1

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

137

0-315-12105-X

National Library
of CanadaBibliothèque nationale
du Canada

Canadian Theses Division

Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

60018

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

DAVID WILLIAM CRAIG

Date of Birth — Date de naissance

18 SEPTEMBER 1955

Country of Birth — Lieu de naissance

CANADA

Permanent Address — Résidence fixe

3049 EARL GREY ST.
VICTORIA, B.C.
V9A-1W8

Title of Thesis — Titre de la thèse

"FUNCTION ALLOCATION AND DATA DIVISION FOR
FREQUENCY DOMAIN BEAMFORMING ON A
MULTIPLE PROCESSOR"

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF ENGINEERING

Year this degree conferred — Année d'obtention de ce grade

1983

Name of Supervisor — Nom du directeur de thèse

C.M. WOODSIDE

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies of
the film.The author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permission.L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et de
prêter ou de vendre des exemplaires du filmL'auteur se réserve les autres droits de publication, ni la thèse
ni de longs extraits de celle-ci ne doivent être imprimés ou
autrement reproduits sans l'autorisation écrite de l'auteur.

Date

10 NOVEMBER 1982

Signature

DW Craig

FUNCTION ALLOCATION AND DATA DIVISION FOR FREQUENCY
DOMAIN BEAMFORMING ON A MULTIPLE PROCESSOR

by

© David W. Craig

Submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

November 1982

The undersigned, recommends to the Faculty of Graduate Studies and Research, acceptance of the thesis

"FUNCTION ALLOCATION AND DATA DIVISION FOR FREQUENCY DOMAIN
BEAMFORMING ON A MULTIPLE PROCESSOR"

by David W. Craig, in partial fulfillment of the requirements for
the degree of Master of Engineering.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering

November 1982

ABSTRACT

A methodology is described for the allocation and scheduling of deterministic signal processing tasks on a multiple processor. The effect of equal versus unequal data unit division, pipeline versus uniform function allocation, width and number of buses is investigated for frequency domain beamforming and a radar signal processing example. Timing information is generated for both function execution and data transfers, so that bus congestion and waiting time could be studied. An heuristic improvement technique is used to reassign processing and tune the response, with an improvement of up to 35% in the cycle time.

ACKNOWLEDGEMENTS

I would like to thank Professor C.M. Woodside for his advice, encouragement and support during the preparation of this thesis. His patience and guidance were greatly appreciated.

Financial assistance from the Department of National Defense and educational leave from the Defense Research Establishment Pacific made this work possible.

TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENTS	ii
Table of Contents	iii
List of Figures	v
List of Tables	vi
 Chapter 1 - Introduction	 1
1.0 Motivation	1
1.1 Previous Work	2
1.2 Present Work	5
 Chapter 2 - A Frequency Domain Beamforming Task	 7
2.0 Objective	7
2.1 Notation and Problem Statement	7
2.2 Notation Summary	16
2.3 Frequency Domain Beamforming Task	18
2.3.1 Data Input and Buffering	22
2.3.2 f_1 : Fourier Transformation of Sensor Inputs	23
2.3.3 f_1 : Traceforming of Frequency Cells	24
2.3.4 f_2 : Indirect Method of Traceforming	26
2.3.5 f_3 : Beamforming from Trace Values	28
2.4 Summary of Beamforming Task	29
 Chapter 3 - Function Allocation and Data Scheduling	 32
3.0 Introduction	32
3.1 Timing Chart Example	33
3.2 "Formula" Approach to Determine Cycle Time	37
3.3 Gannt - A Fortran77 Program for Analysis	39
3.3.1 Data Transfer Rules	39
3.3.2 Assignment of Data Units	40
3.3.3 Reassignment of Data Units	42
3.3.4 Structure of Program Gannt	43
3.4 Example of an Interactive Session	47
3.5 Summary	47

Chapter 4 - Implementation of Frequency Domain Beamforming on a Multiprocessor	52
4.0 Scope of Investigation	52
4.1 Task Description	53
4.2 Minimum Cycle Time for Uniform and Pipeline Allocation	54
4.2.1 Uniform Allocation	56
4.2.2 Pipeline Allocation	57
4.3 Implementation with Uniform Allocation	58
4.4 Implementation with Pipeline Allocation	63
4.5 Comparison of Uniform and Pipeline Allocation	65
Chapter 5 - Comparison of a Pulse Doppler Radar Task on a Multiprocessor	67
5.0 Reason for Comparison	67
5.1 Task Description	67
5.2 Task Implementation with Uniform Allocation of Functions	68
5.3 Cycle Time Improvement with Data Scheduling for Uniform Allocation	72
5.4 Comparison of Improved Uniform and Pipeline Allocation	75
Chapter 6: Conclusions	80
References	83
Appendix	

LIST OF FIGURES

2.1	Topologies considered for a distributed signal processor	9
2.2	Assignment of L_1 data units to k PE's	12
2.3	A data set and its transpose	14
2.4	Typical transfers for normal and cornerturning exchanges	17
2.5	An arriving wavefront and alias impinging on a line array	19
2.6	Functional diagram for the task of frequency domain beamforming	21
2.7	Transfer between Fourier transform f_1 and traceforming f_2 for a band of 4 frequency cells	25
3.1	Data flow for a signal processing task example	35
3.2	Timing diagram for a signal processing task example	36
3.3	Effect of bus transfer size on the assignment of data units	41
3.4	An overview of the modules for program GANNT	44
3.5	Example of an interactive session with program GANNT	48
4.1	Uniform function allocation for the beamforming task with 3 PE's	54
4.2	Pipeline function allocation for the beamforming task with 1 PE for f_1 and 2 PE's for f_2, f_3	54
4.3	Effective number of PE's for the beamforming task on a multiprocessor with a double and single width bus	60
4.4	Percent PE utilization for the beamforming task on a multiprocessor with a double and single width bus	61
4.5	Percent bus utilization for the beamforming task on a multiprocessor with a double and single width bus	62
4.6	Effective number of PE's for pipeline allocation, single width bus, two buses used	64
4.7	Effective number of PE's for uniform versus pipeline allocation	66
5.1	Effective number of PE's for the radar task with one or two buses required	69
5.2	Percent PE utilization for the radar task with one and two buses	70
5.3	Percent bus utilization for the radar task with one and two buses	71
5.4	Effective number of PE's for the radar task with data scheduling for one and two buses	73
5.5	Percent PE utilization for the radar task with data scheduling for one and two buses	74
5.6	Timing diagram for uniform function allocation with 8 PE's for the radar task	76
5.7	Timing diagram for uniform function allocation and data scheduling with 8 PE's for the radar task	77
5.8	Timing diagram for pipeline function allocation and data scheduling with 8 PE's for the radar task	78

LIST OF TABLES

2.1	A comparison of cycle counts for direct and indirect methods of traceforming	27
2.2	Functional requirements for the beamforming task	30
3.1	Functional requirements for a signal processing task example	34

Chapter 1

INTRODUCTION

1.0 Motivation

There is a continuing requirement for larger throughputs for real time signal processing systems. Real time processing requires that the processing speed match or exceed the input data rate. Examples of real time processing occur in sonar and radar systems, where data are input from an array of distributed sensors [1-3]. This present work grew out of a requirement to design a signal processing system operating in real time for a passive sonar line array. Existing processing systems based on a uniprocessor design were inadequate to cope with the current and projected processing loads. Due to many independent sensors in the line array, computation may be distributed over a number of processing elements. The computation is characterized by a set of signal processing functions to be done on the input data, repeated for each new data input.

When the computation is spread over different processing elements (PE's), data must be exchanged between signal processing functions. Data sharing between PE's is determined by the nature of the task. Often the processing done requires that data be realigned in a different manner between functions, as, for example, from frequency versus range to range versus frequency. This realignment has been termed cornerturning, and is similar to matrix transposition [1-4]. The nature of this transposition requires that a large amount of data sharing be done between PE's if the task is distributed.

This thesis examines how a repetitive deterministic signal processing task like frequency domain beamforming may be implemented on a multiprocessor. The signal processing task is repeated for each new data set and does not change. Such tasks are commonly found in the processing of radar and sonar data. The design of a frequency domain beamformer and a pulse Doppler radar system will be used as examples.

1.1 Previous Work

Major areas of research into the design of a multiprocessor system have included the problem of task allocation [4-7], interconnection of PE's and memory, and performance modelling. Most work on task allocation [5-7] has concentrated on reducing interprocessor communication and balancing workloads evenly among PE's. Ideally, processing capacity over one processor increases linearly as PE's are added. In practical systems, control overheads, data transfer requirements and precedence relationships limit the effective processing capacity. Methods suggested for task allocation include graph theoretic mathematical programming or heuristic approaches. These techniques assume that a fully connected network of PE's is available on which a steady stream of jobs is arriving, to be allocated to the PE's. Some stochastic mix of jobs is usually assumed, where the nature and arrival time of the job is not known in advance. The present problem concerns deterministic tasks and hence analysis techniques for stochastic jobs do not apply. The analysis technique in [4] deals specifically with function or job

allocation for a deterministic repetitive task. In it, interprocessor communication and results of function allocation are determined by construction of a timing chart for all PE's.

In a similar manner, a great deal of work has been done on identifying interconnection networks for general processing requirements. A recent survey of interconnection networks is available in [8]. In the present case, interconnection needed between PE's is known completely from the input, output, and interprocessor communication requirements for the PE's.

Bus structures (single bus, bus cluster and multiple bus) are chosen for examination since they are well suited for the type of signal processing task considered. As outlined in [9], the bus architecture provides the lowest cost and least complex system. In the limit of one bus per PE, the system forms a fully connected network. However, multiple bus systems get increasingly complex and expensive as more buses are added. Recent papers that have examined stochastic workloads on multiprocessors with a time shared or multiple bus are [10] and [11].

Considerable interest has been shown in implementing multiprocessor systems with microprocessors. A survey of the problems associated with practical implementation of multiprocessors with microprocessors is given in [12] along with a description of some representative microprocessor chips. Some practical microprocessor implementations for radar [13,14] have been built for very simple tasks with little or no interprocessor communication.

Investigation into how to design systems for more complex tasks has not been extensive. Analysis of multiprocessor systems has for the most part been limited to jobs of a stochastic nature, where the length and type of job is not known a priori. Many sonar and radar systems carry out repetitive tasks for each new data input. Real time processing has, as a design criterion, the minimization of the time taken for each data set input, or cycle time. The cycle time cannot exceed the processing period from one data input to the next, or required processing period. Cycle time is dependent on such things as precedence constraints among the functions comprising the task and requirements for data sharing between functions.

A recent paper by Glass and Kotiveeriah [4] examined how a deterministic task is implemented on a multiprocessor and constructed formulae by which the cycle time could be computed. The multiprocessor consisted of a number of identical PE's connected by buses for data transfer. Each function in the task was allocated to a group of PE's. The input data set was assumed to be evenly divided (being zero extended if necessary) among the PE's. Timing charts for various function allocations (termed uniform and nonuniform in [4]) show a high degree of symmetry owing to the equal division of data. The formulae for cycle times were based on examination of general classes of timing charts. A different formula was required for each combination of tasks, function allocation and bus structure. Most of the analysis in [4] concentrated on a two bus cluster and simple tasks of up to two intermediate transfers. An example of a cycle time

formula for uniform allocation of a task with two functions will be given in Chapter 3.

Equal data division is required in order to make the formula approach tractable but in certain cases this leads to an optimistic estimate of the cycle time [15]. The function done on a PE usually operates on a discrete data amount, termed a data unit. Data must therefore be divided by data units which may not equally divide among the group of PE's. If it does not, then the symmetry of the timing diagram is destroyed and the formula approach is not tenable.

1.2 Present Work

This thesis seeks to extend the analysis technique in [4] to more complex tasks with finite data units and to examine how it may be applied to the task of frequency domain beamforming. The notation and problem statement is reformulated to account for discrete data units and allows for unequal division of data. The nonsymmetric nature of the timing chart makes the formula approach unreasonable. On the other hand, to construct a new chart manually for each function allocation and data division explicitly would require a prohibitive amount of effort. For this reason, a computer program was designed to do the timing chart calculations and compute cycle time for an arbitrary task and bus structure, with specific rules for function assignment and data transfer. This amounts to doing a computer simulation of the processing of one data set. Such a program makes possible the computation of processor and bus utilization, not easily describable by formulae. Data transfer rules for the simulation

enabled bus congestion to be accounted for while constructing the transfer times, rather than examining congestion afterwards.

The contribution of this thesis is primarily in the extension of the analysis technique introduced in [4]. By use of a program to generate timing information rather than using formulae, the following advantages are obtained:

- (1) More general tasks with an arbitrary number of intermediate transfers can be handled.
- (2) A greater number of bus structures may be specified.
- (3) The interactive analysis program is easier to apply than the formula approach.
- (4) Bus and processor utilization are computed as well as cycle times.
- (5) Bus congestion is incorporated directly rather than examined afterwards. Hence the number of buses may be specified not to exceed a given number.
- (6) Unequal data distribution is made possible by the reformulation of the notation used in [4].

Going beyond simulation, the redistribution of data units is investigated to see if unequal distribution of data leads to shorter cycle times. An option of the interactive program enables improvement of the cycle time to be investigated by means of heuristic data scheduling rules.

Chapter 2

A FREQUENCY DOMAIN BEAMFORMING TASK

2.0 Objective

As the first step in the design of a distributed signal processing system, the task to be done must be clearly identified. The aim of this chapter is to define the notation used and to apply the notation to the formulation of the frequency domain beamforming operation, a task common to radar and sonar systems [2,3]. To obtain the cycle time and utilization constraints on a distributed signal processor (DSP), the task structure and data transfer requirements must be examined. This chapter will examine how the processing period may be expressed in clock cycles, and what signal processing functions are associated with the task. Frequency domain beamforming will be explained and broken down into three constituent functions. (As a simpler, more tutorial example, another task involving a pulse Doppler radar detection system will be considered in chapter 5).

2.1 Notation and Problem Statement

A signal processing task F is expressible in terms of a series of functions f_i that must be performed in order. Each f_i must complete and transfer data before the next function f_{i+1} can begin. The present task consists of three functions: a Fourier transform (f_1), a traceforming operation (f_2), and a beamforming operation (f_3). These functions are described in a later section.

Some terminology concerning data must be made clear before proceeding. The expressions "point", "word" and "unit" are used with the following definitions:

- data point: the smallest logical element addressed by a function. Real and complex numbers are examples of data points.
- data word: the amount of data transferred on a data bus in one clock cycle. Depending on bus capacity, fractions or multiples of a data point may constitute a word.
- data unit: the smallest number of points required by a function for processing so that the function may be completed entirely by one processing element (PE).

The DSP is assumed to consist of a number of identical PE's with the same clock rate. Topologies considered include the single bus, single cluster, network cluster and fully connected network. Each PE has a private memory sufficient to store its data and program and some mechanism is assumed to exist to control access to the buses. Figure 2.1 illustrates the bus interconnection networks considered.

When allocated to a PE, a function f_i operates on at least one data unit to produce either an intermediate result to be passed on to the next function f_{i+1} , or a final result if it is the last function of the task. For notational convenience, output from the last function of n functions will be defined to be input to function f_{n+1} .

(a) Single bus

(b) Single cluster

(c) Network cluster

(d) Fully connected network

Fig. 2.1: Topologies considered for a distributed signal processor

As an example of data terminology, suppose that a function to be done is the Fourier transform of 16 points, where each point is complex. A data unit is then 16 points or $m = 16$ points/unit. If the bus can support the transfer of a point in one clock cycle, $w = 1$ word/point. If, as is more likely, two clock cycles are required (one for the real and imaginary part respectively) then $w = 2$ words/point. For each function f_i the following definitions hold:

m_i = the number of points per data unit that is input to function f_i .

w_i = the number of words/point needed to support the transfer of a point for input to function f_i .

Transfer of data out of the DSP to function f_{n+1} may not be required, depending on the nature of the task.

The number of data units input to a function f_i is L_i , where L_i data units must be divided among K PE's allocated to that function. Figure 2.2 illustrates the assignment of L_i data units to f_i . If we define:

$x_{j,i}$ = the number of data units assigned to PE_j for input to function f_i .

then a $K \times n$ matrix of elements can describe the assignment of data units to K PE's for n functions:

$$L = \begin{vmatrix} l_{11} & l_{12} & \dots & l_{1n} \\ l_{21} & l_{22} & \dots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{k1} & l_{k2} & \dots & l_{kn} \end{vmatrix}$$

The time taken to process a data unit by any one of the PE's may be expressed as a number of clock cycles. To obtain an estimate of processing time the most accurate method would be to benchmark execution time on the actual hardware, since all the overhead and implementation details would be taken into account. Failing the availability of hardware, the algorithm used to compute the function must be examined so as to estimate memory and processing requirements. The estimate will only be approximate since overhead will not be taken into account. By either method, the definition of processing time is:

c_i = the number of clock cycles required for completion of function f_i for an input data size of m_i points.

All of the PE's used to process a function f_i are assumed to be identical.

When a data unit is processed, the function usually decreases the number and possibly changes the type of points per unit. For example, 16 complex points may be processed to give 8 real points as output. The decimation factor for function f_i is defined by

Fig. 2.2: Assignment of L_i data units to K PE's.

r_i = the ratio of output points to input points per data unit
for function f_i .

The decimation factor r_i is usually less than one.

Two types of intermediate data transfers will be considered in this work. Normal data transfers occur when a data unit output from function f_i becomes the input to function f_{i+1} . Where the next function is done on a different PE, data transfer must occur. A second type of transfer occurs in signal processing tasks known as realignment or cornerturning. This type requires that the data set from function f_i be transposed prior to commencing function f_{i+1} . Such a realignment is similar to matrix transposition. Figure 2.3 illustrates the transpose of a data set. The number of data units change, with the number of points per data unit output from function f_i becoming the number of data units input to function f_{i+1} .

The volume of data transferred is given in words, where a word requires one clock cycle to transfer from one PE to another. The transfer is defined by:

τ_i = the number of words to be transferred across a data bus
prior to commencing function f_i .

The number of words transferred for intermediate transfers is generally less than the number of words input to the function, since some data remains resident within the PE. For data input to the first function and output from the last function, τ_1 and τ_{n+1} are given by:

Fig. 2.3: A data set and its transpose.

$$\tau_1 = L_1 w_1 m_1 \text{ words}$$

$$\tau_{n+1} = L_n w_n m_{n+1} \text{ words}$$

The number of words for the intermediate transfers $\tau_2, \tau_3, \dots, \tau_n$ depend on the type of transfer, either cornerturning or normal. The following relationships hold for normal transfers:

$$m_{i+1} \frac{\text{points}}{\text{data unit}} = r_i m_i \frac{\text{points}}{\text{data unit}}$$

$$L_{i+1} \text{ data units} = L_i \text{ data units}$$

The number of words transferred from function f_{i-1} to function f_i for normal transfers is

$$\tau_i = \frac{1}{2} \sum_{j=1}^K |\ell_{j,i-1} - \ell_{j,i}| w_i m_i \text{ words.}$$

$i = 2, 3, \dots, n$

If the transfer between function f_{i-1} and function f_i requires that the data be cornerturned, then as a result of the transposition:

$$m_{i+1} \frac{\text{points}}{\text{data unit}} = L_i \text{ data units}$$

$$L_i \text{ data units} = r_i m_i \frac{\text{points}}{\text{data unit}}$$

The number of words transferred on the data bus for cornerturning transfers between f_{i-1} and function f_i is given by:

$$\tau_i = \sum_{m=1}^K \sum_{\substack{j=1 \\ j \neq m}}^K \ell_{j,i-1} \ell_{m,i} w_i \text{ words.}$$

$i = 2, 3, \dots, n$

Figure 2.4 shows two examples of intermediate transfers, both normal and cornerturning.

2.2 Notation Summary

The specification of a signal processing task requires that the function and transfer requirements be determined. The following vectors and matrix describe the task specification and implementation on K PE's:

$$F = [f_1, f_2, f_3, \dots, f_n, f_{n+1}] \text{ functions}$$

$$m = [m_1, m_2, m_3, \dots, m_n, m_{n+1}] \text{ points/data unit}$$

$$w = [w_1, w_2, w_3, \dots, w_n, w_{n+1}] \text{ words/point}$$

$$L = [L_1, L_2, L_3, \dots, L_n] \text{ data units}$$

$$l = \begin{bmatrix} l_{11} & l_{12} & \dots & l_{1n} \\ l_{21} & l_{22} & \dots & l_{2n} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ l_{K1} & l_{K2} & \dots & l_{Kn} \end{bmatrix} \text{ data unit assignment to K PE's}$$

$$r = [r_1, r_2, r_3, \dots, r_n] \text{ decimation factors}$$

$$\tau = [\tau_1, \tau_2, \tau_3, \dots, \tau_n, \tau_{n+1}] \text{ word transfers}$$

$$c = [c_1, c_2, c_3, \dots, c_n] \text{ cycles/data unit}$$

It should be noted that the vector elements are not all independent from one another. For example L_{j+1} and L_j are related by the type of intermediate transfer.

Fig, 2.4: Typical transfers for normal and cornturning exchanges.

2.3 Frequency Domain Beamforming Task

This section describes the signal processing task of frequency domain beamforming within the framework of the previous section. The task is split up into three functions, each of which is examined for computation and transfer requirements. Before describing the constituent functions, an overview of frequency domain beamforming is given.

The objective of beamforming in general is to gain information about the spatial distribution of an acoustic field from a distributed set of sensors. This information is normally displayed as a presentation of frequency versus bearing. A common sensor distribution is the equi-spaced line array, and that is the geometry considered here. Due to the symmetric nature of the line array, wavefront arrivals coming from either side of the array cannot be distinguished. Bearings are only determined to within 180 degrees relative to the front of the array. Figure 2.5 illustrates a wavefront and the corresponding alias impinging on a line array of sensors.

Wavefront arrival information is obtained by combining sensor outputs to obtain wavefront time delay information. For the algorithm described here, sensor outputs are implemented by phase shifts. By applying an exponential phase factor to each of the transformed sensor outputs, array response may be steered in an arbitrary direction from 0 to 180 degrees. Unweighted sensor outputs have a maximal array response at 90 degrees, corresponding to no time delay between sensors. The equation describing sensor output combination for G sensors is:

Fig. 2.5: An arriving wavefront and alias impinging on a line array of sensors. Bearing θ is determined only from 0 to 180° relative to the array front.

$$B_{\theta}(k) = \sum_{i=0}^{G-1} X_i(k) e^{-j\phi(\theta,k)i} \quad (1)$$

$B_{\theta}(k)$ = array response in direction θ for frequency bin k .

$X_i(k)$ = Fourier transform of sensor output for frequency bin k and sensor i .

$\phi(\theta,k)$ = exponential phase factor for direction θ and frequency bin k .

What is normally desired is the beam power magnitude $|B_{\theta}(k)|$ as a function of angle θ and bin k . It is possible to compute $|B_{\theta}(k)|^2 = B_{\theta}(k)B_{\theta}^*(k)$ using equation (1) to get

$$|B_{\theta}(k)|^2 = \text{Trace}(0) + 2 \operatorname{Re} \left\{ \sum_{i=1}^{G-1} \text{Trace}(i) e^{-j\phi(\theta,k)i} \right\} \quad (2)$$

where

$$\text{Trace}(i) = \sum_{\ell=0}^{G-1-i} X_{\ell}(k) X_{i+\ell}^*(k) \quad i = 0, 1, \dots, G-1. \quad (3)$$

In cases where sensor data is obtained from a wide-sense stationary environment, averaging of the trace values prior to formulation of the beam magnitude may be done to increase statistical stability. This is not considered here.

The task of obtaining the beam power for G sensors of a line array may conveniently be broken down into three functions:

Fig. 2.6: Functional diagram for the task of frequency domain beamforming

- f_1 : the computation of $X_\ell(k)$, the Fourier transform of sensor ℓ , $\ell = 0, 1, 2, \dots, G-1$.
- f_2 : the computation of trace values for each frequency cell k , using equation (3).
- f_3 : the formulation of a number of beams in different directions by a weighted combination of trace values, using equation (2).

Figure 2.6 shows a representation of the three functions, with data input, output and intermediate data transfers. Each of the functions will now be examined in detail.

2.3.1 Data Input and Buffering

Data from each sensor is continuously collected and fed into an input buffer area, used to store data while the DSP is processing the previous data set. Data transfer from the buffer area to the PE's occurs via one specific bus, termed the "input feed bus". The buffer size must be large enough to hold an input sample set of size N from G sensors, or a total of GN points. When the buffer is full, the buffer contents are fed from the buffer via the input feed bus to each of the PE's.

Following the definition of a data unit outlined previously, the input sample set from each sensor is a data unit with the number of points per data unit being the sample size. This gives $L_1 = G$ data units and $m_1 = N$ points/data unit.

For the system to operate in real time, so no data are lost while processing, the buffer storage area must not overflow and therefore

processing time cannot exceed data collection time. Depending on the sampling rate and sample size, a required processing period is thus defined. For example, a 400 Hz sampling frequency with a 1024 point sample size has a required processing period of:

$$T_c = \frac{1024 \text{ samples}}{400 \text{ samples/sec}} = 2.56 \text{ seconds}.$$

The cycle time (time to complete processing of one buffer and be ready to input the next) cannot exceed the required processing period for the DSP to operate in real time.

2.3.2 f_1 : Fourier Transformation of Sensor Inputs

Computation of transforms of data units proceeds as soon as inputs to PE's are complete. Normally an FFT algorithm is used to compute the transform, and an estimate can be made of the number of clock cycles to compute an N-point transform. Such an estimate is highly processor dependent and therefore approximate. Benchmark times on an actual microprocessor would likely be used in an implementation. For a PE with a single-word memory fetch and store, real adder and real multiplier, 28 clock cycles are required to compute one butterfly [16]. With $N/2 \log_2 N$ butterflies per N-point transform, the FFT function requires $14N \log_2 N$ clock cycles in total. Thus, a 1024 point transform takes $c_1 = 143,360$ clock cycles to complete.

The transfer between the FFT function f_1 and the traceforming function f_2 is of the cornerturning type. The data must be realigned so that frequency cells of a particular frequency from each sensor may

be processed together. It is not necessary however to process the entire frequency range of the transform. A subset (frequency band) may be chosen to reduce the processing load. In this case, the number of data units input to the traceforming function would be reduced. As an example, the transposition of 4 frequency cells is illustrated in figure 2.7, with $r_1 = 1/2$.

2.3.3 f_2 : Traceforming of Frequency Cells

This function involves the computation of the cross power spectrum between pairs of sensors, and is done for each frequency bin in the frequency band processed (equation (3)). The computation requires that enough memory be resident within each PE to accommodate both G trace and G frequency cell values, as the computation cannot proceed in place. A direct implementation of equation (3) requires $\frac{G(G+1)}{2}$ complex multiplications and $\frac{G(G-1)}{2}$ additions for G frequency cells. Assuming again a single word memory fetch, real adder and real multiplier, a complex multiply requires 8 memory cycles (4 fetch and 4 store), 2 multiply and 2 add cycles. Both estimates do not account for store and fetch pipelining or overhead. A more accurate estimate would again be made by benchmarking the function on a particular microprocessor.

The number of cycles required is therefore taken as $\frac{14G(G+1)}{2} + \frac{10G(G-1)}{2}$ cycles per data unit consisting of G frequency cells. For 16 sensors, this yields $c_2 = 3,104$ cycles per data unit.

Fig. 2.7: Transfer between transform and traceforming for a band of 4 frequency cells.

2.3.4 f_2 : Indirect Method of Traceforming

A second, more indirect method of computing equation (3) involves using inverse Fourier transforms [17]. Trace elements are obtained from an inverse transform of 2G-point real coefficients:

$$\begin{matrix} \text{Trace } (i) \\ i=0,1,\dots,G-1 \end{matrix} = \text{IDFT} [\text{Mag}(\ell)] \quad \ell = 0,1,\dots,2G-1 \quad (4)$$

The 2G-point real coefficients are generated by two G-point inverse transforms:

$$\begin{matrix} \text{Mag}(\ell) \\ \ell=0,2,4,\dots,2G-2 \end{matrix} = \left| \text{IDFT}[X_m(k)] \right|^2 \quad m = 0,1,\dots,G-1 \quad (5)$$

$$\begin{matrix} \text{Mag}(\ell) \\ \ell=1,3,5,\dots,2G-1 \end{matrix} = \left| \text{IDFT}[e^{j\frac{\pi m \ell}{G}} X_m(k)] \right|^2 \quad m = 0,1,\dots,G-1 \quad (6)$$

Computation via this method requires one 2G-point real and two G-point complex transforms, G complex multiplies and 2G magnitude calculations. A 2G-point real or G-point complex transform takes $14G \log_2 G$ cycles, a complex multiply takes 14 cycles and a magnitude calculation takes 1 cycle to complete. The total number of cycles required to compute function f_2 via this indirect method is therefore $42G \log_2 G + 12G$. The number of cycles required is less for 16 sensors or more, and will be used in chapter 4 where 16 sensors are used in the task example. Table 2.1 shows the number of cycles per data unit for the number of sensors being a power of 2. Depending on the FFT routine used, the indirect method may be restricted to cases where the number of sensors is a power of 2.

NUMBER OF SENSORS	NUMBER OF CYCLES PER DATA UNIT	
	DIRECT METHOD	INDIRECT METHOD
2	52	96
4	200	384
8	784	1,104
16	3,104	2,880
32	12,352	7,104

Table 2.1: A comparison of cycle counts for direct and indirect methods of trace computation.

Transfer of data units between the traceforming operation f_2 and the beamforming operation f_3 is normal, with each data unit being passed on for further processing. No transfer is required if the data unit is processed by the same PE.

2.3.5 f_3 : Beamforming From Trace Values

The last function f_3 involves the formation of a number of beams between angles θ_1 and θ_2 where $0 < \theta_1 < \theta_2 < 180$ degrees. Instead of using equation (2) directly it is preferable and easier to generate the exponential phase factors recursively for each direction and frequency cell. Suppose that B beams are to be generated for a frequency cell k from G trace values between θ_1 and θ_2 inclusively. If the following exponential phase factors are defined:

$$\alpha = \exp j \left| \frac{2\pi d f_s \cos \theta_1}{Nv} \right|, \quad \beta = \exp j \left| \frac{2\pi d f_s (\cos \theta_2 - \cos \theta_1)}{Nv(B-1)} \right| \quad (7)$$

where

d = intersensor separation in the line array

f_s = sensor sampling frequency

v = velocity of the plane waves in the medium

N = sensor sample size

θ_1 = first beam angle

θ_2 = last beam angle

B = number of beams to form between θ_2 and θ_1 inclusive.

Equation (2) may be rewritten to incorporate the exponential phase factors of equation (7):

$$|B_{\theta}(k)|^2 = \text{Trace}(0) + 2 \operatorname{Re} \left\{ \sum_{i=1}^{G-1} \text{Trace}(i) \alpha^k \beta^{bk} \right\} \quad (8)$$

for $b = 0, 1, \dots, B-1$ beam angles.

In this manner, provided that α and β were stored in each PE, phase factors could either be generated when needed or (with enough memory) be pre-stored as constants. Assuming that α^k and β^k were stored and available, computation of equation (8) requires one real multiply, $3(G-1) + 1$ complex multiplies and $G-1$ additions for $b \geq 1$. For $b = 0$, only one real multiply, $2(G-1)$ multiplications and $G-1$ additions are required. Each complex multiply takes 14 cycles while real additions and multiplications take 3 cycles each (fetch, add/multiply and store). To produce B beams thus takes $17(B-1) + 45(G-1)(B-1) + 31(G-1) + 3$ cycles to complete. For example, assuming stored α^k and β^k , to produce 10 beams from 16 trace values takes $c_3 = 6696$ cycles.

Transfer of the beam output may be required if the results are sent via a data bus to a display device. The output data unit consists of B beam magnitudes for each frequency cell desired.

2.4 Summary of the Beamforming Task

The workload of the beamforming task is expressible in terms of the functions f_1 , f_2 and f_3 where the processing time is expressed in clock cycles. Table 2.2 summarizes the three function requirements

$$\begin{aligned}
 F &= (f_1, f_2, f_3) = (\text{Fourier transform, traceform, beamform}) \\
 m &= (m_1, m_2, m_3, m_4) = (N \text{ real, } G \text{ complex, } G \text{ real}) \frac{\text{points}}{\text{data unit}} \\
 L &= (L_1, L_2, L_3) = (G, r_1 N, r_1 N) \text{ data units} \\
 c &= (c_1, c_2, c_3) = (14N \log_2 N, c_2, 17(B-1) + 45(G-1XB-1) + 31(G-1) + 3) \frac{\text{cycles}}{\text{data unit}}
 \end{aligned}$$

$$\text{where } c_2 = \begin{cases} \frac{14G(G+1)}{2} + \frac{10G(G-1)}{10} & \text{for direct method} \\ 42 G \log_2 G + 12G & \text{for indirect method} \end{cases}$$

$$\tau = (\tau_1, \tau_2, \tau_3, \tau_4) = (w_1 G N, \tau_2, \tau_3, w_4 G B) \text{ words} \quad \tau_2 = \text{cornerturning transfer}$$

$$\tau_3 = \text{normal transfer}$$

Table 2.2: Functional requirements for the beamforming task.

in terms of N (the sample size from each sensor), G (the number of sensors), r_1 (the frequency band processed) and B (the number of beams formed).

In chapter 4 a representative set for N , G , r_1 and B is selected and the function requirements are implemented on a multiprocessor. The effect of different numbers of PE's, function allocation and data bus width is investigated.

Chapter 3

FUNCTION ALLOCATION AND DATA SCHEDULING

3.0 Introduction

In the analysis of very regular, never changing, repetitive signal processing tasks, the objective is to determine the minimum possible cyclic repetition period. The achievable throughput rate is determined by the time taken from one input to the next. Careful allocation of functions and division of data to the PE's in a DSP will ensure high utilization and throughput. An alternative objective is, given the cycle time that must be achieved to accomplish real time processing, to find the minimum number of PE's required. In the example cited in the last chapter for the beamforming problem, the required cycle time was 2.56 seconds. This chapter outlines the analysis method and gives a computer program to do the analysis. The next chapter applies the analysis method to the beamforming problem.

Since the signal processing task is completely known, it is possible to construct a timing chart once the data division and function assignment has been made. A simple signal processing task is examined in the next section to show how the timing chart is constructed.

3.1 Timing Chart Example

A simple example illustrates construction of a timing chart. The signal processing task example consists of two functions where the intermediate transfer is of the cornerturning type. A summary of the functional requirements of this example is given in table 3.1. The structure of the DSP for this example is the single bus architecture. Figure 3.1 shows the data flow over one cycle from input to output. Figure 3.2 illustrates the timing diagram and bus utilization.

The constraint of bus feed, transfer and output requirements means that each PE cannot be fully utilized. The next input cannot start until the input bus is free and the PE's are ready to process another data set. When output is not required from the last function, the cycle time is determined by the largest cycle period at any PE. When output is required, the next input must wait until the last output has completed if there is only one bus available, as in this example. For this task, the utilization of each PE is 45.5% (20 out of 44 cycles) and the bus utilization is 61.4% (27 out of 44 cycles). The processing period is determined by the cycle time and clock rate (R Hz) where:

$$\text{processing period} = \frac{44}{R} \text{ seconds .}$$

Response time is defined to be the time from input commencement to completion of the last function. In this example, response time and processing period are identical. In situations where more than one bus is available, the response time will in general be greater

$F: \{f_1, f_2\}$
 $m: \{m_1, m_2, m_3\} = \{6, 3, 1\}$ points/data unit
 $w: \{w_1, w_2, w_3\} = \{1, 1, 1\}$ words/point
 $L: \{L_1, L_2\} = \{3, 3\}$ data units $c: \{c_1, c_2\} = \{10, 10\}$ cycles/data unit

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \\ l_{31} & l_{23} \end{bmatrix}$$
 data unit assignment
 $r: \{r_1, r_2\} = \{1/2, 1/3\}$ decimation factor
 $\tau: \{\tau_1, \tau_2, \tau_3\} = \{18, 6, 3\}$ words τ_2 - cornerturning

Table 3.1: Function requirements for a signal processing task example.

Fig. 3.1: Data flow for a signal processing task example.

Fig. 3.2: Timing Diagram for a signal processing task example.

than the processing period. The processing period (or cycle time) must not exceed the required processing period if real time processing is desired.

3.2 "Formula" Approach to Determine Cycle Time

Construction of a timing chart allows a cycle time to be computed, based on the number of cycles required for each PE. For equal data division there is symmetry among the PE's assigned to each function. In [4], this symmetry was used to construct formulae that described cycle time in terms of functional parameters and type of function allocation. All intermediate data transfers were assumed to be of the cornerturning type and output from the last function was not accounted for. Formulae was developed for tasks with up to two intermediate transfers with uniform or pipeline allocation on a two bus cluster, based on the assumption of equal data division. The concept of a data unit was not developed.

As an example of a formulae for cycle time for a task with one intermediate transfer with uniform allocation, the following formula describes the number of cycles required:

$$T_c = M + \frac{N}{P} + \frac{2r_1 M}{P^2} + t_w$$

where, in the present notation:

$M = \text{total data input} = L_1 m_1 w_1$

$N = \text{total number of cycles for } f_1 \text{ and } f_2 \text{ to complete} =$
 $c_1 L_1 + c_2 L_2$

$t_w = \text{waiting period, if any, due to bus or processor unavailability.}$

Since the formula approach was unable to account for bus congestion directly, t_w had to be determined by examination of the timing diagram afterwards.

Where the number of data units in a data set could not be equally divided among the PE's assigned to a function, the formulae given in [4] were approximate and gave an overly optimistic estimate of cycle time. The effect of finite sized data units has been examined in [15]. The idea of extending the formulae developed in [4] was rejected for the following reasons:

- (1) Unequal data division destroys the symmetry between PE's and makes development of a general formula difficult.
- (2) More complex bus structures and tasks with more functions are not easily analyzable in terms of formulae.
- (3) Processor and bus utilization are not easily computable with formulae.
- (4) Bus congestion cannot be accounted for, but must be examined afterwards. The assumption that enough buses were present to accommodate all transfers must be verified afterwards.

Clearly, a better method is required to fully investigate cycle times for more general bus structures and tasks. The next section details an interactive program that computes cycle times based on input of function parameters. In addition to computing cycle times, bus and processor utilization are output and bus congestion is incorporated directly via data transfer rules. A heuristic method is available to improve data unit assignment to PE's.

3.3 Gannt - A Fortran77 Program for Analysis

A method to determine cycle time without explicitly requiring a timing chart has been programmed in Fortran77 for the Honeywell Level-66 computer at Carleton University. The program name is taken from the Gannt activity chart used in activity analysis. The program is designed to be used interactively, with the user prompted for responses to questions asked. The data transfer rules used as well as an explanation of the improvement process for assignment of data units now follow.

3.3.1 Data Transfer Rules

In the transfer of data from one PE to another it is frequently the case that more than one transfer is enabled for a given clock cycle. If not enough buses are available to support all the transfers, contention for the bus occurs. The number of transfers that can proceed are constrained by the nature of the PE's (a PE is assumed not to be able to transmit and receive data simultaneously) and by the number of buses available. Some rules must be adopted as to what PE

should gain access to the bus and in what order so as to minimize cycle time. The following heuristic rules are used by the program:

- (1) Transfers from one PE to another cannot be interrupted by another transfer once the transfer of data has commenced; i.e., all transfers are carried to completion.
- (2) Transfers from one PE to another (both the sending of data and receiving, if any) are done consecutively. A send operation from a lower numbered PE is started first, followed by a send operation from the PE that received the data.
- (3) Transfers that are ready to proceed are given priority. Transfer requests are queued first come, first served.

These rules were adopted for simplicity and so that no excessive transfer delays occur. Since input begins on lower numbered PE's, these were given priority in sending.

3.3.2 Assignment of Data Units

Based on the initial number of data units and the processing done, the number of data units input to each function is calculated by program Gantt. These are initially equally distributed to all the PE's in the group assigned to the function. Where this is not possible, the distribution of extra units begins at the lower numbered PE's.

In a cornerturning transfer, if two points are transferred per word, it will not be possible to allocate individual data units to PE's.

Figure 3.3 illustrates an example of where this occurs in a cornerturning

Fig. 3.3: Effect of bus transfer size on allocation of data units.

transfer, and how the transfer of one point per word enables individual data units to be assigned to a greater number of PE's.

3.3.3 Reassignment of Data Units

While the equal division of data units usually gives reasonably good cycle times, it can be improved upon in most cases. One method of improvement is to remove a data unit from one PE and assign it to another, calculate the resultant cycle time and see if the cycle time has been shortened. This trial and error method requires that a complete set of times for the timing chart be calculated for every new data unit allocation. If removal of a data unit from one PE shortens the cycle time, that PE is examined again. This is done for data unit input to each function until no single data unit may be reallocated to any PE for any function without lengthening the cycle time.

While robust, the improvement process requires large amounts of CPU time for increasing numbers of processing elements. In addition, the final result is not optimal as multiple unit reallocation (i.e., switching 2 data units from one PE to another) may lead to a shorter cycle time. Nevertheless, substantial improvement in cycle time can be obtained, as will be seen in the example of chapter 5.

3.3.4 Structure of Program Gannt

Figure 3.4 shows an overview of the program module structure. An arrow direction indicates that one module calls another; e.g., Program Gannt calls subroutine Init.

A short description of each of the modules and the function performed now follows. Listings of each of the modules are included in the Appendix.

Program Gannt: The control module that co-ordinates activity.

All other modules are called from this module, and the user interactively controls activity from it.

Subroutine Init: Called from program Gannt, this module initializes all data values and zeros the arrays used to hold timing information.

Subroutine Setup: After calling Init, program Gannt calls this module to ask for parameters that characterize the signal processing task to be done. If called a second time (to do another task) previous inputs may be retained by a carriage return.

Subroutine Assign: Prior to beginning timing analysis, the program assigns an initial division of data units to each function based on functional assignment to PE's and using an equal a division of data units as possible.

Subroutine Input Data: The start and stop times for each PE to input data are calculated by this module. The input bus is always bus 1.

Fig. 3.4: An overview of the modules for program GANNT.

Subroutine Do Func: This module calculates the start and stop times for each PE to complete a function. It is called after either transfer or input times have been computed. Subroutines Do Func and Transfer are called in succession until all functions are done.

Subroutine Transfer: Transfer times for both normal and corner-turning transfers are calculated by this module. It is called after a function has been completed, if it is not the last function in the task. Rules for data transfer are discussed in the next section.

Subroutine Out: Start and stop times for output from the last function on a single bus are calculated by this module. The first available bus (apart from the input feed bus, which is not used unless required) is used.

Subroutine Find Cycles: The cycle period for each PE are found by this module. It is called after times for all functions in the task have been computed.

Subroutine Find Maximum: The cycle time from input to input is computed by this module. The cycle time is generally the maximum cycle period of the PE's unless constrained by output of the final result.

Subroutine Result: All of the timing information calculated by the above modules is output to the user by this module.

Subroutine Reassign: The assignment of data units to PE's may be altered by this module so as to examine the effect on cycle time. The user is prompted interactively to enter a new assignment after the present one is output for information. If no new assignment is entered (by defaulting with a carriage return) the present assignment for the function is not changed.

Subroutine Improve: In this module, each function data assignment is examined to see if reassignment of a data unit from one PE to another will shorten the cycle time. The set of routines used to compute cycle time (Input Data, Do Func, Transfer, Find Cycles, Find Maximum) are used each time a new assignment of data units is made. Any new assignment that is not advantageous is rejected and the original assignment maintained. The improvement process terminates when no switch from one PE to another of one data unit for any function in the task leads to a decrease in cycle time.

System Routine Prompt: A system routine written in PL6 enabled Fortran input and output to be on the same line. The routine, Prompt, is included in the appendix and has the calling sequence

Call PROMPT ('string', no-characters) where:

'String' is a string of up to 30 characters output before a read is done.

no-characters is the integer variable containing the number of characters in the string.

3.4 An Example of an Interactive Session

The record of an interactive session is given in figure 3.5. The first question asks the type of terminal, since specialized I/O features like reverse video were used to highlight output on a CYBERNEX terminal. The user is then prompted for all necessary information to specify the task and the number and type of bus structure, characterized by the bus width and number of buses available. Each function may be allocated to all the PE's or a subset of them. For this example, the sample task in Section 3.2 is used again.

On completion of user input, the cycle time with average PE and bus utilization are output. The user is prompted as to whether more detailed information is desired. If answered yes, the start and stop times of input, functions, transfers and output are given with a carriage return between each to allow viewing of the output on the terminal screen. The option of improving the reassignment of data units is given, and is used in this example to show that equal distribution was the best division that could be found. The option of returning to the start of the program is available, and previous answers may be retained.

3.5 Summary

Once the task to be done has been completely defined, it is possible to construct a timing chart to obtain cycle time, processor and bus utilization. By assuming a clock speed, throughput and response time may also be obtained. Construction of the timing chart is tedious for

Fig. 3.5: Example of an interactive session with program Gannt.

```

PE Element Utilization
-----
PE 1: 45.45 X
PE 2: 45.45 X
PE 3: 45.45 X

Bus Utilization:
-----
Bus Number 1: 81.76 X

Do you want to reassign data units?
Answer Y or N..... Y

Data Unit Reassignment
-----

For func 0 1 the assignment now is:
1 1
For func 0 2 the assignment now is:
1 1
For func 0 1:
Enter the new assignment.....0,1,2
For func 0 2:
Enter the new assignment.....2,1,0

The cycle time is 45 cycles.

The average processing element utilization = 30.77 X

Bus Utilization:
-----
Bus Number 1: 44.42 X

Do you want detailed timing information?
Answer Y or N..... Y

**** Program Gantt Results ****
-----
PE 1: Input Start End Func #1 Start End
PE 2: 0 1 0 7
PE 3: 1 18 0 19

Func #1 Start End
0 7
19 38

PE Element Utilization
-----
PE 1: 30.77 X
PE 2: 30.77 X
PE 3: 30.77 X

Bus Utilization:
-----
Bus Number 1: 44.42 X

Do you want to reassign data units?
Answer Y or N..... N

CR to continue.....

Transfer Func 1--> Func 2
-----
Proc 1 Proc 2 Proc 3
Proc 1 8 8 8
Proc 2 18 0 0
Proc 3 20 0 0
Proc 3 39 43 0
Proc 4 42 44 0
CR to continue.....

Func # 2 Start End
PE 1: 43 62
PE 2: 48 58
CR to continue.....

Output Start End
PE 1: 63 64
PE 2: 55 58
CR to continue.....

Cycle Start End Cycle Period
PE 1: 18 64 47
PE 2: 1 55 55
PE 3: 1 44 38
CR to continue.....

PE Element Utilization
-----
PE 1: 30.77 X
PE 2: 30.77 X
PE 3: 30.77 X

Bus Utilization:
-----
Bus Number 1: 44.42 X
  
```

Fig. 3.5: Example of an interactive session with program Gantt (cont'd).

```

Do you want to try to improve the
assignment of data units to the functions?
Answer Y or N.....

*** Starting the data unit reassignment ***

Improved cycle time is 44 For function 1 Data unit from PE 2 moved to 1
Improved cycle time is 58 For function 1 Data unit from PE 3 moved to 1
Improved cycle time is 33 For function 1 Data unit from PE 1 moved to 2
Improved cycle time is 44 For function 2 Data unit from PE 1 moved to 3

The best cycle time was 44

The best assignment of data units was:
For function #1
1 1
For function #2
1 1
CR to see the timings.....

The cycle time is 44 cycles.

PE Utilization
-----
The average processing element utilization = 45.45 %

Bus Utilization
-----
Bus Number 11 41.36 %

Do you want detailed timing information?
Answer Y or N..... N

Do you want to reassign data units?
Answer Y or N..... N

Do you want to try to improve the
assignment of data units to the functions?
Answer Y or N..... N

Want to try another signal processing task?
Answer Y or N..... N

**STOP**
/ INPUT MESSAGE
DATE TIME OFF 11:34 10/05/82

```

Fig. 3.5: Example of an interactive session with program Gannt (cont'd).

more complicated tasks and structures. Derivation of formulae to cover every case is not practical owing to the wide variation of input assignment, and is limited in any case by the complexity of tasks to be handled. A more flexible approach is to use a computer program to generate timings of activities without explicit construction of a chart. Interactive input allows a wide variation of tasks, data assignments and structures to be examined. In addition, bus congestion is incorporated directly and need not be examined afterwards.

Chapter 4

IMPLEMENTATION OF FREQUENCY DOMAIN BEAMFORMING ON A MULTIPROCESSOR

4.0 Scope of Investigation

This chapter is intended to show the implementation of frequency domain beamforming as described in Chapter 2, using the analysis techniques outlined in Chapter 3. The task implementation on a multiprocessor with increasing numbers of PE's will illustrate the effect of finite data units. Bounds on the processing period result from finite sized data units, regardless of how many PE's are added. In this chapter uniform function allocation is considered for a multiprocessor with a single bus of single and double width. A single width bus can support a real point transfer in one clock cycle while a double width bus can support a complex point or two real point transfers.

A comparison will be made between uniform function allocation, where each function is allocated to all the PE's, and pipeline function allocation, where each function is allocated to a group of PE's. The particular pipeline function allocation considered here allocates f_1 to one group of PE's and f_2, f_3 to the other group. Figure 4.1 illustrates uniform and pipeline allocation for 3 PE's. For the comparison between uniform and pipeline allocation, a multiprocessor with a single width bus will be considered.

In this chapter equal data unit division, as far as possible, will be used when assigning data units to PE's for both uniform and pipeline allocation. Unequal data unit division will be considered in Chapter 5 for the radar task where rescheduling of data leads to a reduction in cycle time.

Bus and processor utilization for the uniform allocation case will also be shown to illustrate the effect of finite data units.

4.1 Task Description

To investigate all possible values of numbers of sensors, sample sizes, frequency bands and number of beams would be beyond the scope of this work. Instead, a particular choice of G , N , r_1 and B is made, representative of realistic values that might be encountered. The four values characterizing the beamforming task are:

- G : the number of sensors in the line array. More sensors give a higher spacial resolution for the beams. For this implementation, the line array consists of 16 sensors.
- N : the sample size from each sensor. In conjunction with the sampling frequency, the sample size fixes the frequency resolution. Here, the sample size is fixed at 1024 points.
- r_1 : the fraction of frequency bins from the initial transform retained for further processing. Not all frequency bins are usually of interest. In this instance, 128 out of 1024 bins are retained for examination ($r_1 = 0.125$).

Fig. 4.1: Uniform function allocation for beamforming task with 3 PE's.

Fig. 4.2: Pipeline function allocation for beamforming task with 1 PE for f_1 and 2 PE's for f_2, f_3 .

B: the number of beams to form. The number is related to the spacial resolution available. Here 10 beams for each frequency bin retained will be formed.

The output from the beamforming function f_3 will be sent to some displays or storage device via a data bus. In a single bus multiprocessor, the input feed bus must be used. For pipeline allocation, using a single bus causes the group processing f_1 to not input data until the output from function f_3 has completed. To avoid long waits for input, a second data bus is used to output beams. The bus topology used for pipeline allocation is the single cluster.

With the task description above, and using the indirect method of traceforming for function f_2 , the task specification becomes

$$m = (1024 \text{ real}, 16 \text{ complex}, 16 \text{ complex}, 10 \text{ real}) \frac{\text{points}}{\text{data unit}}$$

$$w = (1,2,2,1) \frac{\text{words}}{\text{point}} \text{ (single width bus)}$$

$$w = (0.5,1,1,0.5) \frac{\text{words}}{\text{point}} \text{ (double width bus)}$$

$$L = (16,128,128) \text{ data units}$$

$$c = (143360,2880,6996) \frac{\text{cycles}}{\text{data unit}}$$

4.2 Minimum Cycle Time for Uniform and Pipeline Allocation:

Using the concept of a data unit and the processing times of each function, it is easy to place a minimum bound on the cycle time achievable for either uniform and pipeline function allocation. The

minimal amount of data that can be allocated to a PE is one data unit. Input, processing and transfer of function f_1 (transform) will be the bottleneck for both uniform and pipeline allocation, since $c_1 = 143360$ is two orders of magnitude greater than c_2 or c_3 . Since the transfer to f_2 is cornerturning, f_1 must complete on all PE's prior to the commencement of f_2 on any PE. For systems with only one bus, output of the result must complete before input of the next data set can begin. In order to make pipeline allocation sensible, a second data bus for output is required. The minimum cycle time achievable depends on the bus width and type of function allocation.

4.2.1 Uniform Allocation

The minimum cycle time will occur when each PE is allocated one data unit for f_1 , f_2 and f_3 . For the task parameters considered, 128 PE's are required to reach this limit. The cycle time is then just the input time for one data set, processing and transfer time for one data unit for each function plus output of the result. The calculation is listed below for both single and double wide bus.

(i) single wide bus

data set input:	$1024 \frac{\text{points}}{\text{unit}} \times 16 \text{ units} \times 1 \frac{\text{word}}{\text{point}} =$	16384
process f_1 :	$1 \text{ unit} \times 143360 \frac{\text{cycles}}{\text{unit}} =$	143360
send and receive 16 complex points:	$2 \times 16 \frac{\text{points}}{\text{unit}} \times 2 \frac{\text{words}}{\text{point}} =$	64

process f_2 :	$1 \text{ unit} \times 2380 \frac{\text{cycles}}{\text{unit}}$	=	2880
process f_3 :	$1 \text{ unit} \times 6996 \frac{\text{cycles}}{\text{unit}}$	=	6996
output result:	$10 \frac{\text{points}}{\text{unit}} \times 128 \text{ units} \times 1 \frac{\text{word}}{\text{point}}$	=	1280
Total number of cycles			170964

(ii) double wide bus

data set input:	$1024 \frac{\text{points}}{\text{unit}} \times 16 \text{ units} \times 0.5 \frac{\text{words}}{\text{point}}$	=	1892
process f_1 :	$1 \text{ unit} \times 123360 \frac{\text{cycles}}{\text{unit}}$	=	143360
send and receive 16 complex points:	$2 \times 16 \frac{\text{points}}{\text{unit}} \times 1 \frac{\text{word}}{\text{point}}$	=	32
process f_2 :	$1 \text{ unit} \times 2880 \frac{\text{cycles}}{\text{unit}}$	=	2880
process f_3 :	$1 \text{ unit} \times 6996 \frac{\text{cycles}}{\text{unit}}$	=	6996
output result:	$10 \frac{\text{points}}{\text{unit}} \times 128 \text{ units} \times 0.5 \frac{\text{words}}{\text{point}}$	=	64
Total number of cycles			162100

4.2.2 Pipeline Allocation

In the pipeline allocation, cycle time is determined by the largest cycle period for either group of PE's. Since c_1 is much larger than c_2 or c_3 , the bound on the cycle time will result from processing f_1 . Pipeline allocation requires that a second data bus be available for output. For a single wide bus the cycle time is bounded by:

input data set:	$1024 \text{ points} \times 16 \text{ units} \times 1 \frac{\text{word}}{\text{point}}$	= 16384
process f_1 :	$1 \text{ unit} \times 143360 \frac{\text{cycles}}{\text{unit}}$	= 145360
transfer one unit:	$1 \text{ unit} \times 128 \frac{\text{points}}{\text{unit}} \times 2 \frac{\text{words}}{\text{point}}$	= 256
Total cycles for group		160000

4.3 Implementation with Uniform Allocation

Using the program outlined in chapter 3, the cycle time for the beamforming task as executed on a multiprocessor with up to fifty PE's was computed for both single and double width bus. To plot the increase in processing speed, the speedup in processing time over a single PE was computed for each PE number. This speedup was defined to be the "effective number of PE's", using the following ratio:

$$\text{effective number of PE's for } j \text{ PE's} = \frac{\text{cycle time for } j \text{ PE's}}{\text{cycle time for 1 PE}}$$

In ideal situations, the speedup is linear with more PE's. In reality, the effect of data unit size and data transfer requirements limit processing gain until the minimum possible cycle time is reached. In terms of effective number of PE's the maximum obtainable speedup over a single PE are 20.7 and 21.8 for a single and double width bus respectively.

Figure 4.3 shows a plot of effective number of PE's versus the actual number of PE's for the beamforming task with uniform function allocation and a single and double width bus. The curves in figure 4.3 show the effect of being unable to equally divide data units for

function f_1 among the PE's. This accounts for the large deviation from linearity for 15 PE's and the sudden jump in processing speed when the 16 data units for function f_1 are evenly divisible at 16 PE's. Smaller jumps result from data units for f_2 and f_3 being divisible for certain numbers of PE's. Similar large increases are noted at 4 and 8 PE's, as the data input and processing of f_1 are much longer than for the other functions.

The graphs of processor and bus utilization shown in figures 4.4 and 4.5 show a similar effect. A large increase in PE efficiency results at 16 PE's as the other PE's do not have to wait while one PE processes 2 data units for function f_1 . Similar peaks in PE efficiency are noted at 4 and 8 PE's. As more transfers are required, PE utilization decreases and bus utilization increases. Bus utilization also rises sharply at 16 PE's.

The zigzag pattern of processing speedup shown here has also recently been reported in [18] for tasks showing data unit indivisibility allocated on the multiprocessor CM*.

For this task the single bus is not a bottleneck to processing speedup. Even in cases of up to fifty PE's, no more than one bus is ever required for intermediate transfers. A small increase in processing speed and processor utilization results by using a double width bus to decrease data transfer time. A decrease in bus utilization results when a double width bus is used. The double width bus gives most advantage for high numbers of PE's, when more transfers are required.

Fig. 4.3: Effective number of PE's for the beamforming task on a multiprocessor with a double and single width bus

Fig. 4.4: Percent PE utilization for beamforming task on a multiprocessor with double and single width bus

Fig. 4.5: Percent bus utilization for the beamforming task on a multiprocessor with a double and single width bus

4.4 Implementation with Pipeline Allocation

A similar zig-zag effect is also seen for speedup of the execution time of the beamformer task for pipeline allocation of functions. Figure 4.6 shows the effective versus actual number of PE's for the task. In the pipeline arrangement a great deal more flexibility exists in choosing what function to allocate to each group of PE's and the number of PE's within each group. In this instance, function f_1 was allocated to one group and functions f_2 and f_3 were allocated to the other. The most advantageous combination of PE's for each group are plotted in figure 4.6. Not all numbers of PE's are plotted, owing to the large number of possible combinations to try. The general trend is evident for the points plotted. The cycle time minimum is reached for 26 PE's, and adding more PE's beyond that point will not increase the processing speed. The cycle time minimum here is smaller than for uniform allocation, with the maximum processing speedup being 22.1. This implementation requires two buses instead of one however to allow the group of PE's allocated to function f_1 to input data while the group assigned to f_2 and f_3 process and output data.

The jumps in processing speed still occur when the number of data units divide equally into the number of PE's allocated to the function. Owing to the smaller number of PE's within each group these jumps are not as dramatic as in the previous case for uniform allocation. A comparison of the processing speed for a given number of PE's for each type of allocation follows in the next section.

Fig. 4.6: Effective number of PE's for pipeline allocation, single wide bus, two buses used.

4.5 Comparison of Uniform and Pipeline Allocation

The question as to which arrangement to choose, pipeline or uniform allocation, will depend on the processing speed required and hardware cost. To facilitate comparison between uniform allocation with a single width bus and pipeline allocation with two single width buses, the relevant portions of figures 4.6 and 4.3 have been replotted in figure 4.7. For up to 11 PE's, no allocation has a clear superiority. From 12 to 15 PE's, pipeline allocation is preferred while from 16 to around 20 PE's uniform allocation gives the faster processing time. In each case the pipeline arrangement carries with it the extra cost of another data bus. The use of a double wide bus may be enough to meet the processing speed requirement by reducing data transfer time.

A number of trade-offs are involved in the choice of a distributed signal processing system, principally performance versus cost. Not all designs have been considered here, but the general principles of system design have been illustrated. One possibility is the use of specialized processors to do the FFT. Although necessitating the use of a pipeline arrangement (since f_3 , the beamforming task, could not presumably be done on a specialized processor) the use of such specialized processors might be required to meet stringent time requirements if general purpose PE's were not fast enough.

Fig. 4.7: Effective number of PE's for uniform versus pipeline allocation.

Chapter 5

COMPARISON OF A PULSE DOPPLER RADAR TASK ON A MULTIPROCESSOR

5.0 Reason for Comparison

As a more tutorial example where bus utilization is a constraint on processing time, it is instructive to consider the signal processing example from [4], involving a pulse Doppler radar system. In the beamforming task, data units were so large for function f_1 that reassignment of data units could not be used to advantage to reduce cycle time. Here, smaller data units may be reassigned for one PE to another in such a way as to reduce cycle time for a given number of PE's. This example will illustrate how reassignment of data units can improve cycle time for uniform allocation, and how cycle time is affected when the system is constrained to fewer buses than required for all simultaneous transfers to occur (i.e., transfers are required to queue for access to a bus).

5.1 Task Description

The task involves processing of a pulse Doppler radar waveform, having 1024 range gates each with 16 pulse returns. The processing consists of three functions:

f_1 : a 16-point complex FFT for each range gate

f_2 : magnitude of the FFT coefficients

f_3 : a sliding window average for signal detection across the 1024 range gates for each of the 16 frequencies.

Derivation of the processing time in cycles for each of the functions is given in [4]. Stated in the notation developed earlier, the task description is as follows:

$$F = (f_1, f_2, f_3) = (\text{Fourier transform, magnitude, sliding window average})$$

$$m = (m_1, m_2, m_3, m_4) = (16 \text{ complex, } 16 \text{ complex, } 1024 \text{ real, } 0) \frac{\text{points}}{\text{data unit}}$$

$$L = (L_1, L_2, L_3) = (1024, 1024, 16) \text{ data units}$$

$$\tau = (\tau_1, \tau_2, \tau_3) = (w_1 m_1 L_1, \tau_2, \tau_3, 0) \text{ where } \tau_2 = \text{normal transfer,}$$

$$\tau_3 = \text{cornerturning transfer}$$

$$c = (c_1, c_2, c_3) = (64, 48, 6144) \frac{\text{cycles}}{\text{data unit}}$$

For this comparison, a bus width of one real point was selected, so

$$w = (2, 2, 1, 0) \frac{\text{words}}{\text{point}}$$

Note that since no output is required from f_3 , $m_4 = w_4 = 0$.

5.2 Task Implementation with Uniform Allocation of Functions

Figure 5.1 illustrates the effect of bus constraint on processing speedup for the radar task outlined. The jump at 8 PE's is due to the even division of data units, as for the beamforming task. However, for greater than 16 PE's two buses are required to accomplish all the simultaneous transfers that can occur. When the system is constrained to one bus, processing speedup levels off since the decrease in processing time is offset by an increase in bus transfer time. As seen in figure 5.3, bus utilization also levels off for one bus. Processor utilization, after showing a peak for 8 PE's, continues to decrease as is seen in figure 5.2.

Fig. 5.1: Effective number of PE's for the radar task with 1 or 2 buses required

Fig. 5.2: Percent PE utilization for the radar task with 1 or 2 buses

Fig. 5.3: Percent bus utilization for the radar task with 1 or 2 buses

When two buses are available for transfer, processing speed continues to increase as is shown in figure 5.1. Similarly, although bus utilization for bus 1 levels off, the second bus begins to pick up the data transfer requirements and utilization for bus 2 continues to increase, as is shown in figure 5.2. Due to the reduction in data transfer time, processor utilization for a two bus system is higher for greater than 16 PE's.

Although not shown here, it is anticipated that three or more data buses may be required before the processing bound of one data unit per PE for function f_1 and f_2 is reached at 1024 PE's.

5.3 Cycle Time Improvement with Data Scheduling for Uniform Allocation

By using the data unit reassignment outlined in Chapter 3, it is possible to improve the processing speedup substantially. Figure 5.4 shows the graph of effective PE numbers for the radar task in cases where one or two buses are present. Much more use is made of the second bus, beginning at 4 PE's. Figure 5.5 shows the increase in processor utilization for up to 9 PE's. As the improvement process is very expensive in CPU time, only a small range of PE numbers are plotted.

Even when constrained to one bus, the data unit reassignment still leads to a large processing speed improvement. For 7 PE's, data scheduling leads to a 35% improvement for both 1 and 2 buses.

Fig. 5.4: Effective number of PE's for the radar task with data scheduling for 1 or 2 buses

Fig. 5.5: Percent PE utilization for the radar task with data scheduling for 1 or 2 buses

The improvement in scheduling of data units leads to a much smoother curve for both processor utilization and processing speed. With small sized data units, reassignment of units minimize waiting for transfers to occur. An example of the timing diagrams will be given in the next section for normal and improved allocation of data units to PE's.

Bus utilization, although not shown, shows a similar increase in utilization for rescheduling of data units.

5.4 Comparison of Improved Uniform and Pipeline Allocation

In order to show how improved data unit scheduling of data units improves cycle time over uniform division in the uniform allocation of functions, examples of timing diagrams from [15] are shown here for 8 PE's. A timing diagram for the pipeline allocation case with data scheduling is also shown. These examples do not compare exactly with the previous sections as for purposes of comparison with the examples in [4], functions f_1 and f_2 have been combined into a single function f_{12} and a double width bus was used.

From the timing information output by program Gannt, timing charts were prepared showing cases of equal and unequal data division in the uniform allocation case (figures 5.6 and 5.7 respectively) and unequal data division in the pipeline case (figure 5.8). In comparing the timing charts, it is seen that unequal division of data units in the uniform allocation case compares favorably with the pipeline case.

Fig. 5.6: Timing diagram for uniform function allocation with 8 PE's for radar task

Fig. 5.7: Timing diagram for uniform function allocation and data scheduling with 8 PE's for radar task.

Fig. 5.8: Timing diagram for pipeline function allocation and data scheduling with 8 PE's for radar task.

The cycle time for the pipeline case is slightly smaller, but the data unit division may not be optimum. For the pipeline case, improved data unit division differed only slightly from equal division and gave only a small improvement in cycle time.

The examples of timing charts shown here illustrate that data unit reassignment can make uniform allocation competitive with pipeline allocation even in cases where pipeline allocation would normally be preferred. For other numbers of PE's (for example, 7 PE's) uniform allocation would be preferred over pipeline allocation even without improvement of data unit assignment.

Chapter 6

CONCLUSION AND SUMMARY

A methodology has been described for the implementation of deterministic signal processing tasks on a multiple processor. Two signal processing tasks for sonar and radar are examined, and the function intercommunication defined in terms of normal and cornerturning transfers. In Chapter 2, a new terminology to describe unequal data division has been defined in terms of data points, words and units. Formulae for the number of words transferred between functions have been shown for normal and cornerturning transfers.

The method for computing cycle times has been described in chapter 3 using a Fortran77 program GANNT which generates timing information according to heuristic rules for data assignment and transfer. This method of computer simulation is superior to [4] in describing cycle times since bus congestion is incorporated directly and waiting time for transfers is known explicitly. Previous work in [4] used formulae to describe specific function and data assignment on various bus structures and necessitated verification of bus congestion afterwards.

In chapter 4, implementation of the beamforming task for uniform and pipeline function allocation on single and double wide buses have been investigated. No more than one bus was required to accomplish interfunction transfers. The behavior of processing speedup, PE and bus utilization was dominated by the large data unit size for the first

function. When the number of data units for the first function equally divided into the number of PE's, large increases in processing speed, PE and bus utilization resulted. Either pipeline or uniform allocation may be preferred, depending on the requirements of cycle time or response time. Uniform allocation gives a faster response time and may give a better cycle time for small numbers of PE's. Pipeline function allocation gives a better cycle time for larger numbers of PE's. For pipeline function allocation, an extra data bus for output incurs a higher hardware cost. Use of a double wide bus reduced the cycle time for larger numbers of PE's, where more transfers were required.

In chapter 5, using the example of a radar signal processing task in [4], the effect of data unit scheduling for uniform function allocation was investigated. Here, improving data unit assignment resulting in unequal data unit division decreases rather than increases the cycle time. A heuristic data unit improvement method gives an up to a 35% improvement in cycle time over equal division of data units among PE's and increased PE and bus utilization. The effect of bus congestion was investigated and it was found that processing speedup levelled off when bus congestion occurred. Provision of an extra data bus at this point enabled further processing speedup to continue. Using computer generated timing information, it was possible to determine at what number of PE's an extra data bus was required.

This thesis has presented a method by which deterministic signal processing tasks may be described and implemented on a multiple processor. The system designer is then able to make an estimate of the number of PE's required for a specific task and the effect of adding additional PE's. Frequency domain beamforming and a Doppler pulse radar signal processing task have been investigated as examples of distributed processing tasks. Generation of all the timing information via an interactive computer design program allows a wide variety and complexity of tasks to be investigated while incorporating both hardware and software constraints.

REFERENCES

- [1] W.C. Knight, R.G. Pudham and S.M. Kay, "Digital Signal Processing for Sonar", Proc. IEEE, vol. 69, pp. 1451-1506, November 1981.
- [2] T.E. Curtis and R.J. Ward, "Digital Beamforming for Sonar Systems", IEE Proc., vol. 127, Pt. F, No. 4, pp. 257-265, August 1980.
- [3] P. Barton, "Digital Beamforming for Radar", IEE Proc., vol. 127, Pt. F, No. 4, pp. 266-277, August 1980.
- [4] J.M. Class and P. Kotiveeriah, "The Allocation Problem in Distributed Signal Processing", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-29, No. 4, August 1981.
- [5] W.W. Chu, L.J. Holloway, M.T. Lan and K. Efe, "Task Allocation in Distributed Data Processing", Computer, November 1980.
- [6] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", Computer, June 1982.
- [7] H.S. Stone and S.H. Bokhari, "Control of Distributed Processes", Computer, July 1978.
- [8] T. Feng, "A Survey of Interconnection Networks", Computer, December 1981.
- [9] P.H. Enslow Jr., "Multiprocessor Organization - A Survey", Computing Surveys, vol. 9, No. 1, March 1977.
- [10] D. Protopapas and E.J. Smith, "Modelling and Analysis of Single and Multiple-Bus Multi-Microprocessor Systems", Proceedings Compcon Fall 1980, September 1980, pp. 471-478.
- [11] L.L. Kinney and R.G. Arnold, "Analysis of a Multiprocessor System with a Shared Bus", Proc. Fifth. Symp. Computer Architecture, April 1978, pp. 89-95.
- [12] A.J. Weissberger, "Analysis of Multiple-Microprocessor System Architectures", Computer Design, June 1977, pp. 151-163.
- [13] E.E. Swartzlander, Jr. "Signal Processor Design for Digital Beam Forming", Eastcon '80, September 1980, pp. 234-248.
- [14] C.V.W. Armstrong et al., "An Adaptive Multi-microprocessor Array Computing Structure for Radar Signal Processing Applications", Proc. Sixth Symp. Computer Architecture, April 1979, pp. 68-74.

- [15] C.M. Woodside and D.W. Craig, "The Effect of Finite-Sized Data Units on Function Allocation in Distributed Signal Processing", Tech. Rept. SCE-82- , November 1982.
- [16] L.R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing", Prentice-Hall, 1975, pp. 596.
- [17] D.W. Craig and C.M. Woodside, "Computation of the Crosspower Spectrum Matrix for Frequency Domain Beamforming", Tech. Rept. SCE-82-2, February 1982.
- [18] E.F. Gehringer et al., "The CM* Testbed", Computer, October 1982.


```

117: 1 NPROC,PROC,BUS NUMBER,BUSES USED,
118: 1 TIME BUS,TOTAL BUS,TOTAL PROC,START OUTPUT,
119: 1 END OUTPUT
120: 1 COMMON/CHAR/ ANSWER,ESC
121: 1
122: 1
123: 1 CHARACTER
124: 1 IFOPT, REPEAT, LOOP AGAIN
125: 1 INTEGER
126: 1 LASTUNITNO, UNITNO
127: 1 REAL
128: 1 PERCENTBUS(7), PERCENTPROC(50),SUM PROC
129: 1 LOGICAL
130: 1 FIRST
131: 1
132: 1 CALL INIT
133: 1 CALL SETUP
134: 1 FIRST=.TRUE.
135: 1 REPEAT='N'
136: 1 IFOPT='N'
137: 1
138: 1 Now do the calculation of the Gantt chart:
139: 1
140: 1 CONTINUE
141: 1
142: 1 Reassign data units or improve the assignment?
143: 1
144: 1 IF (.NOT.FIRST.AND.REPEAT.EQ.'Y') CALL REASSIGN
145: 1 IF (.NOT.FIRST.AND.IFOPT.EQ.'Y') CALL IMPROVE
146: 1
147: 1 Input the data and do the first task:
148: 1
149: 1 IF (IFOPT.NE.'Y'.AND.REPEAT.NE.'Y')
150: 1 CALL ASSIGN(1,NUNIT,NPROC(1))
151: 1 LAST UNITNO=NUNITS
152: 1 CALL INPUT DATA
153: 1 CALL DO FUNC(1)
154: 1
155: 1 Loop over all the other tasks:
156: 1
157: 1 DO 20 N=2,NFUNCS
158: 1 IF (TRANSFER(N-1)) THEN
159: 1 IF (WORDS PER POINT(N).LT.1.0) THEN
160: 1 UNITNO=
161: 1 IF (XIFLOAT(UNIT OUTPUT(N-1),WORDS PER POINT(N)+0.5)
162: 1 TIMESLEN-1.0/WORDS PER POINT(N)
163: 1 UNIT LENGTH(N)=
164: 1 IF (XIFLOAT(UNIT LENGTH(N),TIMESLEN)
165: 1 END IF
166: 1 IF (WORDS PER POINT(N).GE.1.0)
167: 1 UNITNO=UNIT OUTPUT(N-1)
168: 1 LAST UNITNO=UNIT OUTPUT(N-1)
169: 1 END IF
170: 1 IF (.NOT.TRANSFER(N-1)) UNITNO=LAST UNITNO
171: 1 IF (IFOPT.NE.'Y'.AND.REPEAT.NE.'Y')
172: 1 CALL ASSIGN(N,UNITNO,NPROC(N))
173: 1 IF (N.LE.NFUNCS) CALL TRANSFER(N-1,N)
174: 1

```

```

175: 1 CALL DO FUNC(N)
176: 1 CONTINUE
177: 1
178: 1 Output last data as required:
179: 1
180: 1 IF (LAST OUTPUT) CALL OUT
181: 1
182: 1 CALL FIND CYCLES
183: 1 CALL FIND MAXIMUM(MAX)
184: 1 WRITE (108,'(A,I,A)') , The cycle time is 'MAX,' cycles.'
185: 1
186: 1 Calculate the processor and bus loadings:
187: 1
188: 1 CALL FIND MAXIMUM(MAXPER)
189: 1 SUM PROC=0.0
190: 1 DO 30 L=1,NPROCS
191: 1 PERCENT PROC(L)=FLOAT(TOTAL PROC(L))/FLOAT(MAXPER)*100.0
192: 1 SUM PROC=SUM PROC+PERCENT PROC(L)
193: 1 CONTINUE
194: 1 SUM PROC=SUM PROC/FLOAT(NPROCS)
195: 1 IF (BUS NUMBER.EQ.0) LIMIT=BUSES USED
196: 1 IF (BUS NUMBER.GT.0) LIMIT=BUS NUMBER
197: 1 DO 35 NN=1,LIMIT
198: 1 PERCENT BUS(NN)=FLOAT(TOTAL BUS(NN))/FLOAT(MAXPER)*100.0
199: 1 CONTINUE
200: 1
201: 1 Write out the bus and processor loadings:
202: 1
203: 1 WRITE (108,1000) SUM PROC
204: 1 FORMAT(14X,' PE Utilization'//
205: 1 14X,' -----'//
206: 1 2 , The average processing element utilization = 'F8.2' Z'//)
207: 1
208: 1 WRITE (108,1200)
209: 1 FORMAT(14X,' Bus Utilization'//
210: 1 14X,' -----'//)
211: 1 DO 50 L=1,LIMIT
212: 1 WRITE (108,1300) L,PERCENT BUS(L)
213: 1 FORMAT(' Bus Number ',12,';',18X,F8.2,' Z')
214: 1 CONTINUE
215: 1 WRITE (108,'(A)') , Do you want detailed timing information?
216: 1 CALL PROMPT(' Answer Y or N.....',30)
217: 1 READ (105,'(A)') ANSWER
218: 1 IF (ANSWER.EQ.'Y') CALL RESULT
219: 1
220: 1 ask the user if he wants to reassign data units:
221: 1
222: 1 WRITE (108,'(A)') , Do you want to reassign data units?
223: 1 CALL PROMPT(' Answer Y or N.....',30)
224: 1 READ (105,'(A)') REPEAT
225: 1 IF (REPEAT.EQ.'Y') FIRST=.FALSE,100 TO 10
226: 1
227: 1 ask the user if he wants to improve:
228: 1
229: 1 WRITE (108,'(A)') , Do you want to try to improve the
230: 1 assignment of data units to the functions?
231: 1 READ (105,'(A)') IFOPT
232: 1

```

```

233: C
234: C
235: C
236: C
237: C
238: C
239: C
240: C
241: C
242: C
243: C
244: C
245: C
246: C
247: C
248: C
249: C
250: C
251: C
252: C
253: C
254: C
255: C
256: C
257: C
258: C
259: C
260: C
261: C
262: C
263: C
264: C
265: C
266: C
267: C
268: C
269: C
270: C
271: C
272: C
273: C
274: C
275: C
276: C
277: C
278: C
279: C
280: C
281: C
282: C
283: C
284: C
285: C
286: C
287: C
288: C
289: C
290: C
291: C
292: C
293: C
294: C
295: C
296: C
297: C
298: C
299: C
300: C
301: C
302: C
303: C
304: C
305: C
306: C
307: C
308: C
309: C
310: C
311: C
312: C
313: C
314: C
315: C
316: C
317: C
318: C
319: C
320: C
321: C
322: C
323: C
324: C
325: C
326: C
327: C
328: C
329: C
330: C
331: C
332: C
333: C
334: C
335: C
336: C
337: C
338: C
339: C
340: C
341: C
342: C
343: C
344: C
345: C
346: C
347: C
348: C
349: C
350: C
351: C
352: C
353: C
354: C
355: C
356: C
357: C
358: C
359: C
360: C
361: C
362: C
363: C
364: C
365: C
366: C
367: C
368: C
369: C
370: C
371: C
372: C
373: C
374: C
375: C
376: C
377: C
378: C
379: C
380: C
381: C
382: C
383: C
384: C
385: C
386: C
387: C
388: C
389: C
390: C
391: C
392: C
393: C
394: C
395: C
396: C
397: C
398: C
399: C
400: C
401: C
402: C
403: C
404: C
405: C
406: C
407: C
408: C
409: C
410: C
411: C
412: C
413: C
414: C
415: C
416: C
417: C
418: C
419: C
420: C
421: C
422: C
423: C
424: C
425: C
426: C
427: C
428: C
429: C
430: C
431: C
432: C
433: C
434: C
435: C
436: C
437: C
438: C
439: C
440: C
441: C
442: C
443: C
444: C
445: C
446: C
447: C
448: C
449: C
450: C
451: C
452: C
453: C
454: C
455: C
456: C
457: C
458: C
459: C
460: C
461: C
462: C
463: C
464: C
465: C
466: C
467: C
468: C
469: C
470: C
471: C
472: C
473: C
474: C
475: C
476: C
477: C
478: C
479: C
480: C
481: C
482: C
483: C
484: C
485: C
486: C
487: C
488: C
489: C
490: C
491: C
492: C
493: C
494: C
495: C
496: C
497: C
498: C
499: C
500: C
501: C
502: C
503: C
504: C
505: C
506: C
507: C
508: C
509: C
510: C
511: C
512: C
513: C
514: C
515: C
516: C
517: C
518: C
519: C
520: C
521: C
522: C
523: C
524: C
525: C
526: C
527: C
528: C
529: C
530: C
531: C
532: C
533: C
534: C
535: C
536: C
537: C
538: C
539: C
540: C
541: C
542: C
543: C
544: C
545: C
546: C
547: C
548: C
549: C
550: C
551: C
552: C
553: C
554: C
555: C
556: C
557: C
558: C
559: C
560: C
561: C
562: C
563: C
564: C
565: C
566: C
567: C
568: C
569: C
570: C
571: C
572: C
573: C
574: C
575: C
576: C
577: C
578: C
579: C
580: C
581: C
582: C
583: C
584: C
585: C
586: C
587: C
588: C
589: C
590: C
591: C
592: C
593: C
594: C
595: C
596: C
597: C
598: C
599: C
600: C
601: C
602: C
603: C
604: C
605: C
606: C
607: C
608: C
609: C
610: C
611: C
612: C
613: C
614: C
615: C
616: C
617: C
618: C
619: C
620: C
621: C
622: C
623: C
624: C
625: C
626: C
627: C
628: C
629: C
630: C
631: C
632: C
633: C
634: C
635: C
636: C
637: C
638: C
639: C
640: C
641: C
642: C
643: C
644: C
645: C
646: C
647: C
648: C
649: C
650: C
651: C
652: C
653: C
654: C
655: C
656: C
657: C
658: C
659: C
660: C
661: C
662: C
663: C
664: C
665: C
666: C
667: C
668: C
669: C
670: C
671: C
672: C
673: C
674: C
675: C
676: C
677: C
678: C
679: C
680: C
681: C
682: C
683: C
684: C
685: C
686: C
687: C
688: C
689: C
690: C
691: C
692: C
693: C
694: C
695: C
696: C
697: C
698: C
699: C
700: C
701: C
702: C
703: C
704: C
705: C
706: C
707: C
708: C
709: C
710: C
711: C
712: C
713: C
714: C
715: C
716: C
717: C
718: C
719: C
720: C
721: C
722: C
723: C
724: C
725: C
726: C
727: C
728: C
729: C
730: C
731: C
732: C
733: C
734: C
735: C
736: C
737: C
738: C
739: C
740: C
741: C
742: C
743: C
744: C
745: C
746: C
747: C
748: C
749: C
750: C
751: C
752: C
753: C
754: C
755: C
756: C
757: C
758: C
759: C
760: C
761: C
762: C
763: C
764: C
765: C
766: C
767: C
768: C
769: C
770: C
771: C
772: C
773: C
774: C
775: C
776: C
777: C
778: C
779: C
780: C
781: C
782: C
783: C
784: C
785: C
786: C
787: C
788: C
789: C
790: C
791: C
792: C
793: C
794: C
795: C
796: C
797: C
798: C
799: C
800: C
801: C
802: C
803: C
804: C
805: C
806: C
807: C
808: C
809: C
810: C
811: C
812: C
813: C
814: C
815: C
816: C
817: C
818: C
819: C
820: C
821: C
822: C
823: C
824: C
825: C
826: C
827: C
828: C
829: C
830: C
831: C
832: C
833: C
834: C
835: C
836: C
837: C
838: C
839: C
840: C
841: C
842: C
843: C
844: C
845: C
846: C
847: C
848: C
849: C
850: C
851: C
852: C
853: C
854: C
855: C
856: C
857: C
858: C
859: C
860: C
861: C
862: C
863: C
864: C
865: C
866: C
867: C
868: C
869: C
870: C
871: C
872: C
873: C
874: C
875: C
876: C
877: C
878: C
879: C
880: C
881: C
882: C
883: C
884: C
885: C
886: C
887: C
888: C
889: C
890: C
891: C
892: C
893: C
894: C
895: C
896: C
897: C
898: C
899: C
900: C
901: C
902: C
903: C
904: C
905: C
906: C
907: C
908: C
909: C
910: C
911: C
912: C
913: C
914: C
915: C
916: C
917: C
918: C
919: C
920: C
921: C
922: C
923: C
924: C
925: C
926: C
927: C
928: C
929: C
930: C
931: C
932: C
933: C
934: C
935: C
936: C
937: C
938: C
939: C
940: C
941: C
942: C
943: C
944: C
945: C
946: C
947: C
948: C
949: C
950: C
951: C
952: C
953: C
954: C
955: C
956: C
957: C
958: C
959: C
960: C
961: C
962: C
963: C
964: C
965: C
966: C
967: C
968: C
969: C
970: C
971: C
972: C
973: C
974: C
975: C
976: C
977: C
978: C
979: C
980: C
981: C
982: C
983: C
984: C
985: C
986: C
987: C
988: C
989: C
990: C
991: C
992: C
993: C
994: C
995: C
996: C
997: C
998: C
999: C
1000: C

```

IF (IFOPT.EQ.'Y') FIRST=.FALSE.; GO TO 10
 ask the user if he wants to try again:
 WRITE(106,'(A)') ' Want to try another signal processing task?'
 READ(105,'(A)') LOOP AGAIN
 IF (LOOP AGAIN.EQ.'Y') ANSWER='A'; GO TO 1
 END
 TOTAL ERRORS FOUND: 0

SUBROUTINE INIT
 Subroutine Init
 Purpose: To initialize all the variables in the common area
 to a default value.
 Standard Common Block
 Maxium number of processing elements = 50
 Maxium number of functions = 10
 Maxium number of buses = 7
 INTEGER NPROCS,NFUNCS,UNIT INPUT,MUNITS
 REAL WORDS PER POINT(11)
 INTEGER UNIT OUTPUT(10),UNIT LENGTH(10),INPUT AMOUNT(50)
 INTEGER CYCLE START,INPUT START(50),INPUT END(50)
 INTEGER FUNC LENGTH(10,50),FUNC START(10,50),FUNC END(10,50)
 INTEGER ALLOCATED(10,50),START TRANSFER(50,50,10)
 INTEGER END TRANSFER(50,50,10)
 INTEGER CYCLE BEGIN(50),CYCLE END(50),CYCLE PERIOD(50)
 INTEGER BUS NUMBER,BUSES USED,TIME BUS(7),TOTAL BUS(7)
 INTEGER TOTAL PROC(50),START OUTPUT(50),END OUTPUT(50)
 LOGICAL TRANSPOSE(10)
 LOGICAL CYBERNEX,LAST OUTPUT
 CHARACTER ANSWER
 CHARACTER ESC
 INTEGER NPROC(10),PROC(10,50)
 COMMON/SHARED/ NPROCS,NFUNCS,UNIT INPUT,MUNITS,WORDS PER POINT,
 CYCLE START,INPUT START,INPUT END,
 FUNC LENGTH,FUNC START,FUNC END,
 ALLOCATED,START TRANSFER,END TRANSFER,
 UNIT OUTPUT,UNIT LENGTH,INPUT AMOUNT,
 TRANSPOSE,CYBERNEX,LAST OUTPUT,
 CYCLE BEGIN,CYCLE END,CYCLE PERIOD,
 NPROC,PROC,BUS NUMBER,BUSES USED,
 TIME BUS,TOTAL BUS,TOTAL PROC,START OUTPUT,
 END OUTPUT
 COMMON/CHAR/ ANSWER,ESC
 DO 10 L=1,10
 UNIT OUTPUT(L)=0
 UNIT LENGTH(L)=0
 TRANSPOSE(L)=.FALSE.
 NPROC(L)=0
 CONTINUE
 **** Set all Common variables ****
 DO 10 L=1,10
 UNIT OUTPUT(L)=0
 UNIT LENGTH(L)=0
 TRANSPOSE(L)=.FALSE.
 NPROC(L)=0
 CONTINUE

2 2
OF / DE

```

59: C
60: DO 20 N=1,50
61:   INPUT AMOUNT(N)=0
62:   INPUT START(N)=0
63:   INPUT END(N)=0
64:   CYCLE BEGIN(N)=0
65:   CYCLE END(N)=0
66:   CYCLE PERIOD(N)=0
67:   TOTAL PROC(N)=0
68:   START OUTPUT(N)=0
69:   END OUTPUT(N)=0
70: CONTINUE
71: C
72: C
73: DO 30 N=1,7
74:   TIME BUS(N)=0
75:   TOTAL BUS(N)=0
76: CONTINUE
77: C
78: C
79: DO 40 I=1,11
80:   WORDS PER POINT(I)=0.0
81: CONTINUE
82: C
83: C
84: DO 60 I=1,10
85:   DO 70 J=1,50
86:     FUNC LENGTH(I,J)=0
87:     FUNC START(I,J)=0
88:     FUNC END(I,J)=0
89:     ALLOCATED(I,J)=0
90: CONTINUE
91: C
92: C
93: C
94: DO 80 I=1,50
95:   DO 90 J=1,50
96:     DO 100 K=1,10
97:       START TRANSFER(I,J,K)=0
98:       END TRANSFER(I,J,K)=0
99: CONTINUE
100: C
101: C
102: C
103: C
104: ANSWER='NO'
105: NPROC=0
106: RFUNC=0
107: UNIT INPUT=0
108: NUNITS=0
109: CYCLE START=1
110: LAST OUTPUT=.FALSE.
111: BUS NUMBER=0
112: BUSES USED=0
113: C
114: RETURN
115: END TOTAL ERRORS FOUND: 0

```

***** SUBROUTINE SETUP *****
 ***** Subroutine Setup *****

 Purpose: To input all the required information for Program
 Gaint.
 The program is structured so on a second pass
 through the values can be defaulted.

***** Standard Common Block *****

 Maximum number of processing elements = 50
 Maximum number of functions = 10
 Maximum number of buses = 7

 INTEGER NPROC, NFUNC, UNIT INPUT, NUNITS
 REAL WORDS PER POINT(11)
 INTEGER UNIT OUTPUT(10), UNIT LENGTH(10), INPUT AMOUNT(50)
 INTEGER CYCLE START, INPUT START(50), INPUT END(50)
 INTEGER FUNC LENGTH(10,50), FUNC START(10,50), FUNC END(10,50)
 INTEGER ALLOCATED(10,50), START TRANSFER(50,50,10)
 INTEGER END TRANSFER(50,50,10)
 INTEGER CYCLE BEGIN(50), CYCLE END(50), CYCLE PERIOD(50)
 INTEGER BUS NUMBER, BUSES USED, TIME BUS(7), TOTAL BUS(7)
 INTEGER TOTAL PROC(50), START OUTPUT(50), END OUTPUT(50)
 LOGICAL TRANSPOSE(10)
 LOGICAL CYBERNEX, LAST OUTPUT
 CHARACTER ANSWER
 CHARACTER ESC

 INTEGER NPROC(10), PROC(10,50)
 COMMON/SHARED/ NPROC, NFUNC, UNIT INPUT, NUNITS, WORDS PER POINT,
 CYCLE START, INPUT START, INPUT END,
 FUNC LENGTH, FUNC START, FUNC END,
 ALLOCATED, START TRANSFER, END TRANSFER,
 UNIT OUTPUT, UNIT LENGTH, INPUT AMOUNT,
 TRANSPOSE, CYBERNEX, LAST OUTPUT,
 CYCLE BEGIN, CYCLE END, CYCLE PERIOD,
 NPROC, PROC, BUS NUMBER, BUSES USED,
 TIME BUS, TOTAL BUS, TOTAL PROC, START OUTPUT,
 END OUTPUT
 COMMON/CHAR/ ANSWER, ESC

 INTEGER
 1 PROCESSOR(50), IDUMMY
 REAL RDUMMY
 CHARACTER ADUMMY
 LOGICAL LDUMMY
 ***** Zero out local array *****

```

59: C
60: DO 7 LL=1,50
61: DO 6 MM=1,10
62: PROCESSORS(LL)=0
63: PROG(MM,LL)=0
64: CONTINUE
65: 7
66: C
67: ESC=CHAR(27)
68: IF (ANSWER.NE.O) CYBERNEX=.FALSE.
69: IF (ANSWER.NE.A)
70: CALL PROMPT(' Is this terminal a CYBERNEX? ',30)
71: READ (105,'(A)') ANSWER=ADUMHY
72: IF (ADUMHY.EQ.'Y') CYBERNEX=.TRUE.
73: IF (CYBERNEX) WRITE (108,'(3A1)') ' ',ESC,'E'
74: IF (CYBERNEX) WRITE (108,1000) ESC,'h', '*** PROGRAM GANNT ***',
75: 1000
76: 1000
77: FORMAT(130A1,A1,A1,A1)
78: IF (.NOT.CYBERNEX) WRITE (108,'(130A1)') '*** PROGRAM GANNT ***'
79: WRITE (108,'(//A)') ' Parameter Input:'
80: WRITE (108,'(//A)') ' General information for all functions:'
81: WRITE (108,'(//A/A//)')
82: 1
83: 1
84: 1
85: 1
86: 1
87: CALL PROMPT(' No. of processing elements = 50',
88: 105,'(I1)') IDUMHY
89: CALL PROMPT(' No. of functions? .....',30)
90: READ (105,'(I1)') NFUNCS=IDUMHY
91: CALL PROMPT(' # points/data unit in input? ',30)
92: READ (105,'(I1)') IDUMHY
93: IF (IDUMHY.NE.O) UNIT INPUT=IDUMHY
94: CALL PROMPT(' Number of data units input? ',30)
95: READ (105,'(I1)') IDUMHY
96: IF (IDUMHY.NE.O) NUNITS=IDUMHY
97: CALL PROMPT(' # of words/point for input? ',30)
98: READ (105,'(E)') IDUMHY
99: IF (IDUMHY.GT.O.O) WORDS PER POINT(1)=IDUMHY
100: IF (IDUMHY.GT.O.O) UNIT INPUTS
101: INT(FLOAT(UNIT INPUT)/WORDS PER POINT(1)+0.5)
102: 1
103: 1
104: 1
105: C C C C
106: LAST OUTPUT=.FALSE.
107: DO 50 LL=NFUNCS
108: IF (CYBERNEX) WRITE (108,'(3A1)') ' ',ESC,'E'
109: IF (CYBERNEX) WRITE (108,1000) ESC,'h', '*** PROGRAM GANNT ***',
110: 1000
111: 1000
112: IF (CYBERNEX) WRITE (108,'(//A)')
113: WRITE (108,'(//A//)') ' Specific information for function #'
114: 1
115: CALL PROMPT(' # cycles for each data unit? ',30)
116: READ (105,'(I1)') IDUMHY
117: IF (IDUMHY.NE.O) UNIT LENGTH(L)=IDUMHY
118: CALL PROMPT(' # points output data unit? ',30)
119: READ (105,'(I1)') IDUMHY
120: IF (IDUMHY.NE.O) BUS NUMBER=IDUMHY
121: 1
122: 1
123: 1
124: 1
125: 1
126: 1
127: 1
128: 1
129: 1
130: 1
131: 1
132: 1
133: 1
134: 1
135: 1
136: 1
137: 1
138: 1
139: 1
140: 1
141: 1
142: 1
143: 1
144: 1
145: 1
146: 1
147: 1
148: 1
149: 1
150: 1
151: 1
152: 1
153: 1
154: 1
155: 1
156: 1
157: 1
158: 1
159: 1
160: 1
161: 1
162: 1
163: 1
164: 1
165: 1
166: 1
167: 1
168: 1
169: 1
170: 1
171: 1
172: 1
173: 1
174: 1
175: 1
176: 1
177: 1
178: 1
179: 1
180: 1
181: 1
182: 1
183: 1
184: 1
185: 1
186: 1
187: 1
188: 1
189: 1
190: 1
191: 1
192: 1
193: 1
194: 1
195: 1
196: 1
197: 1
198: 1
199: 1
200: 1
201: 1
202: 1
203: 1
204: 1
205: 1
206: 1
207: 1
208: 1
209: 1
210: 1
211: 1
212: 1
213: 1
214: 1
215: 1
216: 1
217: 1
218: 1
219: 1
220: 1
221: 1
222: 1
223: 1
224: 1
225: 1
226: 1
227: 1
228: 1
229: 1
230: 1
231: 1
232: 1
233: 1
234: 1
235: 1
236: 1
237: 1
238: 1
239: 1
240: 1
241: 1
242: 1
243: 1
244: 1
245: 1
246: 1
247: 1
248: 1
249: 1
250: 1
251: 1
252: 1
253: 1
254: 1
255: 1
256: 1
257: 1
258: 1
259: 1
260: 1
261: 1
262: 1
263: 1
264: 1
265: 1
266: 1
267: 1
268: 1
269: 1
270: 1
271: 1
272: 1
273: 1
274: 1
275: 1
276: 1
277: 1
278: 1
279: 1
280: 1
281: 1
282: 1
283: 1
284: 1
285: 1
286: 1
287: 1
288: 1
289: 1
290: 1
291: 1
292: 1
293: 1
294: 1
295: 1
296: 1
297: 1
298: 1
299: 1
300: 1
301: 1
302: 1
303: 1
304: 1
305: 1
306: 1
307: 1
308: 1
309: 1
310: 1
311: 1
312: 1
313: 1
314: 1
315: 1
316: 1
317: 1
318: 1
319: 1
320: 1
321: 1
322: 1
323: 1
324: 1
325: 1
326: 1
327: 1
328: 1
329: 1
330: 1
331: 1
332: 1
333: 1
334: 1
335: 1
336: 1
337: 1
338: 1
339: 1
340: 1
341: 1
342: 1
343: 1
344: 1
345: 1
346: 1
347: 1
348: 1
349: 1
350: 1
351: 1
352: 1
353: 1
354: 1
355: 1
356: 1
357: 1
358: 1
359: 1
360: 1
361: 1
362: 1
363: 1
364: 1
365: 1
366: 1
367: 1
368: 1
369: 1
370: 1
371: 1
372: 1
373: 1
374: 1
375: 1
376: 1
377: 1
378: 1
379: 1
380: 1
381: 1
382: 1
383: 1
384: 1
385: 1
386: 1
387: 1
388: 1
389: 1
390: 1
391: 1
392: 1
393: 1
394: 1
395: 1
396: 1
397: 1
398: 1
399: 1
400: 1
401: 1
402: 1
403: 1
404: 1
405: 1
406: 1
407: 1
408: 1
409: 1
410: 1
411: 1
412: 1
413: 1
414: 1
415: 1
416: 1
417: 1
418: 1
419: 1
420: 1
421: 1
422: 1
423: 1
424: 1
425: 1
426: 1
427: 1
428: 1
429: 1
430: 1
431: 1
432: 1
433: 1
434: 1
435: 1
436: 1
437: 1
438: 1
439: 1
440: 1
441: 1
442: 1
443: 1
444: 1
445: 1
446: 1
447: 1
448: 1
449: 1
450: 1
451: 1
452: 1
453: 1
454: 1
455: 1
456: 1
457: 1
458: 1
459: 1
460: 1
461: 1
462: 1
463: 1
464: 1
465: 1
466: 1
467: 1
468: 1
469: 1
470: 1
471: 1
472: 1
473: 1
474: 1
475: 1
476: 1
477: 1
478: 1
479: 1
480: 1
481: 1
482: 1
483: 1
484: 1
485: 1
486: 1
487: 1
488: 1
489: 1
490: 1
491: 1
492: 1
493: 1
494: 1
495: 1
496: 1
497: 1
498: 1
499: 1
500: 1
501: 1
502: 1
503: 1
504: 1
505: 1
506: 1
507: 1
508: 1
509: 1
510: 1
511: 1
512: 1
513: 1
514: 1
515: 1
516: 1
517: 1
518: 1
519: 1
520: 1
521: 1
522: 1
523: 1
524: 1
525: 1
526: 1
527: 1
528: 1
529: 1
530: 1
531: 1
532: 1
533: 1
534: 1
535: 1
536: 1
537: 1
538: 1
539: 1
540: 1
541: 1
542: 1
543: 1
544: 1
545: 1
546: 1
547: 1
548: 1
549: 1
550: 1
551: 1
552: 1
553: 1
554: 1
555: 1
556: 1
557: 1
558: 1
559: 1
560: 1
561: 1
562: 1
563: 1
564: 1
565: 1
566: 1
567: 1
568: 1
569: 1
570: 1
571: 1
572: 1
573: 1
574: 1
575: 1
576: 1
577: 1
578: 1
579: 1
580: 1
581: 1
582: 1
583: 1
584: 1
585: 1
586: 1
587: 1
588: 1
589: 1
590: 1
591: 1
592: 1
593: 1
594: 1
595: 1
596: 1
597: 1
598: 1
599: 1
600: 1
601: 1
602: 1
603: 1
604: 1
605: 1
606: 1
607: 1
608: 1
609: 1
610: 1
611: 1
612: 1
613: 1
614: 1
615: 1
616: 1
617: 1
618: 1
619: 1
620: 1
621: 1
622: 1
623: 1
624: 1
625: 1
626: 1
627: 1
628: 1
629: 1
630: 1
631: 1
632: 1
633: 1
634: 1
635: 1
636: 1
637: 1
638: 1
639: 1
640: 1
641: 1
642: 1
643: 1
644: 1
645: 1
646: 1
647: 1
648: 1
649: 1
650: 1
651: 1
652: 1
653: 1
654: 1
655: 1
656: 1
657: 1
658: 1
659: 1
660: 1
661: 1
662: 1
663: 1
664: 1
665: 1
666: 1
667: 1
668: 1
669: 1
670: 1
671: 1
672: 1
673: 1
674: 1
675: 1
676: 1
677: 1
678: 1
679: 1
680: 1
681: 1
682: 1
683: 1
684: 1
685: 1
686: 1
687: 1
688: 1
689: 1
690: 1
691: 1
692: 1
693: 1
694: 1
695: 1
696: 1
697: 1
698: 1
699: 1
700: 1
701: 1
702: 1
703: 1
704: 1
705: 1
706: 1
707: 1
708: 1
709: 1
710: 1
711: 1
712: 1
713: 1
714: 1
715: 1
716: 1
717: 1
718: 1
719: 1
720: 1
721: 1
722: 1
723: 1
724: 1
725: 1
726: 1
727: 1
728: 1
729: 1
730: 1
731: 1
732: 1
733: 1
734: 1
735: 1
736: 1
737: 1
738: 1
739: 1
740: 1
741: 1
742: 1
743: 1
744: 1
745: 1
746: 1
747: 1
748: 1
749: 1
750: 1
751: 1
752: 1
753: 1
754: 1
755: 1
756: 1
757: 1
758: 1
759: 1
760: 1
761: 1
762: 1
763: 1
764: 1
765: 1
766: 1
767: 1
768: 1
769: 1
770: 1
771: 1
772: 1
773: 1
774: 1
775: 1
776: 1
777: 1
778: 1
779: 1
780: 1
781: 1
782: 1
783: 1
784: 1
785: 1
786: 1
787: 1
788: 1
789: 1
790: 1
791: 1
792: 1
793: 1
794: 1
795: 1
796: 1
797: 1
798: 1
799: 1
800: 1
801: 1
802: 1
803: 1
804: 1
805: 1
806: 1
807: 1
808: 1
809: 1
810: 1
811: 1
812: 1
813: 1
814: 1
815: 1
816: 1
817: 1
818: 1
819: 1
820: 1
821: 1
822: 1
823: 1
824: 1
825: 1
826: 1
827: 1
828: 1
829: 1
830: 1
831: 1
832: 1
833: 1
834: 1
835: 1
836: 1
837: 1
838: 1
839: 1
840: 1
841: 1
842: 1
843: 1
844: 1
845: 1
846: 1
847: 1
848: 1
849: 1
850: 1
851: 1
852: 1
853: 1
854: 1
855: 1
856: 1
857: 1
858: 1
859: 1
860: 1
861: 1
862: 1
863: 1
864: 1
865: 1
866: 1
867: 1
868: 1
869: 1
870: 1
871: 1
872: 1
873: 1
874: 1
875: 1
876: 1
877: 1
878: 1
879: 1
880: 1
881: 1
882: 1
883: 1
884: 1
885: 1
886: 1
887: 1
888: 1
889: 1
890: 1
891: 1
892: 1
893: 1
894: 1
895: 1
896: 1
897: 1
898: 1
899: 1
900: 1
901: 1
902: 1
903: 1
904: 1
905: 1
906: 1
907: 1
908: 1
909: 1
910: 1
911: 1
912: 1
913: 1
914: 1
915: 1
916: 1
917: 1
918: 1
919: 1
920: 1
921: 1
922: 1
923: 1
924: 1
925: 1
926: 1
927: 1
928: 1
929: 1
930: 1
931: 1
932: 1
933: 1
934: 1
935: 1
936: 1
937: 1
938: 1
939: 1
940: 1
941: 1
942: 1
943: 1
944: 1
945: 1
946: 1
947: 1
948: 1
949: 1
950: 1
951: 1
952: 1
953: 1
954: 1
955: 1
956: 1
957: 1
958: 1
959: 1
960: 1
961: 1
962: 1
963: 1
964: 1
965: 1
966: 1
967: 1
968: 1
969: 1
970: 1
971: 1
972: 1
973: 1
974: 1
975: 1
976: 1
977: 1
978: 1
979: 1
980: 1
981: 1
982: 1
983: 1
984: 1
985: 1
986: 1
987: 1
988: 1
989: 1
990: 1
991: 1
992: 1
993: 1
994: 1
995: 1
996: 1
997: 1
998: 1
999: 1
1000: 1

```

1


```

117: C
118: C
119: C
120: C
121: C
122: C
123: C
124: C
125: C
126: C
127: C
128: C
129: C
130: C
131: C
132: C
133: C
134: C
135: C
136: C
137: C
138: C
139: C
140: C
141: C
142: C
143: C
144: C
145: C
146: C
147: C
148: C
149: C
150: C
151: C
152: C
153: C
154: C
155: C
156: C
157: C
158: C
159: C
160: C
161: C
162: C
163: C
164: C
165: C
166: C
167: C
168: C
169: C
170: C
171: C
172: C
173: C
174: C

*** Output times, if required ***
IF (LAST OUTPUT) THEN
  WRITE (108,2105)
  FORMAT(/14X, Output //
1 14X, Start //
2 14X, End //
2 14X, //)
  DO 75 L=1,NPROCS
    WRITE (108,2205) L, START OUTPUT(L), END OUTPUT(L)
    FORMAT(PE,12,1,112,113)
  CONTINUE
  READ (105,111) IDUMMY
  END IF

Print out the cycle times and cycle periods
WRITE (108,2300)
FORMAT(114, Cycle //
1 114, Start //
2 114, End //
2 114, //)
  DO 80 L=1,NPROCS
    WRITE (108,2400) L, CYCLE BEGIN(L), CYCLE END(L), CYCLE PERIOD(L)
    FORMAT(PE,12,1,7X,110,113,113)
  CONTINUE
  READ (105,111) IDUMMY

Calculate the processor and bus loadings:
CALL FIND MAXIMUM(MAXPER)
DO 90 L=1,NPROCS
  PERCENT PROC(L)=FLOAT(TOTAL PROC(L))/FLOAT(MAXPER)*100.0
  CONTINUE
  IF (BUS NUMBER.EQ.0) LIMIT=BUSES USED
  IF (BUS NUMBER.GT.0) LIMIT=BUS NUMBER
  DO 100 NH=1,LIMIT
    PERCENT BUS(NH)=FLOAT(TOTAL BUS(NH))/FLOAT(MAXPER)*100.0
  CONTINUE

Write out the bus and processor loadings:
WRITE (108,2500)
FORMAT(14X, PE Element Utilization//
1 14X, //)
  DO 110 L=1,NPROCS
    WRITE (108,2600) L, PERCENT PROC(L)
    FORMAT(PE,12,1,20X,F8.2, X)
  CONTINUE
  WRITE (108,2700)
  FORMAT(14X, Bus Utilization//
1 14X, //)
  DO 120 L=1,LIMIT
    WRITE (108,2800) L, PERCENT BUS(L)
    FORMAT(Bus Number,12,1,18X,F8.2, X)
  CONTINUE
  RETURN

```

175: END TOTAL ERRORS FOUND: 0

[illegible]


```

175:      END IF  

176:    END IF  

177:  CONTINUE  

178:  ***** Find a bus to do the transfer ****  

179:  

180:  IF (BUS NUMBER=0.O.OR.BUSES USED.LT,BUS NUMBER) THEN  

181:    END TRANSFER(L SAVE,M SAVE,FUNC)=START TRANSFER(L SAVE,  

182:    M SAVE,FUNC)+AMOUNT SAVE -1  

183:    DO 60 K-1,BUSES USED  

184:    IF (EAGLEST TRANSFER.GT.,TIME BUS(K)) THEN  

185:      .AND..NOT.(DONE TRANSFER(L SAVE,M SAVE)) THEN  

186:        TIME BUS(K)=END TRANSFER(L SAVE,M SAVE,FUNC)  

187:        TOTAL BUS(K)=TOTAL BUS(K)+AMOUNT SAVE  

188:        LAST BUSY(L SAVE)=TIME BUS(K)  

189:        DONE TRANSFER(L SAVE,M SAVE)=.TRUE.  

190:        AMOUNT TRANSFER=ALLOCATED(FUNC,M SAVE)  

191:        ALLOCATED(NEXT FUNC,L SAVE)  

192:        IF (WORDS PER POINT(NEXT FUNC).GT.,1.O)  

193:          AMOUNT TRANSFER=IFIX(FLDAT(AMOUNT TRANSFER)*  

194:            WORDS PER POINT(NEXT FUNC))  

195:        START TRANSFER(M SAVE,L SAVE,FUNC)=TIME BUS(K)+1  

196:        END TRANSFER(M SAVE,L SAVE,FUNC)  

197:        =START TRANSFER(M SAVE,L SAVE,FUNC)  

198:        +AMOUNT TRANSFER -1  

199:        TIME BUS(K)=END TRANSFER(M SAVE,L SAVE,FUNC)  

200:        TOTAL BUS(K)=TOTAL BUS(K)+AMOUNT TRANSFER  

201:        LAST BUSY(M SAVE)=TIME BUS(K)  

202:        DONE TRANSFER(M SAVE,L SAVE)=.TRUE.  

203:      END IF  

204:    END IF  

205:  CONTINUE  

206:  IF (.NOT.DONE TRANSFER(L SAVE,M SAVE)) THEN  

207:    BUSES USED=BUSES USED+1  

208:    IF (BUSES USED.GT.,7) THEN  

209:      WRITE (108,'(A)', * Too many buses required'  

210:      STOP  

211:    ELSE  

212:      WRITE (108,'(A)', * ** PROGRAM TERMINATED ***'  

213:      STOP  

214:    END IF  

215:  END IF  

216:  TIME BUS(BUSES USED)=END TRANSFER(L SAVE,M SAVE,FUNC)  

217:  TOTAL BUS(BUSES USED)=TOTAL BUS(BUSES USED)  

218:  +AMOUNT SAVE  

219:  LAST BUSY(L SAVE)=TIME BUS(BUSES USED)  

220:  DONE TRANSFER(L SAVE,M SAVE)=.TRUE.  

221:  AMOUNT TRANSFER=ALLOCATED(FUNC,M SAVE)  

222:  ALLOCATED(NEXT FUNC,L SAVE)  

223:  IF (WORDS PER POINT(NEXT FUNC).GT.,1.O)  

224:    AMOUNT TRANSFER=IFIX(FLDAT(AMOUNT TRANSFER)*  

225:      WORDS PER POINT(NEXT FUNC))  

226:  START TRANSFER(M SAVE,L SAVE,FUNC)  

227:  =TIME BUS(BUSES USED)+1  

228:  END TRANSFER(M SAVE,L SAVE,FUNC)  

229:  *****  

230:  *****  

231:  *****  

232:  *****  

233:  *****

```



```

349: 1      .AND.NET TRANSFER(L.SAVE).GT.0) THEN
350: 1      TIME BUS(K)=END TRANSFER(L.SAVE,M.SAVE,FUNC)
351: 1      TOTAL BUS(K)=TOTAL BUS(K)+AMOUNT SAVE
352: 1      LAST BUS(TL.SAVE)=TIME BUS(K)
353: 1      NET TRANSFER(L.SAVE)=NET TRANSFER(L.SAVE)
354: 1      -AMOUNT SAVE
355: 1      NET TRANSFER(M.SAVE)=NET TRANSFER(M.SAVE)
356: 1      +AMOUNT SAVE
357: 1      END IF
358: 1      CONTINUE
359: 160      IF (NET TRANSFER(L.SAVE).GT.0) THEN
360: 160      BUSES USED=BUSES USED+1
361: 160      IF (BUSES USED.GT.7) THEN
362: 160      WRITE (108,'(A)') ' Too many Buses required'
363: 160      WRITE (108,'(A)') ' *** PROGRAM TERMINATED ***'
364: 160      STOP
365: 160      END IF
366: 160      TIME BUS(BUSES USED)=END TRANSFER(L.SAVE,M.SAVE,FUNC)
367: 160      TOTAL BUS(BUSES USED)=TOTAL BUS(BUSES USED)
368: 160      +AMOUNT SAVE
369: 160      LAST BUS(TL.SAVE)=TIME BUS(BUSES USED)
370: 160      NET TRANSFER(L.SAVE)=NET TRANSFER(L.SAVE)
371: 160      -AMOUNT SAVE
372: 160      NET TRANSFER(M.SAVE)=NET TRANSFER(M.SAVE)
373: 160      +AMOUNT SAVE
374: 160      END IF
375: 160      ELSE IF (BUS NUMBER.GT.0.AND.BUSES USED.EQ.BUS NUMBER) THEN
376: 160      END TRANSFER(L.SAVE,M.SAVE,FUNC)=START TRANSFER(L.SAVE,
377: 160      M.SAVE,FUNC)+AMOUNT SAVE -1
378: 160      DO 170 K=1,BUSES USED
379: 160      IF (EARLIEST TRANSFER.GT.TIME BUS(K)
380: 160      .AND.NET TRANSFER(L.SAVE).GT.0) THEN
381: 160      TIME BUS(K)=END TRANSFER(L.SAVE,M.SAVE,FUNC)
382: 160      TOTAL BUS(K)=TOTAL BUS(K)+AMOUNT SAVE
383: 160      LAST BUS(TL.SAVE)=TIME BUS(K)
384: 160      NET TRANSFER(L.SAVE)=NET TRANSFER(L.SAVE)
385: 160      -AMOUNT SAVE
386: 160      NET TRANSFER(M.SAVE)=NET TRANSFER(M.SAVE)
387: 160      +AMOUNT SAVE
388: 160      END IF
389: 160      END IF
390: 160      CONTINUE
391: 170      IF (NET TRANSFER(L.SAVE).GT.0) THEN
392: 170      FIRST=.TRUE.
393: 170      DO 180 K=1,BUS NUMBER
394: 170      IF (FIRST) NEXT TIME=TIME BUS(K)/FIRST+.FALSE.
395: 170      IF (.NOT.FIRST) THEN
396: 170      IF (TIME BUS(K).LT.NEXT TIME) THEN
397: 170      NEXT TIME=TIME BUS(K)
398: 170      NEXT BUS=K
399: 170      END IF
400: 170      CONTINUE
401: 170      START TRANSFER(L.SAVE,M.SAVE,FUNC)= TIME BUS(NEXT BUS)
402: 170      END TRANSFER(L.SAVE,M.SAVE,FUNC)
403: 170
404: 170
405: 170
406: 170
407: 1      *START TRANSFER(L.SAVE,M.SAVE,FUNC)+AMOUNT SAVE
408: 1      NET TRANSFER(L.SAVE)=NET TRANSFER(L.SAVE)
409: 1      -AMOUNT SAVE
410: 1      TIME BUS(NEXT BUS)=END TRANSFER(L.SAVE,M.SAVE,FUNC)
411: 1      TOTAL BUS(NEXT BUS)=TOTAL BUS(NEXT BUS)+AMOUNT SAVE
412: 1      LAST BUS(TL.SAVE)=TIME BUS(NEXT BUS)
413: 1      NET TRANSFER(L.SAVE)=NET TRANSFER(L.SAVE)
414: 1      -AMOUNT SAVE
415: 1      NET TRANSFER(M.SAVE)=NET TRANSFER(M.SAVE)
416: 1      +AMOUNT SAVE
417: 1      END IF
418: 1      END IF
419: 1      *** Find out if all the transfers are completed ***
420: 1      MORE TRANSFERS=.FALSE.
421: 1      DO 190 LL=1,NPROCS
422: 1      IF (NET TRANSFER(LL).GT.0) MORE TRANSFERS=.TRUE.
423: 1      CONTINUE
424: 1      *** Reset the start transfer times ***
425: 1      DO 195 LL=1,NPROCS
426: 1      DD 197 MM=1,NPROCS
427: 1      IF (START TRANSFER(LL,MM,FUNC).GT.0.AND.
428: 1      END TRANSFER(LL,MM,FUNC).EQ.0)
429: 1      START TRANSFER(LL,MM,FUNC)=0
430: 1      CONTINUE
431: 1      CONTINUE
432: 1      END IF
433: 1      RETURN
434: 1      END TOTAL ERRORS FOUND: 0
435: 1
436: 1
437: 1
438: 1
439: 1
440: 1
441: 1
442: 1
443: 1
444: 1
445: 1
446: 1
447: 1
448: 1
449: 1
450: 1
451: 1
452: 1
453: 1
454: 1
455: 1
456: 1
457: 1
458: 1
459: 1
460: 1
461: 1
462: 1
463: 1
464: 1
465: 1
466: 1
467: 1
468: 1
469: 1
470: 1
471: 1
472: 1
473: 1
474: 1
475: 1
476: 1
477: 1
478: 1
479: 1
480: 1
481: 1
482: 1
483: 1
484: 1
485: 1
486: 1
487: 1
488: 1
489: 1
490: 1
491: 1
492: 1
493: 1
494: 1
495: 1
496: 1
497: 1
498: 1
499: 1
500: 1
501: 1
502: 1
503: 1
504: 1
505: 1
506: 1
507: 1
508: 1
509: 1
510: 1
511: 1
512: 1
513: 1
514: 1
515: 1
516: 1
517: 1
518: 1
519: 1
520: 1
521: 1
522: 1
523: 1
524: 1
525: 1
526: 1
527: 1
528: 1
529: 1
530: 1
531: 1
532: 1
533: 1
534: 1
535: 1
536: 1
537: 1
538: 1
539: 1
540: 1
541: 1
542: 1
543: 1
544: 1
545: 1
546: 1
547: 1
548: 1
549: 1
550: 1
551: 1
552: 1
553: 1
554: 1
555: 1
556: 1
557: 1
558: 1
559: 1
560: 1
561: 1
562: 1
563: 1
564: 1
565: 1
566: 1
567: 1
568: 1
569: 1
570: 1
571: 1
572: 1
573: 1
574: 1
575: 1
576: 1
577: 1
578: 1
579: 1
580: 1
581: 1
582: 1
583: 1
584: 1
585: 1
586: 1
587: 1
588: 1
589: 1
590: 1
591: 1
592: 1
593: 1
594: 1
595: 1
596: 1
597: 1
598: 1
599: 1
600: 1
601: 1
602: 1
603: 1
604: 1
605: 1
606: 1
607: 1
608: 1
609: 1
610: 1
611: 1
612: 1
613: 1
614: 1
615: 1
616: 1
617: 1
618: 1
619: 1
620: 1
621: 1
622: 1
623: 1
624: 1
625: 1
626: 1
627: 1
628: 1
629: 1
630: 1
631: 1
632: 1
633: 1
634: 1
635: 1
636: 1
637: 1
638: 1
639: 1
640: 1
641: 1
642: 1
643: 1
644: 1
645: 1
646: 1
647: 1
648: 1
649: 1
650: 1
651: 1
652: 1
653: 1
654: 1
655: 1
656: 1
657: 1
658: 1
659: 1
660: 1
661: 1
662: 1
663: 1
664: 1
665: 1
666: 1
667: 1
668: 1
669: 1
670: 1
671: 1
672: 1
673: 1
674: 1
675: 1
676: 1
677: 1
678: 1
679: 1
680: 1
681: 1
682: 1
683: 1
684: 1
685: 1
686: 1
687: 1
688: 1
689: 1
690: 1
691: 1
692: 1
693: 1
694: 1
695: 1
696: 1
697: 1
698: 1
699: 1
700: 1
701: 1
702: 1
703: 1
704: 1
705: 1
706: 1
707: 1
708: 1
709: 1
710: 1
711: 1
712: 1
713: 1
714: 1
715: 1
716: 1
717: 1
718: 1
719: 1
720: 1
721: 1
722: 1
723: 1
724: 1
725: 1
726: 1
727: 1
728: 1
729: 1
730: 1
731: 1
732: 1
733: 1
734: 1
735: 1
736: 1
737: 1
738: 1
739: 1
740: 1
741: 1
742: 1
743: 1
744: 1
745: 1
746: 1
747: 1
748: 1
749: 1
750: 1
751: 1
752: 1
753: 1
754: 1
755: 1
756: 1
757: 1
758: 1
759: 1
760: 1
761: 1
762: 1
763: 1
764: 1
765: 1
766: 1
767: 1
768: 1
769: 1
770: 1
771: 1
772: 1
773: 1
774: 1
775: 1
776: 1
777: 1
778: 1
779: 1
780: 1
781: 1
782: 1
783: 1
784: 1
785: 1
786: 1
787: 1
788: 1
789: 1
790: 1
791: 1
792: 1
793: 1
794: 1
795: 1
796: 1
797: 1
798: 1
799: 1
800: 1
801: 1
802: 1
803: 1
804: 1
805: 1
806: 1
807: 1
808: 1
809: 1
810: 1
811: 1
812: 1
813: 1
814: 1
815: 1
816: 1
817: 1
818: 1
819: 1
820: 1
821: 1
822: 1
823: 1
824: 1
825: 1
826: 1
827: 1
828: 1
829: 1
830: 1
831: 1
832: 1
833: 1
834: 1
835: 1
836: 1
837: 1
838: 1
839: 1
840: 1
841: 1
842: 1
843: 1
844: 1
845: 1
846: 1
847: 1
848: 1
849: 1
850: 1
851: 1
852: 1
853: 1
854: 1
855: 1
856: 1
857: 1
858: 1
859: 1
860: 1
861: 1
862: 1
863: 1
864: 1
865: 1
866: 1
867: 1
868: 1
869: 1
870: 1
871: 1
872: 1
873: 1
874: 1
875: 1
876: 1
877: 1
878: 1
879: 1
880: 1
881: 1
882: 1
883: 1
884: 1
885: 1
886: 1
887: 1
888: 1
889: 1
890: 1
891: 1
892: 1
893: 1
894: 1
895: 1
896: 1
897: 1
898: 1
899: 1
900: 1
901: 1
902: 1
903: 1
904: 1
905: 1
906: 1
907: 1
908: 1
909: 1
910: 1
911: 1
912: 1
913: 1
914: 1
915: 1
916: 1
917: 1
918: 1
919: 1
920: 1
921: 1
922: 1
923: 1
924: 1
925: 1
926: 1
927: 1
928: 1
929: 1
930: 1
931: 1
932: 1
933: 1
934: 1
935: 1
936: 1
937: 1
938: 1
939: 1
940: 1
941: 1
942: 1
943: 1
944: 1
945: 1
946: 1
947: 1
948: 1
949: 1
950: 1
951: 1
952: 1
953: 1
954: 1
955: 1
956: 1
957: 1
958: 1
959: 1
960: 1
961: 1
962: 1
963: 1
964: 1
965: 1
966: 1
967: 1
968: 1
969: 1
970: 1
971: 1
972: 1
973: 1
974: 1
975: 1
976: 1
977: 1
978: 1
979: 1
980: 1
981: 1
982: 1
983: 1
984: 1
985: 1
986: 1
987: 1
988: 1
989: 1
990: 1
991: 1
992: 1
993: 1
994: 1
995: 1
996: 1
997: 1
998: 1
999: 1
1000: 1

```


100


```

1: SUBROUTINE IMPROVE
2:     Subroutine Improve
3:
4:     Purpose: To improve the assignment of data units among the
5:             processing elements, by a trial and error method
6:             that checks to see if switching one data unit from
7:             one PE to another will help. This is done for all
8:             functions in the signal processing task.
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
82
```



```

PROMPT: PROC (USERPROMPT, LENGTH);
DCL USERPROMPT CHAR(LENGTH);
DCL LENGTH SRIN WORD;
ZINCLUDE CP_6;
ZINCLUDE RK_VFC_C;
ZFPT PROMPT (VFC=YES);
DCL PROMPTWITHVFC CHAR(32);
CALL INSERT (PROMPTWITHVFC, ' ', VFC_TRANSITIONO, USERPROMPT);
FPT_PROMPT_PROMPT_ = VECTOR (SUBSTR(PROMPTWITHVFC, 'MINIMUM(LENGTH,30)+1));
CALL M$PROMPT (FPT_PROMPT);
END;

```

END

1	0	0	9	8	4
---	---	---	---	---	---

FIN