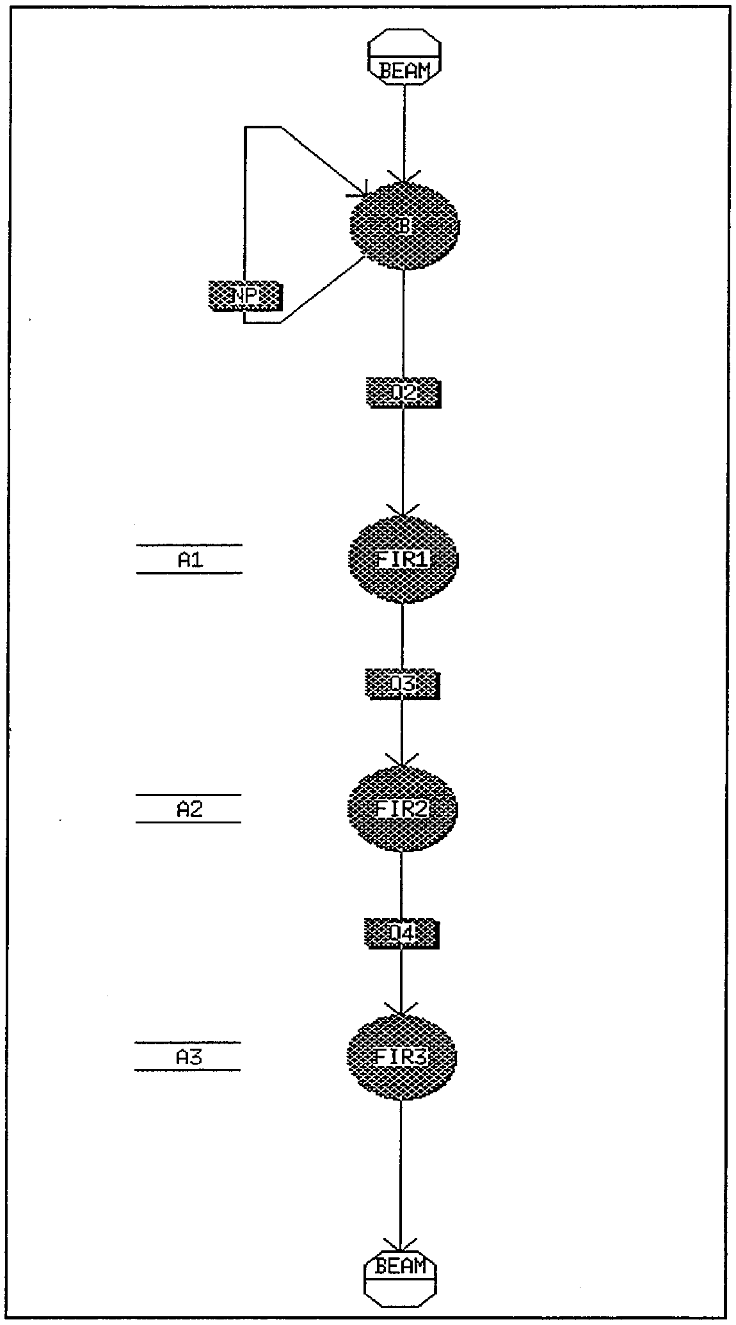
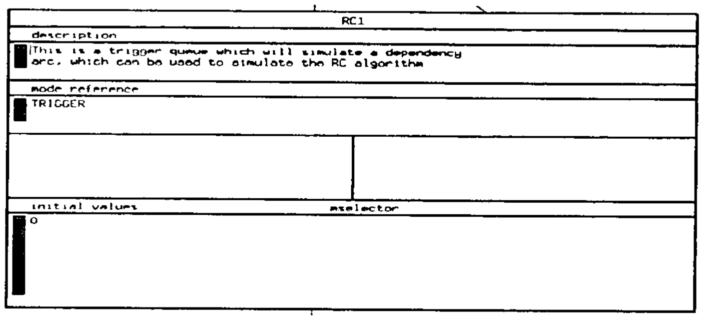


Figure 20: PGM_TUT Sample graph

1. Trigger Queue

The trigger queue channels synchronization signals called trigger queue pulses rather than data. This allows the forced synchronization of an otherwise asynchronous, data driven sequence of operations. The data elements on a trigger queue are pulses which have no semantic value.

To implement a trigger queue on a graph, the graphical editor **gred**, must be utilized. Invoke the gred function, and read in the graph to be modified. Select INTRODUCE / LOCAL QUEUE, choose single or family, then place the queue on the graph in a position relative to the node which will be modified. The next step is the naming of the queue. Select the queue with the arrow and depress the #1 button (the left of three), at which time a menu will pop up, select name. Enter a name in the text bar section of the gred display. Depress the return button, then the #2 (center button) on the mouse which acknowledges the 'ok' display.


RC1

description

This is a trigger queue which will simulate a dependency
arc, which can be used to simulate the RC algorithm

mode reference

TRIGGER

initial values | mselector

0

Figure 21: Queue Definition Window

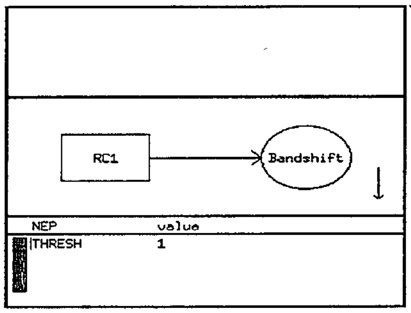
2. Initializing the Trigger Queue

Now that the queue has been named, the internal characteristics must be defined. The queue is edited by selecting the queue with the #1 mouse button. Depress the #3 mouse (right) button, a pop up menu appears and select the edit%Queue. The Local Queue Definition Window as in Figure 21, will appear. The description field is used to enter text which will appear as comments in the SPGN code. The mode reference field is where the mode of the queue is defined by selecting the TRIGGER mode. The initial values field refers to the number of trigger pulses to be placed on the trigger queue. A single value may be entered. The member selector (mselector) field is used when a family local queue is being defined, and used to enter initial values into the queue.

If the queue is a family trigger queue, two additional fields, the family dimensions/local index, and initial value/mselector fields can be defined. The family dimensions of a queue could be a one dimension such as 1..8. The initial value / member selector field can be input by mapping the values to the member selector.

3. Input Port Connection and Setting Threshold Levels

The queue must be connected to the node which will receive the trigger pulses. This is accomplished by creating a connection. Select the INTRODUCE / CONNECTION, single or family (must correspond to the queue definition). Depress the #2 mouse button on the queue, then the #2 mouse button on the destination Node. The characteristics of this Port must now be defined. Place the mouse button arrow on the Port and depress #1 mouse button to edit the port. The Port definition window will appear. Enter the Node Execution Parameter (NEP) THRESH (see Figure 22), which represents the number of data elements required to be on the queue in order for the node to execute.



NEP	value
THRESH	1

Figure 22: Input Port Connection

4. Output Port Connections

The trigger queue must be connected from a node with input to the queue. The OUTPUT Port connection (see Figure 23) menu, is utilized to declare the number of trigger pulses to be placed on a trigger queue. In Figure 23, the Node Execution Parameter is declared as type pulse, with a member selector of 1 which means one trigger pulse is placed onto the connection.

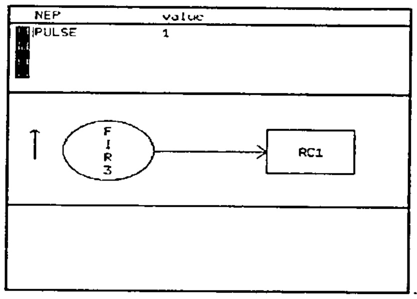
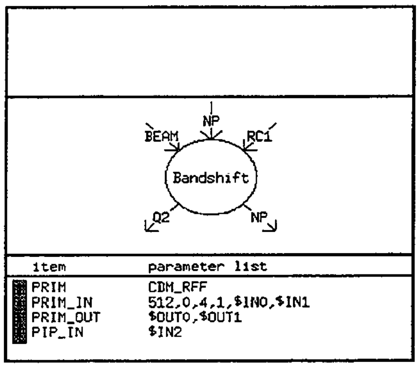


Figure 23: Output Port Connection

5. Modifying Node Descriptions

When a node is declared in a graph, a Primitive Interface Definition (PID) must be used to define the node descriptions. But trigger queues are not accounted for in the PID library in the ECOS system. To counter this, the trigger queues must be introduced using the Parallel Interface Port (PIP) [NRL 90]. This method of using the PIP allows data that is not specified within the PRIM_IN macro of the PID to be utilized as a normal execution function.

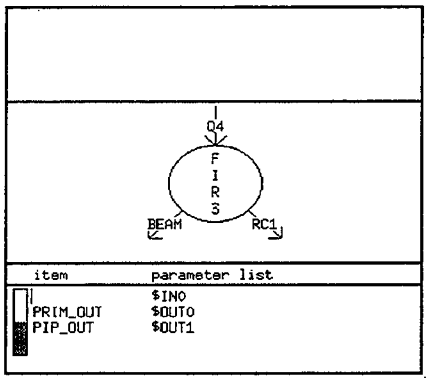
The node's PIP allows information to be passed into a node via the inport port which is not associated with the node primitive. The declaration PIP_IN, utilizes the macro $INx.



item	parameter list
PRIM	CDM_RFF
PRIM_IN	512,0,4,1,$IN0,$IN1
PRIM_OUT	$OUT0,$OUT1
PIP_IN	$IN2

Figure 24: PIP_IN Declaration

The node's PIP may pass information out through node output ports not associated with the node's primitive. Such a node output port is attached to a queue of mode trigger. The PIP may place zero or more initial pulses on the trigger queue. The declaration PIP_OUT, which utilizes the macro $OUTx, specifies the actual parameters passed out of the node which do not originate at the node's underlying primitive.



item	parameter list
	$IN0
PRIM_OUT	$OUT0
PIP_OUT	$OUT1

Figure 25: PIP_OUT Declaration

If the node is connected to a trigger queue, the Output Connection from the node to the trigger queue must be edited. By selecting the Output port, the Node Execution Parameter PULSE must be added. PULSE represents the number of trigger pulses to be placed on a trigger queue. The value entry represents the number of tokens to be transferred to the trigger queue.

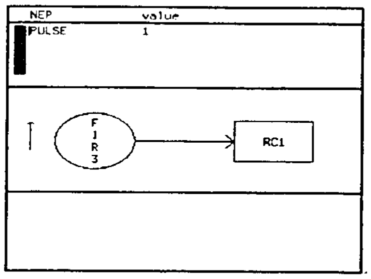


Figure 26: Output Port

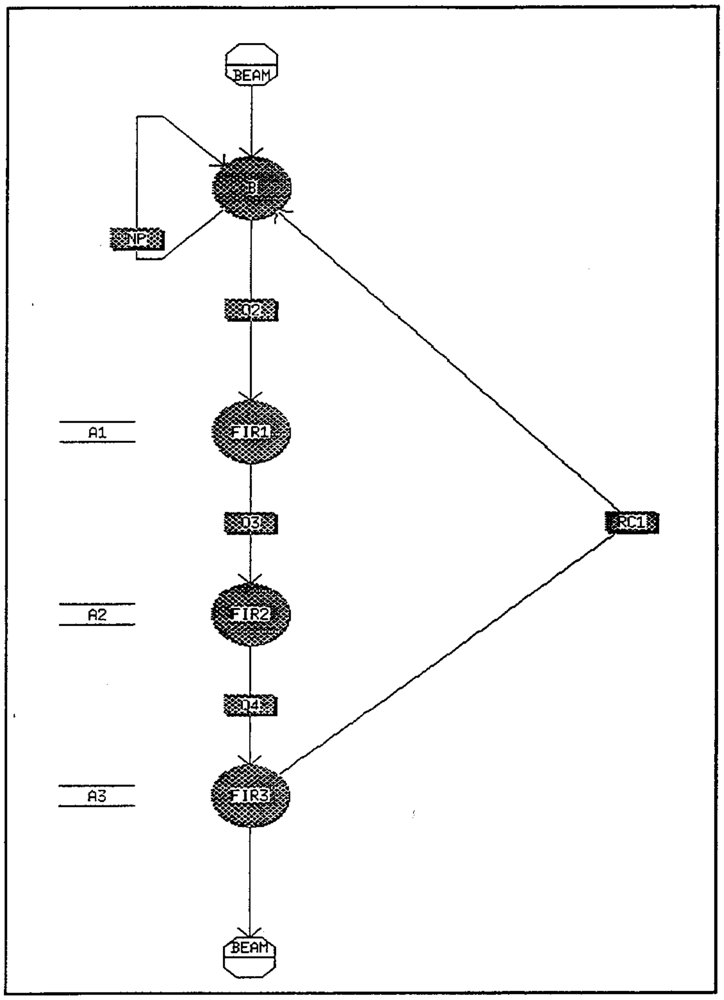


Figure 27: Restructured Graph

D. RESTRUCTURED GRAPH EXECUTIONS

To demonstrate that a trigger queue does perform as a control mechanism, the graph was executed three times with different characteristics. The Restructured Graph in Figure 27, consists of one trigger queue RC1. This trigger queue creates the capability to control the node execution, providing the technique which can be utilized to analyze and predict a graph's performance. This graph was executed using the same input files which were examined in Chapter 2. The only variables which were modified were the number of trigger tokens, and the threshold levels.

The simulation of graphs using the ets++ function, was executed for three graphs to provide analysis of the effects of the trigger queue on a graph. The Basic Sample Graph was executed to provide a benchmark with results in Figure 28. The second execution, using the trigger queue as displayed in Figure 27, was set with a threshold of zero, and two trigger tokens. The introduction of the two trigger tokens into the graph reduced the number of times the graph was executed which can be seen in Figure 29. The third execution, used a threshold of one and zero trigger tokens, which caused deadlock in the graph. Reviewing the results in Figure 30, one can see that none of the nodes were executed.

```
Node  Node                GM    times
 id   name                id  scheduled
----  ----              ---- ---------
   0  >BANDSHIFT           3    389
   1  >FIR1                4    389
   2  >FIR2                3    388
   3  >FIR3                4    387
```

Figure 28: Basic Graph

```
Node  Node                GM    times
 id   name                id  scheduled
----  ----              ---- ---------
   0  >BANDSHIFT           3    355
   1  >FIR1                4    355
   2  >FIR2                3    354
   3  >FIR3                4    353
```

Figure 29: Threshold = 0 / Trigger 2

```
Node  Node                GM    times
 id   name                id  scheduled
----  ----              ---- ---------
   0  >BANDSHIFT           3      0
   1  >FIR1                4      0
   2  >FIR2                3      0
   3  >FIR3                4      0
```

DEADLOCK

SITUATION

Figure 30: Threshold = 1 / Trigger = 0

The trigger queue technique demonstrated by the three executions of these basic graphs demonstrates how the transfer of data between nodes can be varied. This method can be utilized to implement a control process into graphs which will allow the analysis and the ability to predict data flow in graphs.

V. CONCLUSIONS AND TOPICS FOR FUTURE RESEARCH

A. CONCLUSION

As the U. S. Navy moves into the Twenty-first Century the signal processing requirements will continue to increase. With the quick reaction time needed onboard ships to counter the enemies' offensive weapons or the timely processing of data needed to compute a fire control solution, the Navy must optimize the use of the AN/UYS-2 multiprocessor. Real time and parallel systems will be needed to meet these needs, and will continue to grow in size and complexity. While scalability is one facet of a parallel program, it is not a direct measure of performance [HEN 90]. This area, performance of electronic equipment is what the U. S Navy must be concerned with.The trigger queue implementation provides a technique to predict performance, which will allow the testing of new features of the AN/UYS-2 multiprocessor before upgrades and/or new features are entered into the fleet. Compile time analysis of whether the new graph will meet the required data rate estimates for a new system will become possible. Data-flow execution can be carried out in a controlled manner using the trigger queue, thus improving predictability of the application graph.

B. TOPICS FOR FUTURE RESEARCH

Several topics for further study can be derived from this investigation. All of them are related to either improving performance or to explaining events which can be utilized to optimize the AN/UYS-2 multiprocessor.

Performance analysis studies of the trigger queues on graphs currently under development for use by the U.S. Navy should be conducted and analyzed. The current focus of ASW (Anti-Submarine Warfare) has shifted from the passive mode of prosecution of nuclear submarines to the active mode of searching for diesal submarines which should result in new procedures being developed.

The topic of scalability must be addressed and what effect executing several graphs simultaneously has on performance of the graph. This system looks at only one specific graph and tailors the hardware configuration file for that graph. Analysis must be performed on real-time scenarios encountered by U.S. Navy ASW platforms and the effect on the graph execution statistics.,

In the hardware configuration area, the interconnection structure of the AN/UYS-2, and possible additions of cache memory for processors and the effect on performance. The introduction of trigger queues greatly increased the number of system language messages onto the Data Transfer Network and the addition of cache memory may reduce this high number.

APPENDIX A. CORRELATOR GRAPH COMMAND PROGRAM

This appendix contains the Correlator Graph command program, test.cp.

```
%INITCOMPROG()

    int     STI = 4;
    float   F = 500.0;
    float   SR = 1000.0;
    float   TC = 50.0;

    IO_PROC_IDiop1,iop2,iop3,iop4;
    GRAPH_IDG;
    QUEUE_IDib[2],ob[2];

    ib[0]=%CREATEQ(FIXED);
    ib[1]=%CREATEQ(FIXED);
    ob[0]=%CREATEQ(INT);
    ob[1]=%CREATEQ(INT);
G = %START(E006
    GIP = SR,F,TC,STI
    INPUTQ = FAMILY(ib[0],ib[1])
    OUTPUTQ = FAMILY(ob[0],ob[1])
    PRIORITY = 2);

if (%SCODE) exit(1);

iop1 = %INITIO(IN1   INPUTQ = ib[0]);
iop2 = %INITIO(IN2   INPUTQ = ib[1]);
iop3 = %INITIO(OUT1   OUTPUTQ = ob[0]);
iop4 = %INITIO(OUT2   OUTPUTQ = ob[1]);
%STARTIO(iop1);
%STARTIO(iop2);
%STARTIO(iop3);
%STARTIO(iop4);

%PRINT(%TERM,1,G);

%ENDPROGRAM
```

APPENDIX B. CORRELATOR GRAPH INPUT/OUTPUT FILE

This appendix contains the Correlator Graph Input/Output file, 1.io.

```
# @(#) /b/cm/emsp/sef/sccs/b2/eo/s.eo.ioproc.cf 1.2 3/5/91

# The format of the i/o procedure file is tabular; the information
# content is simple, so this should cause no problems, and it reduces
# the amount of baggage in the simulator for interpreting this file.

# NAME is the symbolic identifier of the i/o procedure named in
# the %INITIO statement in the command program; it must match exactly;
# case is significant.
#
# TYPE is one of INPUT, BIDIRECTIONAL, or OUTPUT
#
# DATA RATES for graph inputs are expressed in words/second
#
# THRESH, READ, CONSUME, and PRODUCE amounts are in words
#
# comments look like this, everything on a line to the right
# of the sharp sign is discarded

#                               number of
#                               ports
# name          type            output
#-------        ------          -------
IN1             INPUT             1

#               rate            produce
#-----          ------          -------
                2048            2048

IN2             INPUT             1

#               rate            produce
#               ------          -------
                2048            2048
```

```
#                                  number of
#                                  ports
# name          type               input
#-----         -----        -----
OUT1           OUTPUT           1

# thresh        read            consume
#------        ------        -------
   513           513              513

OUT2            OUTPUT          1

# thresh         read             consume
#------          -----         ------
   513             513              513
```

APPENDIX C. CORRELATOR GRAPH SIMULATION FILE

This appendix contains the Correlator Graph simulation file corr.ets.

```
version 2.0


ioprocs 4

#ioproc                           iop                      input        output
#ident        name                ident  type              ports        ports
#-----        ---------------- ----- ----              -----        ------
      0       IN1                0     I                 0            1
      1       IN2                0     I                 0            1
      2       OUT1               0     O                 1            0
      3       OUT2               0     O                 1            0

graph E006 2
PIDS /users/res/ecos/temp/Ecosws/Pids/pidtoc
(1
            (E006
                        (32  ;Queues
                        (X  (1
                                (    0 (1   2 ))))
                        (GRAMOUT  (1
                                (    2 (1   2 ))))
                        FIXFLOUT1
                        BAND1OUT
                        BAND2OUT
                        FIXFLOUT2
                        FIR2OUT
                        FIR1OUT
                        ZFILOUT
                        FFT2OUT
                        FFT1OUT
                        WIND1OUT
                        WIND2OUT
                        IFFTOUT
                        PWRMULTOUT
                        SQRTOUT
                        ASQRTOUT
                        MAGOUT
                        DIVOUT
                        STIOUT
                        EAVNOUT
                        MULTOUT
                        PWR1OUT
                        PWR2OUT
                        COEFFPTR1
                        COEFFPTR2
                        EAVNFEED
                        REFVECT2
                        (REP2OUT  (1
                                    (   30 (1   2 ))))
                        (REP1OUT  (1
                                    (   32 (1   2 ))))
                        (REP3OUT  (1
                                    (   34 (1   2 ))))
                        REFVECT
                        )
                        (27
                        ; 27 Nodes
                        FIXFL1
                        FIXFL2
                        BAND1
                        FIR1
                        BAND2
                        FIR2
                        ZEROFILL
                        FFT2
                        FFT1
                        WINDOW1
                        WINDOW2
                        REPLICATE2
                        REPLICATE1
                        MULTXY
                        POWERX
                        POWERY
                        INVERSEFFT
                        MAGNITUDE
                        MULTPOWER
                        SQRT
                        CHANGE
                        NORMALIZE
                        INTEGRATE
                        REPLICATE3
                        GRAM
                        EXPAVG
                        ASCAN
                        )
                        ; 0 Vars
            )
    )
```

```
nodes 27

#node                               is   exec    input output
#ident [index]name       class type size cycles  ports  ports   primitive
#----- ---------------- ---------------------------------------------
      0 FIXFL1           1        P   256  6174   1       1  DMC_FXFL
      1 FIXFL2           1        P   256  6174   1       1  DMC_FXFL
      2 BAND1            1        P   256  16492  2       2  CDM_RVF
      3 FIR1             1        P   256  67781  1       1  FIR_C2S
      4 BAND2            1        P   256  16492  2       2  CDM_RVF
      5 FIR2             1        P   256  67781  1       1  FIR_C2S
      6 ZEROFILL         1        P   256   8244  1       1  DFC_REORD
      7 FFT2             1        P   256  21403  1       1  FFT_CC
      8 FFT1             1        P   256  21403  1       1  FFT_CC
      9 WINDOW1          1        P   256   5193  1       1  DCP_HAMN
     10 WINDOW2          1        P   256   5193  1       1  DCP_HAMN
     11 REPLICATE2       1        P   256  0      1       2  DFC_REP
     12 REPLICATE1       1        P   256  0      1       2  DFC_REP
     13 MULTXY           1        P    256  7206  2       1  VCC_VMUL
     14 POWERX           1        P    256  3105  1       1  VOC_PWR
     15 POWERY           1        P    256  3105  1       1  VOC_PWR
     16 INVERSEFFT       1        P    256  23507 1       1  FFT_CC
     17 MAGNITUDE        1        P    256  2099  1       1  DCP_CSMG
     18 MULTPOWER        1        P    256  52    2       1  VRR_VMUL
     19 SQRT             1        P    256  70    1       1  VOR_VSQR
     20 CHANGE           1        P    256  0     1       1  DMC_EMC
     21 NORMALIZE        1        P    256  5180  2       1  VRR_VDIV
     22 INTEGRATE        1        P    256  7031  1       1  DCP_STI
     23 REPLICATE3       1        P    256  0     1       2  DFC_REP
     24 GRAM             1        P    256  9271  2       1  DFC_REQ
     25 EXPAVG           1        P    256  3821  2       2  DCP_EAVN
     26 ASCAN            1        P    256  9271  2       1  DFC_REQ

queues 37

#queue            init elem  src  src   prod sink  sink thresh read cons   lm
#[dims]name type elem size  node port  amt  node  port amt    amount amt  type
#--------------- ---------------------------------------------------------
[1]X          I     0     1     0     0      0     0      0   16384  16384  16384 OM
[2]X          I     0     1     1     0      0     1      0   16384  16384  16384 OM
[1]GRAMOUT    O     0     1    24     0    513     2     0       0      0      0 ####
[2]GRAMOUT    O     0     1    26     0    513     3     0       0      0      0 ####
FIXFLOUT1     S     0     2     0     0 16384     2     1   16384  16384   16384 OM
BAND1OUT      S    39     4     2     0 16384     3     0   16423  16423   16384 OM
BAND2OUT      S    39     4     4     0 16384     5     0   16423  16423   16384 OM
FIXFLOUT2     S     0     2     1     0 16384     4     1   16384  16384   16384 OM
FIR2OUT       S     0     4     5     0   4096     6     0    4096   4096    4096 OM
FIR1OUT       S     0     4     3     0   4096     8     0    4096   4096    4096 OM
ZFILOUT       S     0     4     6     0   4096     7     0    4096   4096    4096 OM
FFT2OUT       S     0     4     7     0   4096    10     0    4096   4096    4096 OM
FFT1OUT       S     0     4     8     0   4096     9     0    4096   4096    4096 OM
WIND1OUT      S     0     4     9     0   4096    12     0    4096   4096    4096 OM
```

```
WIND2OUT     S    0   4   10   0  4096  11   0  4096  4096  4096    OM
IFFTOUT      S    0   4   16   0  2052  17   0  2052  2052  2052    OM
PWRMULTOUT   S    0   2   18   0     4  19   0     4     4     4    OM
SQRTOUT      S    0   2   19   0     1  20   0     4     4     4    OM
ASQRTOUT     S    0   2   20   0     1  21   1     1     1     1    OM
MAGOUT       S    0   2   17   0  2052  21   0   513   513   513    OM
DIVOUT       S    0   2   21   0   513  22   0  2052  2052  2052    OM
STIOUT       S    0   2   22   0   513  23   0   513   513   513    OM
EAVNOUT      S    0   2   25   0   513  26   0   513   513   513    OM
MULTOUT      S    0   4   13   0  4096  16   0  4096  4096  4096    OM
PWR1OUT      S    0   2   14   0     4  18   1     4     4     4    OM
PWR2OUT      S    0   2   15   0     4  18   0     4     4     4    OM
COEFFPTR1    S    1   1    2   1     1   2   0     1     1     1  CULM
COEFFPTR2    S    1   1    4   1     1   4   0     1     1     1  CULM
EAVNFEED     S  513   2   25   1   513  25   0   513   513   513    OM
REFVECT2     S  513   2   -1   0     0  26   1   513   513     0    OM
[1]REP2OUT   S    0   4   11   0  4096  13   1  4096     1     1    OM
[2]REP2OUT   S    0   4   11   1  4096  15   0  4096  4096  4096    OM
[1]REP1OUT   S    0   4   12   0  4096  13   0  4096  4096  4096    OM
[2]REP1OUT   S    0   4   12   1  4096  14   0  4096  4096  4096    OM
[1]REP3OUT   S    0   2   23   0   513  24   0   513   513   513    OM
[2]REP3OUT   S    0   2   23   1   513  25   1   513   513   513    OM
REFVECT      S  513   2   -1   0     0  24   1   513   513     0    OM

graph vars 0
#----- end of E006 -----

connect 0
```

APPENDIX D. CORRELATOR GRAPH gas OUTPUT

This appendix contains the Correlator Graph gas output produced by using the simulation file corr.ets and the Input/Output file 1.io as input.

```
Static Graph Analysis - EWS Release: 5.6

Invocation: gas -g corr.ets -i 1.io -n all -q all -o corr.gout

total graph objects: 27 nodes, 37 queues, 0 graphvars, 4 I/O procedures

graph: E006
pids: /users/res/ecos/temp/Ecosws/Pids/pidtoc
graph objects: 27 nodes[0,26], 37 queues[0,36], 0 graphvars, 4 I/O procedures[0,3]

graph source: corr.ets
i/o procedure source: 1.io

 total node scheduling rate:         3.44 / second
 AP node scheduling rate:            3.19 / second
 total required AU cycles:           3.90e+04 cycles / second
 mean AU cycles per node:            1.22e+04 cycles
 mean queue consumes / node:          1.28 CQ    / node
 mean queue consume amount:      13048.27 words / CQ
 mean queue reads / node:             1.28 RQ    / node
 mean queue read amount:         13120.46 words / RQ
 mean queue writes / node:            1.24 WQ    / node
 mean queue write amount:        13078.38 words / WQ
 mean graph variable reads:           0.00 RGV   / node
 mean gv read amount:                 0.00 words / RGV
 mean graph variable writes:          0.00 WGV   / node
 mean gv write amount:                0.00 words / WGV
 mean queue+gv read amount:      13120.46 words / (RQ,RGV)
 mean queue+gv write amount:     13078.38 words / (WQ,WGV)
 mean AIS size:                     256.00 words / AIS
 Bandwidth AP:                        1.05e+05 words / second
 Bandwidth IOP:                       4.22e+03 words / second
 Bandwidth GM:                        1.09e+05 words / second

 2 nodes with inconsistent scheduling rate:
```

```
E006>MULTXY (id=13) (min,max) = (1.25e-01,5.12e+02) / sec
E006>NORMALIZE (id=21) (min,max) = (3.12e-02,5.00e-01) / sec


Individual node statistics
Node Node            rate       rate      ------- number of -------        i/o rate
 id  name         (exec/sec) (cycl/sec) RQs WQs CQs RGVs WGVs (word/sec)
 ----  ----        ----------  ----------  ---  ---  ---  ----  ----  ----------
  0 E006>FIXFL1        .125     772      1    1    1    0    0    6.14e3
  1 E006>FIXFL2        .125     772      1    1    1    0    0    6.14e3
  2 E006>BAND1         .125     2.06e3   2    2    2    0    0    12.3e3
  3 E006>FIR1          .125     8.47e3   1    1    1    0    0   10.3e3
  4 E006>BAND2         .125     2.06e3   2    2    2    0    0   12.3e3
  5 E006>FIR2          .125     8.47e3   1    1    1    0    0   10.3e3
  6 E006>ZEROFILL      .125     1.03e3   1    1    1    0    0    4.1e3
  7 E006>FFT2          .125     2.68e3   1    1    1    0    0    4.1e3
  8 E006>FFT1          .125     2.68e3   1    1    1    0    0    4.1e3
  9 E006>WINDOW1       .125     649      1    1    1    0    0    4.1e3
 10 E006>WINDOW2       .125     649      1    1    1    0    0   4.1e3
 11 E006>REPLICATE2 .125     0        1    2    1    0    0   6.14e3
 12 E006>REPLICATE1 .125     0        1    2    1    0    0   6.14e3
 13 E006>MULTXY        .125     901      2    1    2    0    0    4.1e3
 14 E006>POWERX        .125     388      1    1    1    0    0   2.05e3
 15 E006>POWERY        .125     388      1    1    1    0    0   2.05e3
 16 E006>INVERSEFFT .125     2.94e3   1    1    1    0    0   3.07e3
 17 E006>MAGNITUDE .125     262      1    1    1    0    0   1.54e3
 18 E006>MULTPOWER .125     6.5      2    1    2    0    0   3
 19 E006>SQRT          .125     8.75     1    1    1    0    0   1.25
 20 E006>CHANGE      0.0312   0        1    1    1    0    0   .312
 21 E006>NORMALIZE 0.0312  162      2    1    2    0    0   64.2
 22 E006>INTEGRATE    .125     879      1    1    1    0    0   641
 23 E006>REPLICATE3   .125     0        1    2    1    0    0   385
 24 E006>GRAM          .125     1.16e3   2    1    2    0    0   321
 25 E006>EXPAVG        .125     478      2    2    2    0    0   513
 26 E006>ASCAN         .125     1.16e3   2    1    2    0    0   321
```

APPENDIX E. CORRELATOR GRAPH MACHINE CONFIGURATION FILE

This appendix contains the Correlator Graph machine configuration file 4-6.cf.

```
CONFIGURATION
#
# Machine configuration file: /users/res/ecos/temp/Ecosws/0graph/4-6.cf
# Produced by med at: Sat Aug 20 10:55:05 1994

#
# number of functional elements of each type
#
#  type    quantity
#  ----    --------
   AP      4
   GM      6
   IOP     2
#
# number of concentrators/distributors on DTN
#
   DIST    8
   CONC    8

HARDWARE
#  type    name    did    cid    class
#  ----    ----    ---    ---    -----
   AP      AP:0    0      0      1
   AP      AP:1    1      1      1
   AP      AP:2    2      2      1
   AP      AP:3    3      3      1
   GM      GM:0    4      4
   GM      GM:1    5      5
   GM      GM:2    6      6
   IOP     IOP:0   7      7
   GM      GM:3    4      4
   GM      GM:4    5      5
   GM      GM:5    6      6
   IOP     IOP:1   7      7
```

APPENDIX F. CORRELATOR GRAPH ets++ OUTPUT

This appendix contains the Correlator Graph ets++ output file produced by utilizing the simulation file corr.ets, the Input/Output file 1.io, and the Machine Configuration file 4-6.cf.

```
EMSP Event Time Simulator - EWS Release: 5.6

Invocation: ets++ -g corr.ets -i 1.io -m 4-6.cf -t 10 -n all -q all

Machine parameter file: '/users/res/ecos/temp/Ecosws/lib/params.d' of 06/18/93
09:12

Configuration
-------------
FE count:  13
AP count:  4
GM count:  6
IOP count: 2
concentrators: 8
distributors: 8

total graph objects: 27 nodes, 37 queues, 0 graphvars, 4 I/O procedures

graph: E006
pids: /users/res/ecos/temp/Ecosws/Pids/pidtoc
graph objects: 27 nodes[0,26], 37 queues[0,36], 0 graphvars, 4 I/O procedures[0,3]


Statistics
----------

Simulation clock = 10.000 seconds
```

```
Functional Element Utilization Statistics

FEID: 0  NAME: AP:0          TYPE: arithmetic processor
conc id: 0   dist id: 0
AP Class: 1
 CU idle =  99.95 %
 CU wait =   0.00 %
 CU busy =   0.05 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 2.43e-06 sec, max = 2.76e-05 sec
 cbus messages sent:   17    received:    0
 dtn  messages sent:    5    received:   11
AG/AU nodes executed: 5  (CU only: 0)
AG/AU busy: 0.14 %
AG/AU pending: 0.14 %
AG/AU efficiency: 49.50 %
Free time (AP:0): 99.86 %


FEID: 1  NAME: AP:1          TYPE: arithmetic processor
conc id: 1   dist id: 1
AP Class: 1
 CU idle =  99.96 %
 CU wait =   0.00 %
 CU busy =   0.04 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:   15    received:    0
 dtn  messages sent:    6    received:   10
AG/AU nodes executed: 4  (CU only: 1)
AG/AU busy: 0.15 %
AG/AU pending: 0.15 %
AG/AU efficiency: 48.83 %
Free time (AP:1): 99.85 %
```

```
FEID:  2  NAME: AP:2            TYPE: arithmetic processor
conc id:  2   dist id:  2
AP Class: 1
 CU idle =  99.95 %
 CU wait =   0.00 %
 CU busy =   0.05 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 7.29e-06 sec, max = 3.33e-05 sec
 cbus messages sent:    19    received:    0
 dtn  messages sent:     7    received:   12
AG/AU nodes executed: 4  (CU only: 1)
AG/AU busy: 0.06 %
AG/AU pending: 0.11 %
AG/AU efficiency: 36.43 %
Free time (AP:2): 99.89 %

FEID:  3  NAME: AP:3            TYPE: arithmetic processor
conc id:  3   dist id:  3
AP Class: 1
 CU idle =  99.95 %
 CU wait =   0.00 %
 CU busy =   0.05 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 2.43e-06 sec, max = 2.76e-05 sec
 cbus messages sent:    17    received:    0
 dtn  messages sent:     6    received:   11
AG/AU nodes executed: 5  (CU only: 0)
AG/AU busy: 0.05 %
AG/AU pending: 0.12 %
AG/AU efficiency: 29.28 %
Free time (AP:3): 99.88 %

FEID:  4  NAME: GM:0            TYPE: global memory
conc id:  4   dist id:  4
 CU idle =  99.94 %
 CU wait =   0.00 %
 CU busy =   0.06 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:     4    received:   12
 dtn  messages sent:     8    received:   12
GM Utilization (busy + dtn i/o):   0.24 %
memory usage, high water mark: 79579;  pointer block: 336, static: 12203
```

```
   Queue table:   224     GI table:  3000
    Node table:   384   Queue Hist:     0
      GV table:     0   Templates:  7020 (appr)
     GV memory:     0       delta:     0 (assm)
  Initial data:  1568    CTC table:     7
Nodes: 6   Queues: 7   Graph Variables: 0

FEID:  5  NAME: GM:1            TYPE: global memory
conc id:  5   dist id:  5
 CU idle =  99.95 %
 CU wait =   0.01 %
 CU busy =   0.04 %
 service wait per cbus request:  avg = 5.71e-07 sec, max = 7.71e-06 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:    5   received:    13
 dtn  messages sent:    8   received:     5
GM Utilization (busy + dtn i/o):   0.14 %
memory usage, high water mark: 43963;  pointer block: 176, static: 10779
   Queue table:   224     GI table:  3000
    Node table:   256   Queue Hist:     0
      GV table:     0   Templates:  7020 (appr)
     GV memory:     0       delta:     0 (assm)
  Initial data:   272    CTC table:     7
Nodes: 4   Queues: 7   Graph Variables: 0

FEID:  6  NAME: GM:2            TYPE: global memory
conc id:  6   dist id:  6
 CU idle =  99.95 %
 CU wait =   0.00 %
 CU busy =   0.05 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:    4   received:    12
 dtn  messages sent:    8   received:    12
GM Utilization (busy + dtn i/o):   0.15 %
memory usage, high water mark: 26969;  pointer block: 80, static: 10505
   Queue table:   160     GI table:  3000
    Node table:   320   Queue Hist:     0
      GV table:     0   Templates:  7020 (appr)
     GV memory:     0       delta:     0 (assm)
  Initial data:     0    CTC table:     5
Nodes: 5   Queues: 5   Graph Variables: 0
```

```
FEID:  7  NAME: GM:3            TYPE: global memory
conc id:  4   dist id:  4
 CU idle =  99.96 %
 CU wait =   0.00 %
 CU busy =   0.04 %
 service wait per cbus request:  avg = 7.14e-07 sec, max = 7.71e-06 sec
 service wait per dtn  request:  avg = 7.23e-05 sec, max = 3.62e-04 sec
 cbus messages sent:     5    received:    10
 dtn  messages sent:     6    received:     5
GM Utilization (busy + dtn i/o):   0.16 %
memory usage, high water mark: 46507;  pointer block: 224, static: 12011
   Queue table:    224      GI table:   3000
    Node table:    192    Queue Hist:      0
      GV table:      0    Templates:   7020 (appr)
     GV memory:      0        delta:      0 (assm)
  Initial data:   1568    CTC table:      7
Nodes: 3    Queues: 7    Graph Variables: 0


FEID:  8  NAME: GM:4            TYPE: global memory
conc id:  5   dist id:  5
 CU idle =  99.95 %
 CU wait =   0.00 %
 CU busy =   0.05 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:     4    received:    12
 dtn  messages sent:     8    received:     4
GM Utilization (busy + dtn i/o):   0.23 %
memory usage, high water mark: 79482;  pointer block: 336, static: 12106
   Queue table:    192      GI table:   3000
    Node table:    320    Queue Hist:      0
      GV table:      0    Templates:   7020 (appr)
     GV memory:      0        delta:      0 (assm)
  Initial data:   1568    CTC table:      6
Nodes: 5    Queues: 6    Graph Variables: 0


FEID:  9  NAME: GM:5            TYPE: global memory
conc id:  6   dist id:  6
 CU idle =  99.97 %
 CU wait =   0.00 %
 CU busy =   0.03 %
 service wait per cbus request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 service wait per dtn  request:  avg = 6.99e-05 sec, max = 2.79e-04 sec
 cbus messages sent:    3    received:    9
 dtn  messages sent:    6    received:    4
GM Utilization (busy + dtn i/o):   0.12 %
memory usage, high water mark: 27177;  pointer block: 96, static: 10441
   Queue table:    160     GI table:   3000
    Node table:    256    Queue Hist:      0
      GV table:      0    Templates:   7020 (appr)
     GV memory:      0        delta:      0 (assm)
  Initial data:      0    CTC table:      5
Nodes: 4    Queues: 5    Graph Variables: 0


FEID: 10  NAME: IOP:0            TYPE: input/output processor
conc id:  7   dist id:  7
Simple IOP model
 cbus messages sent:    0    received:    1
 dtn  messages sent:   18    received:    0
  Number of WQs completed: 18 cancelled by SNDTs: 0
Input queue:   0,  input rate (w/sec): 2.05e+03
Input queue:   1,  input rate (w/sec): 2.05e+03
Output nodes executed: 0


FEID: 11  NAME: IOP:1            TYPE: input/output processor
conc id:  7   dist id:  7
Simple IOP model
 cbus messages sent:    0    received:    1
 dtn  messages sent:    0    received:    0
  Number of WQs completed: 0 cancelled by SNDTs: 0
Output nodes executed: 0


FEID: 12  NAME: SCH              TYPE: scheduler
 CU idle =  99.98 %
 CU wait =   0.00 %
 CU busy =   0.02 %
 service wait per cbus request:  avg = 5.71e-06 sec, max = 7.70e-05 sec
 service wait per dtn  request:  avg = 0.00e+00 sec, max = 0.00e+00 sec
 cbus messages sent:   20    received:   50
 dtn  messages sent:    0    received:    0
```

```
CBUS Utilization Statistics

CBUS message count = 120
idle = 100.00 %
busy = 0.00 %

DTN Utilization Statistics
DTN message count = 86
Conc 0: idle =  99.95 %  busy =  0.05 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 1: idle =  99.92 %  busy =  0.08 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 2: idle =  99.91 %  busy =  0.09 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 3: idle =  99.93 %  busy =  0.07 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 4: idle =  99.85 %  busy =  0.15 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 5: idle =  99.85 %  busy =  0.15 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 6: idle =  99.90 %  busy =  0.10 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Conc 7: idle =  99.97 %  busy =  0.03 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec

Dist 0: idle =  99.90 %  busy =  0.10 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 1: idle =  99.88 %  busy =  0.12 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 2: idle =  99.92 %  busy =  0.08 %  avg. wait = 1.10e-04 sec  max wait = 1.32e-03 sec
Dist 3: idle =  99.92 %  busy =  0.08 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 4: idle =  99.88 %  busy =  0.12 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 5: idle =  99.88 %  busy =  0.12 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 6: idle =  99.92 %  busy =  0.08 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec
Dist 7: idle = 100.00 %  busy =  0.00 %  avg. wait = 0.00e+00 sec  max wait = 0.00e+00 sec

DTN input fifo statistics
   AP:0 (id =  0) PIP: max. messages =  1,  high water mark = 410
   AP:1 (id =  1) PIP: max. messages =  1,  high water mark = 410
   AP:2 (id =  2) PIP: max. messages =  2,  high water mark = 721
   AP:3 (id =  3) PIP: max. messages =  1,  high water mark = 410
   GM:0 (id =  4) PIP: max. messages =  1,  high water mark = 16689
   GM:1 (id =  5) PIP: max. messages =  1,  high water mark = 8633
   GM:2 (id =  6) PIP: max. messages =  1,  high water mark = 4605
   GM:3 (id =  7) PIP: max. messages =  1,  high water mark = 8633
   GM:4 (id =  8) PIP: max. messages =  1,  high water mark = 16689
   GM:5 (id =  9) PIP: max. messages =  1,  high water mark = 4605
  IOP:0 (id = 10) PIP: max. messages =  0,  high water mark = 0
  IOP:1 (id = 11) PIP: max. messages =  0,  high water mark = 0


Node Scheduling Statistics
nodes scheduled = 20
rate = 2.0  nodes/sec
```

```
Cumulative SL message statistics
 RQ          24
 AQ          24
 WQ          42
 CQ          24
 RGV          0
 AGV          0
 WGV          0
 QOT         30
 QUT          0
 QOC          0
 QUC          0
 RFIS        20
 SIS         20
 AIS         20
 EIS/ESN      0
 SNDT         0
 CNDT         0
 CTC          0

 TOTAL      204

Graph Object Statistics

Node Statistics
----    ----------
                                          times    average     average     average
  Node  Node           GM    times        waited   wait        delay       brkdown
    id  name           id    scheduled    for AP   for AP      to exec     delay
  ----  ----           ----  ---------    ------   --------    -------     -------
     0  >FIXFL1         4      1            0      0.00e+00    2.18e-03    2.67e-03
     1  >FIXFL2         6      1            0      0.00e+00    2.18e-03    2.67e-03
     2  >BAND1          5      1            0      0.00e+00    3.79e-03    5.25e-03
     3  >FIR1           8      1            0      0.00e+00    6.56e-03    1.50e-03
     4  >BAND2          7      1            0      0.00e+00    3.80e-03    5.25e-03
     5  >FIR2           4      1            0      0.00e+00    6.56e-03    1.50e-03
     6  >ZEROFILL       5      1            0      0.00e+00    2.18e-03    1.50e-03
     7  >FFT2           6      1            0      0.00e+00    2.18e-03    1.50e-03
     8  >FFT1           9      1            0      0.00e+00    2.18e-03    1.50e-03
     9  >WINDOW1        8      1            0      0.00e+00    2.18e-03    1.50e-03
    10  >WINDOW2        7      1            0      0.00e+00    2.18e-03    1.50e-03
    11  >REPLICATE2     4      1            0      0.00e+00    0.00e+00    4.98e-0
    12  >REPLICATE1     9      1            0      0.00e+00    0.00e+00    4.98e-03
```

```
13 >MULTXY       5   1    0   0.00e+00  2.27e-03  1.60e-03
14 >POWERX       8   1    0   0.00e+00  2.18e-03  3.27e-04
15 >POWERY       6   1    0   0.00e+00  2.18e-03  3.27e-04
16 >INVERSEFFT   9   1    0   0.00e+00  2.19e-03  9.13e-04
17 >MAGNITUDE    6   1    0   0.00e+00  1.45e-03  6.19e-04
18 >MULTPOWER    4   1    0   0.00e+00  9.16e-04  4.32e-04
19 >SQRT         8   1    0   0.00e+00  7.55e-04  3.26e-04
20 >CHANGE       9   0    0   0.00e+00  0.00e+00  0.00e+00
21 >NORMALIZE    4   0    0   0.00e+00  0.00e+00  0.00e+00
22 >INTEGRATE    5   0    0   0.00e+00  0.00e+00  0.00e+00
23 >REPLICATE3   6   0    0   0.00e+00  0.00e+00  0.00e+00
24 >GRAM         7   0    0   0.00e+00  0.00e+00  0.00e+00
25 >EXPAVG       8   0    0   0.00e+00  0.00e+00  0.00e+00
26 >ASCAN        4   0    0   0.00e+00  0.00e+00  0.00e+00

Queue Statistics
----- ----------

                                       remaining
 Queue  Queue            GM   times    words
    id  name             id   written   in queue
 -----  -----     ----  -------  ---------
    0 >[1]X               4     9      2048
    1 >[2]X               6     9      2048
    2 >[1]GRAMOUT         9     0         0
    3 >[2]GRAMOUT         5     0         0
    4 >FIXFLOUT1          5     1         0
    5 >BAND1OUT           8     1       156
    6 >BAND2OUT           4     1       156
    7 >FIXFLOUT2          7     1         0
    8 >FIR2OUT            5     1         0
    9 >FIR1OUT            9     1         0
   10 >ZFILOUT            6     1         0
   11 >FFT2OUT            7     1         0
   12 >FFT1OUT            8     1         0
   13 >WIND1OUT           9     1         0
   14 >WIND2OUT           4     1         0
   15 >IFFTOUT            6     1         0
   16 >PWRMULTOUT         8     1         0
   17 >SQRTOUT            9     1         2
   18 >ASQRTOUT           4     0         0
   19 >MAGOUT             7     1      4104
   20 >DIVOUT             5     0         0
```

```
21 >STIOUT          6   0        0
22 >EAVNOUT         4   0        0
23 >MULTOUT         9   1        0
24 >PWR1OUT         4   1        0
25 >PWR2OUT         5   1        0
26 >COEFFPTR1       5   1        1
27 >COEFFPTR2       7   1        1
28 >EAVNFEED        8   0     1026
29 >REFVECT2        4   0     1026
30 >[1]REP2OUT      5   1    16380
31 >[2]REP2OUT      6   1        0
32 >[1]REP1OUT      7   1        0
33 >[2]REP1OUT      8   1        0
34 >[1]REP3OUT      7   0        0
35 >[2]REP3OUT      8   0        0
36 >REFVECT         7   0     1026
```

LIST OF REFERENCES

[ATT 93] AT&T Technologies, AN/UYS-2A(V) ASIP Users' Manual, Application Programmers Users' Manual Vol 1-3, AT&T Bell Laboratories, 28 June 1993.

[ATT 88] AT&T Technologies, Design and Implementation of an EMSP System Simulator, AT&T Bell Laboratories, 5 August 1988.

[ATT 92] AT&T Technologies, ECOS Workstation Release 5.5 User Manual, AT&T Bell Laboratories, 1 April 1992.

[HAM 92] Hammond, S.W., Mapping Unstructured Grid Computations to Massively Parallel Computers, Doctoral thesis, Rensselaer Polytechnic Institute, Troy, New York, February 1992.

[HEN 90] Hennessy, J.L., and Patterson, D.A., "Computer Architecture, A Quantitative Approach," pp. 585, Morgan Kaufmann Publishers, 1990.

[HUG 86] Hughs, W.P., "Fleet Tactics, theory and practice," pp.252-253, Naval Institute Press, 1992.

[LEW 92] Lewis, T.G., and El-Rewini, H., "Introduction to Parallel Computing," pp. 29-30, Prentice-Hall, 1992.

[LIT 91] Little, B.S., A Technique for Predictable Real-Time Execution in the AN/UYS-2 Parallel Signal Processing Architecture, Master's thesis, Naval Postgraduate School, Monterey, California, December 1991.

[NRL 1] Naval Research Laboratory, Implementation of the Processing Graph Method (PGM) on the EMSP with the Floating Point Arithmetic Processor (FPAP) Specification, 15 December 1987.

[NRL 90] Naval Research Laboratory, Processing Graph Method Tutorial, 8 January 1990.

[RIC 90] Rice, M.L., "The Navy's New Standard Digital Signal Processor: The AN/UYS-2," paper presented at the Association of Scientists and Engineers 27th Annual Technical Symposium, 23 May 1990.

[ROS 91] Rosen, K.H., "Discrete Mathematics and its Applications," pp. 423-424, McGraw-Hill, 1991.

[SWK 93] Swank, D.P., Large Grain Data-Flow Graph Restructuring for EMSP Signal Processing Benchmarks on the ECOS Workstation System, Master's thesis, Naval Postgraduate School, Monterey, California, June 1993.

[VAN 91] van Roermund, A.H.M., Architectures for Real-Time Video, Elsevier Science Publishers B.V., 1991.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Code 052
   Naval Postgraduate School
   Monterey, CA 93943

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Professor Amr Zaky, Code CS/Za 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Professor Shridhar Shukla, Code EC/Sh 2
   Electrical and Computer Engineering Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. LT Richard T. Keys 1
   6061 General Diaz
   New Orleans, LA 70124