Old Dominion University

ODU Digital Commons

Electrical & Computer Engineering Theses & Dissertations

Electrical & Computer Engineering

Fall 1983

Design of Efficient Algorithms Through Minimization of Data Transfers

Yong Mo Chong

Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/ece_etds

Part of the Computational Engineering Commons, and the Signal Processing Commons

Recommended Citation

Chong, Yong M.. "Design of Efficient Algorithms Through Minimization of Data Transfers" (1983). Master of Science (MS), Thesis, Electrical & Computer Engineering, Old Dominion University, DOI: 10.25777/zkjm-z906

https://digitalcommons.odu.edu/ece_etds/164

This Thesis is brought to you for free and open access by the Electrical & Computer Engineering at ODU Digital Commons. It has been accepted for inclusion in Electrical & Computer Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

DESIGN OF EFFICIENT ALGORITHMS THROUGH MINIMIZATION OF DATA TRANSFERS

by

Yong Mo Chong
B.S.E.E. May 1981 Old Dominion University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
November 1983

Approved by:

____

Meghanad D. Wagh (Director)

____

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

©Copyright by Yong M. Chong 1983

All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

DESIGN OF EFFICIENT ALGORITHMS THROUGH MINIMIZATION OF DATA TRANSFERS

Yong M. Chong
Old Dominion University
Director: Meghanad D. Wagh

This thesis explores the time optimal implementation of computational graphs on a finite register machine. The implementation fully exploits the machine architecture, especially, the number of registers. The derived algorithms allow one to obtain time efficient implementations of a given graph in machines with a known number of registers.

These optimization procedures are applied to digital signal processing graphs. It is shown that the regular structure of these graphs allows one to identify computational kernels which, when used repeatedly, can cover the entire graph. The 1- and r-register implementations of Hadamard and Fast Fourier Transforms using various computational kernels are studied for their code sizes and time complexities. The results obtained also allow one to select an optimal hardware devoted to a particular computational application.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGMENT

I would like to thank Dr. Meghanad D. Wagh for his patience, guidance, and help during the research. This work would not have been possible without his enthusiasm, insight and encouragement. In addition, his assistance during the preparation of this thesis was appreciated since it was the result of many long nights together.

I would also like to acknowledge the other members of my thesis committee, Dr. Sherad Kanetkar and Dr. John W. Stoughton, for their time and consideration. Thanks are also due to Teri M. Owens for her assistance in preparing this thesis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

PAGE

CHAPTER 4 (CONTINUED)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF SYMBOLS

SYMBOL	MEANING
Ci	i-th Computationally Organized Block (COB)
G	Computational graph
r	Number of registers
P	Set of computational points
U	Union
Vi	i-th computable path
ϵ	Belong to
$\not\epsilon$	Not belong to
Ø	Null set
$\subset$	Subset

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

INTRODUCTION

1.1 Background

The past two decades have seen rapid strides in the area of digital signal processing. Many new signal processing techniques were designed and many new applications were discovered. However, most of the effort in this area was concentrated on reducing the complexity of the algorithms involved. Since signal processing algorithms are used repeatedly (and in some cases, continuously) for different data sets, a small reduction in their complexity results in a large saving of practical resources. In addition, the demanding real time applications of signal processing techniques are becoming increasingly popular.

A reduction in time complexity may be achieved by employing hardware techniques such as parallel processing and pipelining, by using faster technologies, or by restructuring computational algorithms so that the time intensive operations are reduced. The least expensive of these, the third alternative, is the subject of this thesis.

Traditionally, only the multiplication was viewed as the time consuming operation. However, several breakthroughs in technology have now reduced the multiplication time significantly. As a result, both the number of multiplications and additions in an algorithm are generally used to estimate its computational complexity. The unsuitability of even this complexity measure may be illustrated by pointing out a case of great practical significance. A Fourier

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

transform algorithm designed by Winograd (WFTA) in 1976 [1] had a smaller number of multiplications and additions and was therefore immediately accepted as a replacement for the fast Fourier transform (FFT) [2]. However, an implementation of WFTA on PDP 11/55 and IBM 370/168 was found to be much slower than that of FFT [3]. This discrepancy could be explained only after a detailed operation count was maintained. It was found that on a PDP11/55 (using Assembler), for example, a 1008 point WFTA required 14.6 msec less time for multiplications than FFT, but simultaneously, used up 40.1 msec more for the memory reference operations resulting in an implementation that was 45% slower than the FFT. The fact that memory referencing is very time intensive may also be understood by examining Table 1.1 which compares the times for various operations in many general purpose microprocessors available today. Even though the importance of reducing the number of memory reference operations is thus obvious, little has been done about it to date. There are two main reasons for this. Firstly, the realization of the importance of these operations is rather recent, and secondly, there does not exist a mathematical model which may, in rather systematic manner, pave the way to such optimization.

Table 1.1. Execution times (in μsec) for various microprocessors [4-8].

microproc.	8080	6800	Z-80	8085A	8086	68000	Z8000	TMS9900
clk. cycle	2.0	1.0	0.5	.32	0.2	0.125	0.25	.3333
Load	7	4	4	4.16	2.8	2.0	3.00	7.30
Store	7	4	4	4.16	3.0	2.125	3.50	7.30
Mop(+,-)	7	4	5	n/a	3.0	1.125	3.75	7.32
Copy	5	2	1	1.28	0.4	0.5	0.75	4.60
Rop(+,-)	4	2	1	1.28	0.6	0.5	1.00	4.60

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Compiler designers had realized the importance of reducing memory fetches as early as in 1964. In that year, Anderson designed an algorithm for compiling a computation expressed as a tree using a stack of local registers [9]. His results were later extended by Nakada who obtained compiling algorithm for arithmetic expressions in computers with n accumulators [10]. His algorithm generated an object code which minimized the frequency of storing and was used in a FORTRAN IV compiler for the HITAC-5020 computer which has 14 accumulators. In a computer with limited core memory, a large amount of data has to be stored on a slow, external memory device. Thus while solving problems on such machines, one needs to minimize the reads and writes to that slow memory. Specific algorithm implementations which distinguish between slow and fast memory and reduce references to the slow memory have also been reported. Both Brenner [11] and Naidu [12] have studied computation of FFT of a large sequence resident in an external device such as disk. Similarly, Eklundh [13] and Naidu [14] have implemented fast transposition of matrices too large to be stored in fast memory. More recently, Nawab and McClellan have done a detailed analysis of implementation of WFTA and FFT on finite register machines and have found optimum number of registers for different length WFTA [15].

1.2 Computer Architecture

One possible definition of computer architecture is the characteristics of a machine as seen by a programmer. In general, it is difficult to categorize different computer architectures because of the numerous variations. One possible scheme proposed by Flynn [16] is to divide computer architectures into four distinct categories: SISD

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(Single-Instruction-Stream/Single-Data-Stream), SIMD (Single-Instruction-Stream/Multiple-Data-Stream), MISD (Multiple-Instruction-Stream/Single-Data-Stream), and MIMD (Multiple-Instruction-Stream/Multiple-Data-Stream). With the exception of SISD, all categories use some type of parallel processing with multiple processors. The SISD architecture has only one processor which uses one instruction per instruction cycle. Almost all general purpose computers and microprocessor systems fall in SISD category. For this reason, the remainder of this thesis addresses only the SISD architecture. A typical SISD architecture has a local register file and a large main memory as shown in Fig. 1.1.

The instructions in SISD architecture may be divided in two categories: memory referenced and local register referenced. A memory referenced instruction is one in which an operand resides in memory. A local register instruction, on the other hand, does not access the memory.

For this study, the set of instructions is restricted to the following:

Load : Rn ← Mj	(Load	Register-n	from	Memory-j)
Store : Mj ← Rn	(Store	Register-n	in	Memory-j)
Mop(*): Rn ← Rn * Mj	(+,-,x	Memory-j	to	Register-n)
Copy : Rn ← Rm	(Copy	Register-m	to	Register-n)
Rop(*): Rn ← Rn * Rm	(+,-,x	Register-m	to	Register-n)

The execution times for these instructions are dependent upon the types of operations and the specific architecture of the machine. Further, for memory related instructions (Load, Store, and Mop(*)), it also depends upon the addressing mode. However, in most cases, (see Table 1.1) the execution of memory reference instructions (Load, Store, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

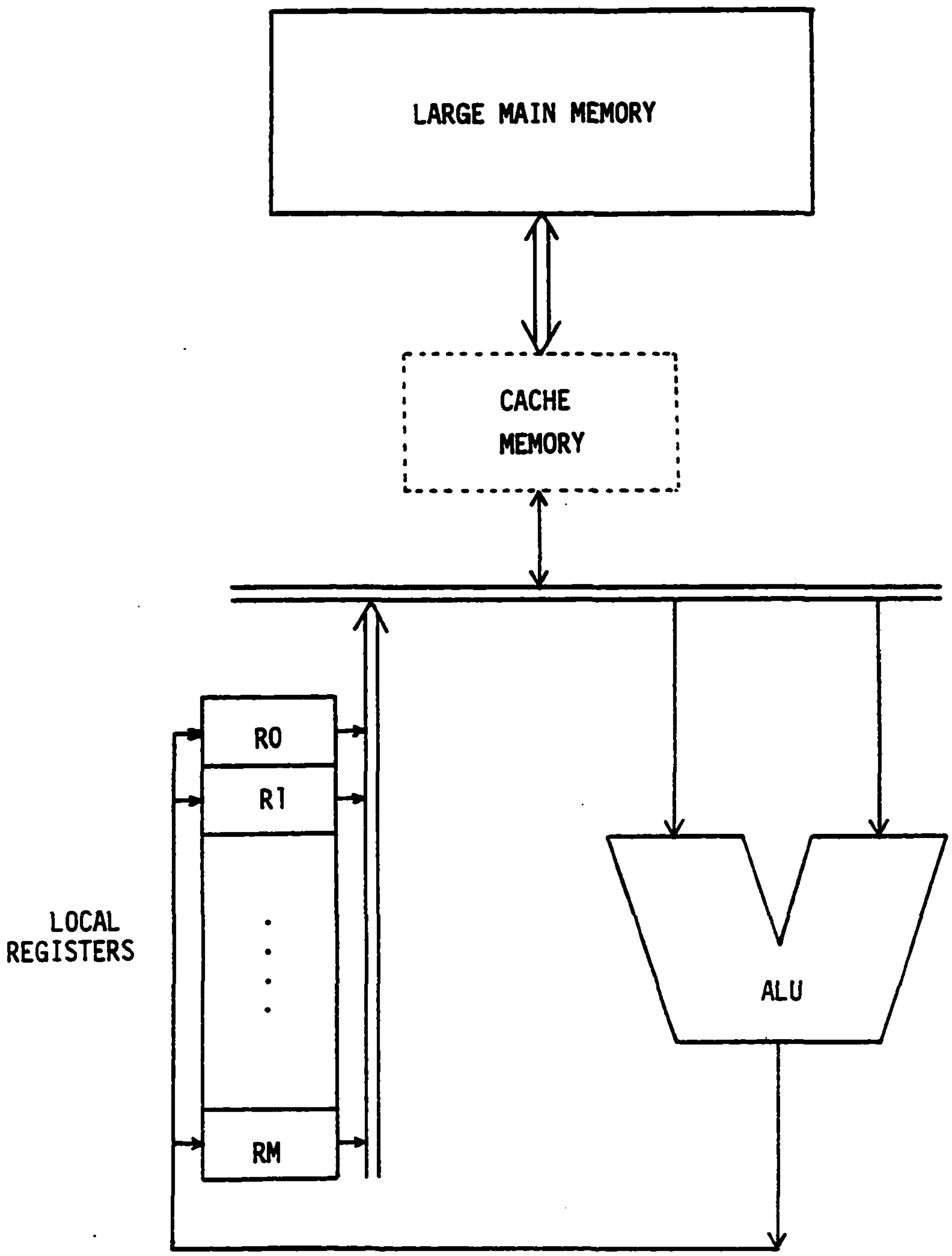


Fig. 1.1 SISD architecture.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Mop(*)) is slower than the equivalent local register instructions (Copy and Rop(*)). This time difference between the two types of instructions is inherent to SISD architecture and may be attributed to the comparatively large access time of memory.

The following normalized times (suggested by actual times listed in Table 1.1) are used in this work to denote the relative time complexity of these instructions.

Tload = 2 units
Tstore = 2 units
Tmop(+,-)= 2 units
Tmop(x) = 4 units
Tcopy = 1 unit
Trop(+,-)= 1 unit

It should be noted that the time differences (Tload-Tcopy), (Tstore-Tcopy), and (Tmop(+,-,x)-Trop(+,-,x)), are chosen to be exactly equal, because they all are identical to the memory access delay of the architecture.

1.3 Problem Identification

Recalling the disscussion in earlier sections, two problems faced by digital signal processing engineers can be easily identified. Firstly, given a machine, how best to exploit its architectural features in order to obtain an efficient implementation of any signal processing algorithm. Since signal processing algorithms are used over and over again, any small improvement in their complexity without calling for an improved hardware is immensely useful.

Secondly, given an algorithm, if one is to construct a special purpose hardware for its implementation, what should be the architectural features that be built in the hardware. Since the cost of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the hardware increases with every new feature added, one must have a clear understanding of the advantages this new feature will provide.

The results obtained in this thesis are first steps towards the solution to these problems. For example, by exploiting the two accumulator feature in a machine (say a 6800 microprocessor) as shown herein, one may improve the computational time of the Fast Fourier Transform by 35.29%. Similarly, the results obtained here demonstrate that a hardware for implementing the Hadamard Transform need not have more than three accumulators, since the gain due to more registers is marginal.

1.4 Unique Approach to the Problem

A directed graph is used here to model a computational algorithm. The nodes of the graph represent actual computations and the edges represent the order between various computations. Since the aim here is to minimize the memory reference operations, the graph is partitioned into subgraphs (called COBs) each of which may be evaluated without any memory reference on a given hardware configuration. This enables one to identify the memory reference operations with the graph edges not included in any COB. In order to minimize such edges, a two step approach is used. First, the given graph is partitioned into COBs suitable for a one accumulator architecture. Next, an accumulator is added to the machine and the COB cover is modified to take into account the availability of the extra register. This second step is repeated until all available registers are used. In addition, the regularity in a signal processing graph is exploited to identify the computational

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

κernels and to implement the graph by repeating the implementation of the κernel.

1.5 Overview

Chapter 2 of this thesis reviews some graph theoretic preliminaries required later. It also defines the Computationally Organized Block (COB) of arbitrary dimension and presents an algorithm to partition the given graph into 1-dimensional COBs. A procedure to cover the graph using r-dimensional COBs ($r \geq 2$) is presented and illustrated in Chapter 3. Using the algorithms, Chapter 4 explores the implementation of efficient algorithms for Hadamard Transform(HT) and Fast Fourier Transform(FFT). This chapter also defines and uses the concept of a primitive COB. Finally, Chapter 5 concludes this thesis by summarizing the results obtained and pointing out directions for future research.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

COMPUTATIONALLY ORGANIZED BLOCK: 1-DIMENSION

As has been stated in Chapter 1, the major thrust of this thesis is the establishment of a mathematical model appropriate for description and implementation of a signal processing algorithm on a finite register machine. Computational graphs for signal processing algorithms are unlike the computational graphs studied in earlier literature in that they do not have the tree structures and instead have feed-forward paths. This chapter is devoted to the investigation and modelling of such graphs.

Section 2.1 describes the nomenclature and the basic properties of signal processing graphs. Based on these properties, Section 2.2 then derives the mathematical models for such computations in a finite register machine. The basic approach here consists of partitioning the graph into modules, each of which may be computed independently in a machine with 'r' registers without making a reference to the memory external to the CPU. These modules are designated herein as COMPUTATIONALLY ORGANIZED BLOCKS (COBs) of dimension r. Since the computations within a COB do not require any memory fetches or stores, the complexity of the algorithm in terms of the number of memory references, then, is determined solely by the number of graph edges joining different COBs. This is shown in Section 2.3. An algorithm to obtain an implementation in terms of 1-dimensional COBs is presented in Section 2.4. and illustrated through an example in Section 2.5.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1 Graph Theory Preliminaries

A computational algorithm can always be represented as a directed graph. Points in such a graph stand for computational nodes and a directed edge from P1 to P2 indicates the involvement of the result at P1 in the computation P2. Alternately, a directed graph G=(P,E) can be represented by a set of points P and a set of ordered pairs, $E=\{(x,y) | x,y \in P\}$ as Figure 2.1 illustrates. Note that in this figure, points A,B,C,D and the dotted lines shown in the graphical representation are not really part of the computational graph and will not be shown in graphs encountered later. We now give some basic definitions and results from graph theory, which would be used later.

Partial Order:

A set $E \subset P \times P$ of ordered pairs is said to be a partial order if it is weakly antisymmetric (i.e., if $(x,y) \in E$, then $(y,x) \notin E$ for $x \neq y$) reflexive (i.e., $(x,x) \in E$ for all $x \in P$) and transitive (i.e., if $(x,y),(y,z) \in E$ then $(x,z) \in E$ for all $x,y,z \in P$). In representing computational graphs we will relax the reflexivity requirement which implies a loop at every computational node. Every computational graph is then a partial order.

Total Order:

In addition to the partial order, if the set $E \subset P \times P$ is such that for any $x,y \in P$ either $(x,y) \in E$ or $(y,x) \in E$ or $x=y$, then E is called a total order. We will show that 1-dimensional COBs are subgraphs with total order.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

E = A + C

F = A - C

G = B + D

H = B - D

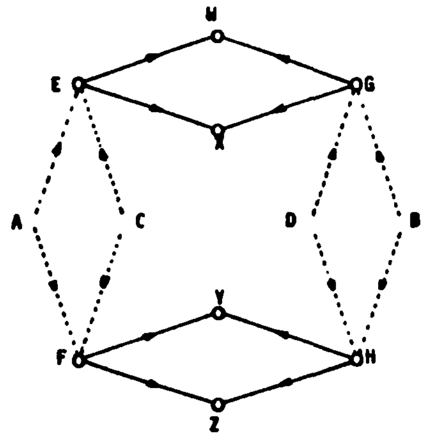
W = E + G

X = E - G

Y = F + H

Z = F - H

COMPUTATION



GRAPHICAL REPRESENTATION

$P = \{E, F, G, H, W, X, Y, Z\}$

$E = \{(E,W), (E,X), (F,Y), (F,Z), (G,W), (G,X), (H,Y), (H,Z)\}$

ALTERNATE REPRESENTATION

Fig. 2.1. Graphical and alternate representation of a computation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Indegree, Outdegree:

Let $Iy=\{x \mid (x,y) \in E\}$ and $Oy=\{x \mid (y,x) \in E\}$. Then $|Iy|$ and $|Oy|$ are called the indegree and the outdegree of point y respectively. In a computational graph, indegree of a point can only be 0, 1 or 2 since we deal only with the binary operations.

Minimal, Maximal points:

Points in a graph with indegree zero are called minimal points. Similarly points with outdegree zero are called maximal points.

Path:

An ordered n-tuple (X1,X2,...Xn) with (Xi,Xi+1) $\in$ E for i=1,2..N-1, is called a path of length n-1 in the graph G=(P,E).

Acyclic Graph:

A graph with no path with idendical first and last points and length ≥ 2 is called an acyclic graph. A computational graph is always acyclic for the following simple reason. (Xi,Xi+1) $\in$ E implies the computation of point Xi+1 requires the result from point Xi. Now if a sequence (X1,X2,X3....Xn-1, Xn=X1) with (Xi,Xi+1) $\in$ E for i=1,2,3,...,n-1 exists, then it implies that the computation of Xn requires Xn-1, which in turn requires Xn-2... . Proceeding in this manner, we conclude that computation of Xn, which is really X1, requires X2. But since (X1,X2) $\in$ E computation of X2 requires X1 and thus this computation cannot be carried out.

Basic Representation of a Graph:

A subgraph obtained by eliminating from the original edge set every pair (X,Y) for which there is a path between X and Y of length ≥ 2 is known as the basic representation of the graph. Figure 2.2 shows a graph and its basic representation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

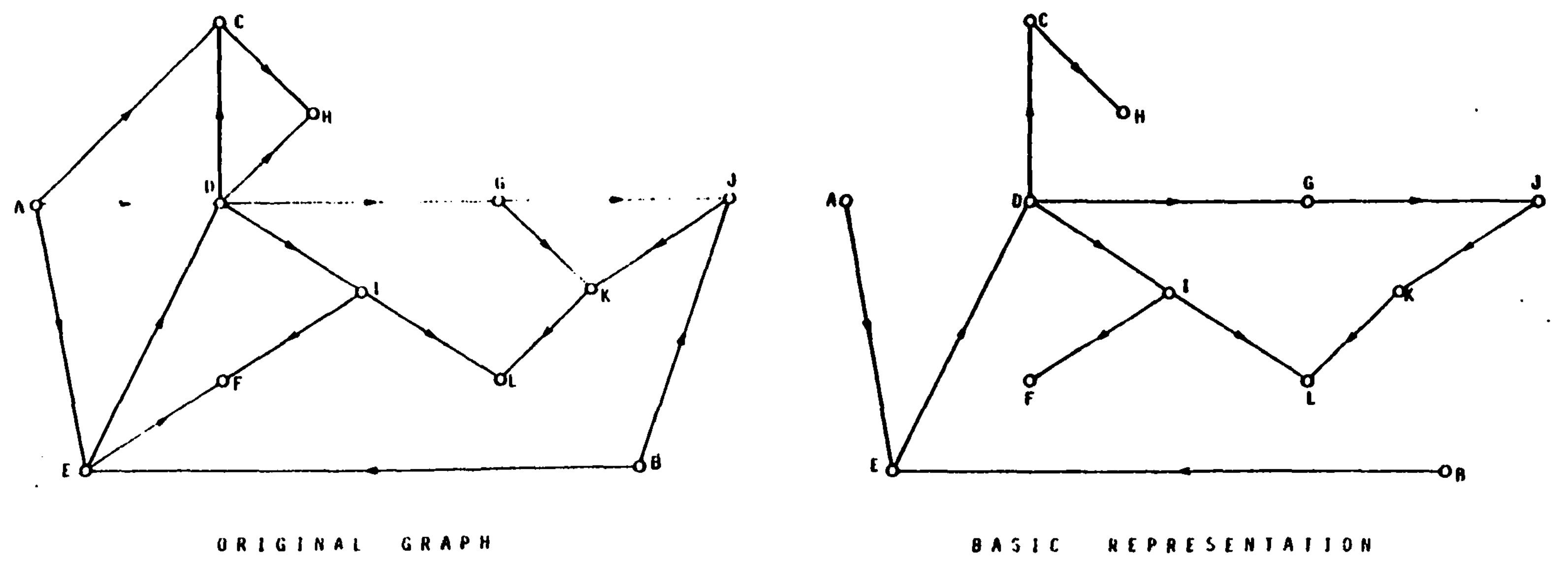


Fig. 2.2. Basic representation of a graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

<u>Topological Sort</u>:

Topological sort of a graph $G=(P,E)$ is a graph $G'=(P,E')$ such that G' has only one minimal point of outdegree one, only one maximal point of indegree one, indegree and outdegree of all points except these are one, and a path from X to Y ($X,Y \in P$) in G implies a path from X to Y in G'. Figure 2.3 illustrates topological sorts.

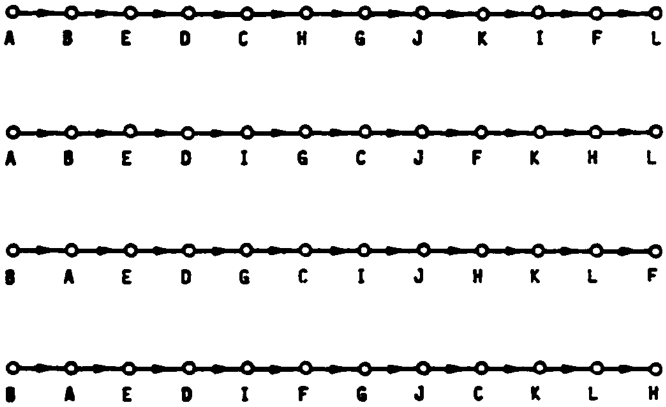


Fig. 2.3. Four topological sorts of the graph in Fig 2.2.

The following results from graph theory are required in this thesis [18].

Theorem 2.1
The restriction of any partial order is itself a partial order.

Theorem 2.2
In a finite nonempty partially ordered set, there is at least one maximal and one minimal element.

Theorem 2.3
If graph G is acyclic, then there exists a unique basic representation.

Theorem 2.4
Topological sort of a finite graph $G=(P,E)$ exists if and only if G is acyclic. Further, this topological sort is unique if and only if E is a total order relation, in which case this sort is the basic representation of G.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2 Computationally Organized Block(COB)

In this section, the concept of Computationally Organized Block (COB) is defined. Then, the computational complexity of an algorithm is related to the partitioning of its graph into various COBs.

Definition of an r-dimensional COB:

Let G=(P,E) be an acyclic computational graph. Let Gy=(Y,Ey) denote the subgraph obtained by restricting the set of points to $Y \subset P$. Then, COB Gy′ of dimension r is a subgraph Gy′= (Y,Ey′), $Ey' \subset Ey$ with the following property:

> The computation represented by Gy′ can be performed in a SISD architecture machine with 'r' registers without any store operations.

For later use, for every COB, we define an integer function n(.) with domain Y such that

(i) $n(A) < n(B)$ if there exists a path from point A to point B in graph G.

(ii) $n(A) \neq n(B)$ if $A \neq B$.

Since 1-dimensional COBs are paths in the original graph and a path in an acyclic graph is a total order, the points in every 1-dimensional COB form a total order.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example of a COB of dimension 1:

In graph G of Fig. 2.4, the subgraph Gy′ =(Y,Ey′) is a COB of dimension 1, where Y ={A,B,C,D} and Ey′={(A,B),(B,C),(C,D)}. It may be implemented as R1 ← F, R1 ← R1+G, R1 ← R1+E, R1 ← R1+J, R1 ← R1+M1.

Example of a COB of dimension 2:

In graph G of Fig. 2.5, the subgraph Gy′ =(Y,Ey′) is a COB of dimension 2, where Y={A,B,C,...,H} , and Ey′ = {(A,B), (B,C), (B,D), (D,E), (E,F), (E,H), (E,G)}. It may be implemented as R1 ← L, R1← R1+M, M1 ← R1, R1 ← R1+K, R2 ← R1, R1 ← R1+M1, R2 ← R2+N, R2 ← R2+O, R1 ← R2, R2 ← R2+J, R2 ← R1, R2 ← R2+P, R1 ← R1+I.

2.3 Complexity of 1-Register Implementation

As can be noted from Fig. 2.4, a one register COB is a total order and except for the minimal(first) point which needs to be evaluated through a Load and a Mop(+), all other points in the COB are computed only through a Mop(+) each. Similarly only the maximal(last) point and points with outdegree ≥ 2 need to be stored in the memory. If a computational graph is covered by 1-register COBs, the complexity of the complete graph may be obtained by summing the complexity associated with the points in each COB. This immediately gives following complexity of 1-register implementation of the total graph.

Number of Loads = Number of COBs

Number of Mop(+)= Total number of points in the graph

Number of Stores= Number of points in the graph with outdegree ≥ 2
+ Number of COBs with last point outdegree ≤ 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

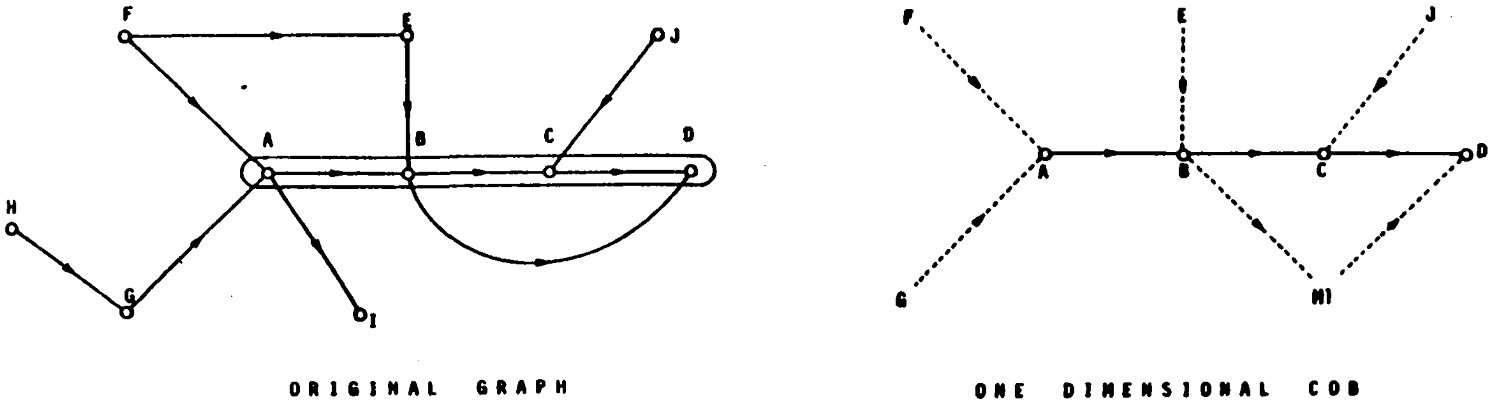


Fig. 2.4. Example of a 1-dimensional COB.

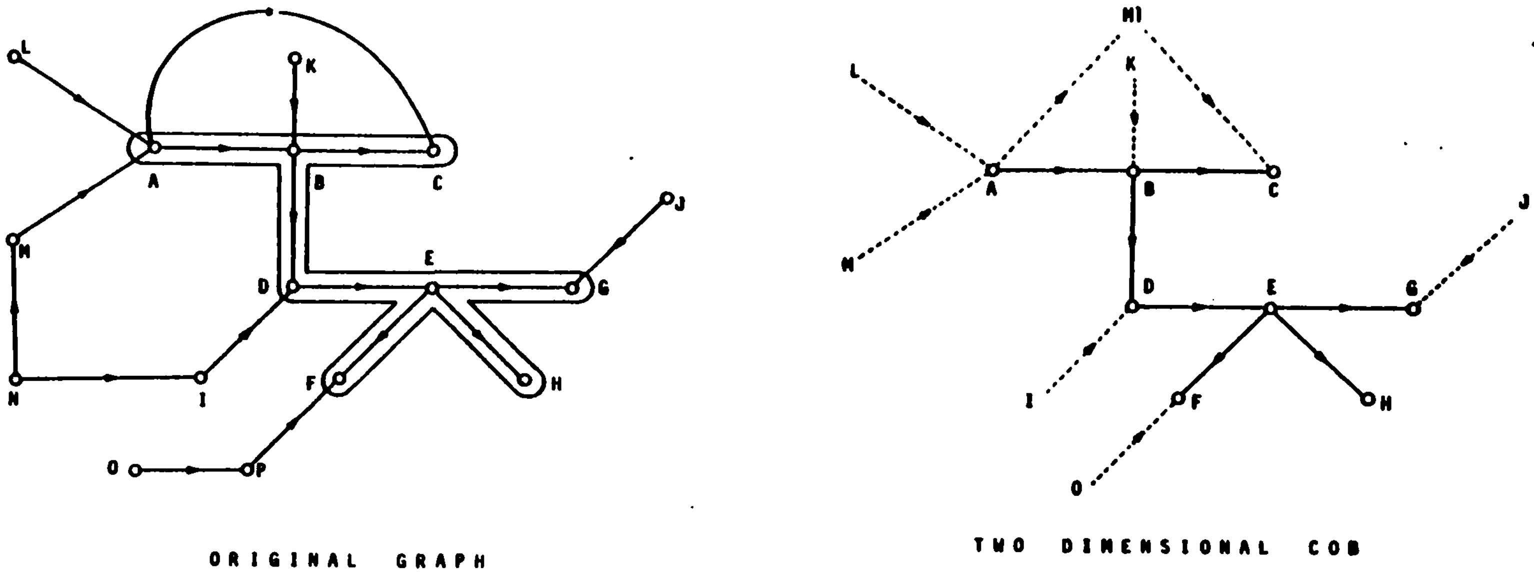


Fig. 2.5. Example of a 2-dimensional COB.

From the assumptions in Chapter 1, each of these operations take exactly two units of time, and hence the total time complexity of computation

$$T = (\#\text{ of Loads}) + (\#\text{ of Mop}(+)) + (\#\text{ of Stores})$$
$$= [(\text{total number of points in the graph}) + (\text{number of points with outdegree} \geq 2 \text{ in the graph})] + [(\text{number of COBs}) + (\text{number of COBs with last point outdegree} < 2)].$$

It should be noted here that both the terms in the first square bracket are totally dependent on the given computational graph. On the other hand, the terms in the second square bracket, namely, the number of COBs and number of COBs with last point's outdegree < 2 are dependent upon the manner in which the COBs are chosen.

2.4 Algorithm for Implementation of a One Register Machine

It was shown in Section 2.3 that the time complexity of an implementation on a 1-register machine is largely dependent upon the number of one dimensional COBs covering the graph. In this section, we present a heuristic algorithm which partitions the original graph into one register COBs in a manner which minimizes the total number of COBs. This partitioning would be referred to as a 1-dimensional COB cover of the graph. Since all points within a COB are evaluated consecutively, computability of the implementation for the entire algorithm demands that the graph obtained by replacing every COB by a point should still be acyclic. Following algorithm guarantees this property of the COB cover.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 1(Initialization)

Set i=1 and let G =(P´,E´) be the Basic Representation of G.

Step 2(Computable path determination)

Find all computable paths in G´. A path (X1,X2,..,Xt) is a computable path if

a. X1 is a minimal point of G´.

b. (Xj,Xj+1) ε E´, j= 1,2,..,t-1.

c. Xj has indegree one for j=2,3,..,t.

d. Either Xt is a maximal point of G´ or, for every X ε P´ such that (Xt,X) ε E´, there exists Y ε P´ such that (Y,X) ε E´ and Y ∉ Xi for i=1,2,...,i-1.

Step 3(Choosing a COB)

(a). If a computable path has a maximal point, choose the path as COB Ci = (Pi,Ei) and go to step 4. (If there is more than one computable path with maximal point, one may choose any of them.)

(b). Generate graph G" from G´ by deleting all points on all computable paths. Let S denote the set of minimal points of G". Find, if possible, computable paths V1,V2,...,Vn with terminal points X1,X2,...,Xn respectively such that for i=1,2,...,n there exist (not necessarily distinct) Yi ε S satisfying (Xi,Yi) ε E´ and for any X ∉ V1 U V2 U ... U Vn, (X,Yi) ∉ E . Choose the path V1 as COB Ci=(Pi,Ei) and go to step 4.

(c). Find computable paths V1, V2,..., Vn with terminal points X1,X2,...,Xn respectively such that for i=2,3,...,n there exist Yi ∉ S and Y1 ε S satisfying (Xi,Yi), (Zi-1,Yi),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(X1,Y1) ε E´ where Zi is the non-terminal point of path Vi. Choose the path V1 as COB Ci=(Pi,Ei).

Step 4(Deleting a COB from the graph)

Let Pi={X1,X2,...,Xt} and Ei = {(Xi,Xi+1)| i=1,2,...,t-1}. Modify E´ ← E´- {(X,Y)|X ε Pi} and P´ ← P´- Pi. If P´= Ø, the procedure ends. Otherwise, i ← i+1 and go to step 2.

The reason for using the basic representation (as per step 1) in the algorithm is to eliminate all extraneous edges from a given computational graph. The edges removed by basic representation are those that can never be part of a computable path. This can be proved as follows:

Let there exist edge (A,B) and path (A,...,C,B) of length ≥ 2 in graph G. Suppose V=(X1,X2,...,Xn,A,B,...) is a computable path. Since both (A,B) and (C,B) ε E, B uses results of both the computations at A and C. Thus point C should also be on the path V before point B i.e., C=Xi, 1≤i≤n. The total order of the points on the path implies that there exists a path from C to A in G. But since (A,...,C,B) is also a path in G, G has a cycle (A,...,C,...,A) and hence is not acyclic. Thus our assumption that edge (A,B) is on a computable path is wrong.

Conditions a. through c. listed in step 2 of the algorithm ensure that every path is computable. Condition d. allows one to choose the longest possible chain of computable points as a computable path.

We now show that the step 3 of the algorithm always allows one to choose a COB. Note that if there is no path with terminal point as a maximal point of G´, then the graph G" is not empty and is acyclic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

because of Theorem 2.1. Furthermore, the set $S \neq \emptyset$ because of Theorem 2.2. Finally, notice that any $s \in S$ has an indegree 2 in G' and indegree 0 in G''. This follows from the fact that $s \in S$, being a minimal point, has indegree 0 in G''. If s had indegree 0 in G', then a path (s) would have been a computable path and $s \notin G''$. Finally if s had indegree 1 in G', then for some X on a computable path, V, $(X,s) \in E'$ and s would be on another computable path identical to V till X and containing s. Thus even in this case $s \notin G''$.

There are at least two computable paths left after eliminating some computable paths which have no points $X \in V$, $s \in S$ such that $(X,s) \in E'$. (The reason why there are at least two and not just one computable paths left is as follows: if the point $s \in S$ gets both of its inputs from the same computable path, V, in G', i.e., (Xi,s), $(Xj,s) \in E'$, $i > j$, with both Xi, $Xj \in V$, then there is a path of length ≥ 2 between Xj and s, namely, the path (Xj,...,Xi,s). Therefore, presence of the edge (Xj,s) in G' contradicts the fact that G' is a basic representation).

To justify the weighing scheme outlined in step 3, suppose that the last node of every COB is colored red. To minimize the number of COBs, one should thus have as few red points as possible in the final graph. All maximal points of G must be red, since COBs computing these must end there. For this reason, if one finds a path with its last point, a maximum point, then one may safely choose it as a COB since no other choice of a COB may ever save the last point of this path from being red.

All points X of the graph for which there exist some indegree one points Y such that $(X,Y) \in E$, are definitely not red, since any computable path containing X can always be extended to Y; and thus, X is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

never the last point of any COB. Thus the only points which may be affected by choice of COBs are those $X \in P'$ for whom every Y with $(X,Y) \in E'$ has an indegree 2.

At any stage (any i value) in the algorithm, no point Y with indegree 2 in G′ of that stage can belong to any computable path because of condition c. of step 2. Thus a point $Y \in P'$ of indegree 2 with $(X,Y), (Z,Y) \in E'$ can occur only in following configurations:

i) $X, Z \in G''$.

ii) $X \notin G''$ and X is non-terminal point of a computable path. $Z \in G''$.

iii) $X, Z \notin G''$. Neither X nor Z are terminal points of their respective paths V1 and V2.

iv) $X, Z \notin G''$. X is a terminal point of path V1 and Z is a terminal point of path V2. ($V1 \neq V2$, as has been shown earlier).

v) $X, Z \notin G''$. X is a terminal point of path V1, but Z is not a terminal point of path V2.

We now determine the effect of choosing a particular path as COB at a given stage on X and Z. In case i), choosing a particular path as a COB at this stage clearly has no effect on the color of X and Z.

To deal with the remaining cases, note that a computable path at any stage, if not chosen as a COB, still remains as a computational path at the next stage. There are only two exceptions to this. Firstly, some initial portion of the path and the chosen COB may be same. In this case, those initial points already computed by the chosen COB will no longer be on the path. Secondly, let X be the terminal point of the computable path and $(X,Y) \notin E'$ for some indegree 2 point $Y \in G''$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The chosen COB might convert Y to a point of indegree 1. In this case, the computable path will be appended at least by point Y.

From the disscussion above, the point X in case ii) and points X and Z in case iii) cannot be painted red regardless of choice of COB. The point Z in case ii) is also obviously not affected by this choice.

Regarding case iv), note that choice of a computational path other than V1 and V2 as a COB does not in any way affect paths V1 and V2. Choosing V1 or V2 as COB has the same effect of painting exactly one of the points X or Z red. Thus at the present stage or some time in future, one of these two points will be painted red. In this case, one can choose one of the paths as a COB since any other choice will not save both the points from being red. The situation described in part (b) of step 3 of the algorithm is a generalization of this case.

Finally, in case v), choosing a computational path other than V1 or V2 has no effect on the two paths as before. If V1 is chosen as a COB, then X becomes a red point, however, choice of V2 as a COB reduces the indegree of Y to one thus implying that X will now never be red. Note that in both cases, point Z is not red, since it is not a terminal point of any COB. One should, in this case, choose V2 as the COB to save one red point. The situation described in part (c) of step 3 of the algorithm is a generalization of this case.

These arguments also allow one to find the bounds on the number of 1-dimensional COBs required to cover a given graph. Minimum number of red points in a graph is equal to the number of maximal points and maximum number of red points equal the maximal (certainly red) points plus indegree two (potentially red) points in the graph. Using the normalized execution times assumed in Section 1.2, one may also get

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

upper and lower bounds on the time complexity. For example, in the graph of Fig. 2.6, there are only 2 maximal points and 5 points with indegree 2. Thus, for this graph,

$$2 \leq \text{Number of COBs} \leq 7.$$

Using the time complexity expression in Section 2.3, and the fact that maximal points have outdegree 0 one gets the time complexity of this graph as:

$$46 \leq \text{Time Complexity} \leq 66.$$

2.5 Example

The following is an example to find implementation of the graph G in Fig. 2.6 on 1-register machine.

Step 1: Basic representation of G = (P , E) is G′ = (P′, E′) where P′ = P = {A,B,...,N} and E′ = E - {(A,B),(K,M)}. Set i=1.

Step 2: The computable paths are V1 = (A,B,C,D), V2 = (A,B,C,J), and V3=(A,E,F).

Step 3: Since V1 has a maximal point, it is chosen as the first COB based on condition (a). C1=(P1,E1) where P1 = (A,B,C,D) and E1 = {(A,B),(B,C),(C,D)}.

Step 4: Modified P′ = {E,F,...,N} and E′ = {(E,F),(F,G),(G,H),(G,K),(H,I),(I,N),(J,K),(K,L),(L,M)} . $i \leftarrow 2$.

Step 2: The computable paths are V1=(E,F,G,H,I), and V2=(J).

Step 3: In the present case, (J,K), (I,N), (G,K) ε E′ , I and J are terminal points of V1 and V2 respectively, K ε S and N $\not\varepsilon$ S. Hence, based on condition (c) of step 3, the second COB C2 is chosen as V1, C2=(P2,E2) where P2 = {E,F,G,H,I} and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

E2 = {(E,F),(F,G),(G,H),(H,I)}.

Step 4: Modified P′ = {J,K,...,N} and E′ = {(J,K),(K,L),(L,M),(M,N)}.

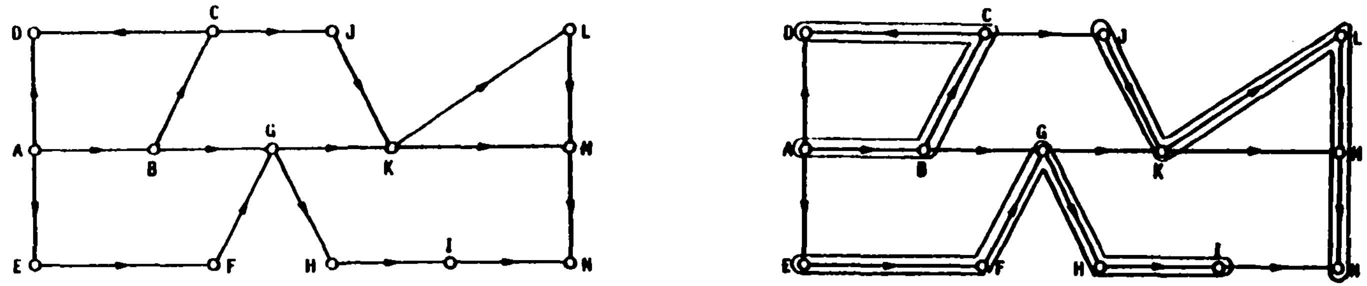
i ← 3.

Step 2: The only computable path is V1=(J,K,L,M,N).

Step 3: Choosing the third COB C3 as V1, C3=(P3,E3) where P3= {J,K,L,M,N} and E3 = {(J,K),(K,L),(L,M),(M,N)}.

The implementation of the computation of Fig. 2.6 in a one register machine will need (from Section 2.3) only 3 Loads, 14 Mop(+,-) and 8 Stores requiring a total of 50 units of time. On the other hand, if each point had been evaluated independently through a Load, Mop(+,-) and Store, then one would have required 84 units of time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



original graph

1-dimensional COB cover

Fig. 2.6. A computational graph and its 1-dimensional COB cover.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

COMPUTATIONALLY ORGANIZED BLOCK: R-DIMENSION

As has been shown in Chapter 2, the number of edges between COBs basically determines the efficiency of implementation of the algorithm. The implementation on an r-register machine thus should be based on cleverly formed r-dimensional COBs with as few interconnections as possible. This would in general be a very difficult task, even for algorithms of moderate complexity. In this thesis we adopt an approach which allows us to design an implementation for an r register machine from that of an r-1 register machine.

In the first section of this chapter, the time complexity of the implementation of a graph using r dimensional COBs is derived. In Section 3.2, an algorithm is presented to merge (r-1)-dimensional COBs to form r-dimensional COB cover for the graph. Using this algorithm repeatedly, any dimensional COB cover may be constructed. In order to illustrate the COB merging process, 4-point Fast Fourier transform algorithm is presented as an example in Section 3.3.

3.1 Complexity of r Register Implementation

In this section, time complexity of an arbitrary computational graph covered by r-dimensional COBs is derived. The derivation is constrained to graphs with points with maximum outdegree 3 points. This limitation does not impose a significant handicap for a realistic computational graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Suppose the given computational graph is partitioned in r-dimensional COBs. The following notation is used in the time complexity derivation.

En : number of edges outside of COBs, which start from the points with outdegree (not including the outdegree due to the edges within COBs) of n.

En´ : number of edges outside of COBs, which end at the points with indegree (not including the indegree due to the edges within COBs) of n.

Pn : number of points with outdegree n in the original graph.

Pn´ : number of points with indegree n in the original graph.

Following operation counts based on an implementation of the graph in terms the r-dimensional COBs are easy to obtain.

# of Store : E1 + E2/2 + E3/3
# of Loads : P0´+ E2´/2
# of Mop(*): P0´+ P1´+ E1´+ E2´/2
# of Copies: P2 + P3 - E1 - E2/2 - E3/3
# of Rop(*): P2´- E1´- E2´/2

If all arithmetic operations are assumed to be (+,-) and the normalized times for various operations given in Section 1.2 are used,

Total Time = [4P0´+ 2P1´+ P2 + P2´+P3]
+ [E1 + E2/2 + E3/3 + E1´ + 1.5 E2´].

The quantities in the first bracket are constants, since they are related to the original graph. However, the quantities in the second bracket are dependent upon the way the graph is partitioned in r-dimensional COBs and are therefore related to the particular choice of a r-dimensional COB cover. Thus, reduction of time complexity of an implementation in a machine with r registers implies proper selection of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a r-dimensional COB cover for the graph which minimizes the number of edges outside the COBs.

3.2 r-Dimensional COB Algorithm

Following algorithm may be used to obtain a r-dimensional COB cover for a graph from a (r-1)-dimensional COB cover.

Step 1(Initialization)

Let C′ be the set of (r-1)-dimensional COBs. Assign an integer function ni to points in each COB Ci ε C′ having the property that ni(x) < ni(y) ; x,y ε Ci iff computation of x is done before the computation of y. Let E" denote the set of edges in the original graph G, not included in any of the COBs in C′. Set m = 1.

Step 2(Finding all computable paths)

A computable path is a sequence of points of C′ along with a subset E′ $\subset$ E". A computable path is generated using the following four transformations:

T1: Let Ci be the last COB of the current path. COB Cj may be appended to the path iff the only inputs to Cj are from COBs on the path, and if Ck preceeds Ci on the path, for some x ε Ck, y ε Ci, (x,y) ε E′, then there should exist (z,w) ε E" such that z ε Ci and w ε Cj and ni(y) $\leq$ ni(z). If Cj is added to the path, set E′ = E′ U (z,w).

T2: COB Ck is inserted between two consecutive COBs Ci and Cj on the path iff the only inputs to Ck are from the COBs on the path upto Ci, and if for some x ε Ci, y ε Cj, (x,y) ε E′, then there exists (x,z) ε E", z ε Ck. If Ck is added to the path, set E′ = E′ U (x,z).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

T3: COB Ck is inserted between two consecutive COBs Ci and Cj on the path iff the only inputs to Ck are from the COBs on the path upto Ci, there is no edge on the path going from Ci to Cj, and for some $x \in Ci$, $y \in Ck$, $(x,y) \in E''$, such that the function ni has its maximum value at x and the function nk has its minimum value at y. If Ck is added to the path, set $E' = E' \cup (x,y)$.

T4: COB Ck is inserted between two consecutive COBs Ci and Cj on the path iff the only inputs to Ck are from the COBs on the path upto Ci, for some $x \in Ci$, $y \in Cj$, $(x,y) \in E'$, function ni has its maximum value at x, and for some $z \in Ci$, $w \in Ck$, $(z,w) \in E''$, such that the function nk has its minimum value at w. If Ck is added to the path, set $E' = E' \cup (x,y)$.

These transformations are illustrated in Figure 3.1.

A computable path is generated as follows:

a. Set $E' = \emptyset$ and choose a COB with no input edges as the first point of the path.

b. Let (C1,C2,...,Ct) be the current path. Insert a COB after Ci in the path by applying rules T1,T2,T3 and T4 above iff no COB can be inserted after C1,C2,...Ci-1.

c. The path is completed when rules T1,T2,T3 and T4 can no more be applied to add COBs to that path.

Step 3(Choosing an r-dimensional COB)

(a) For each computable path, find the number of COBs which can be attached to a path if input edges of attached COBs coming from COBs not on the path are disregarded. If there exists a path

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

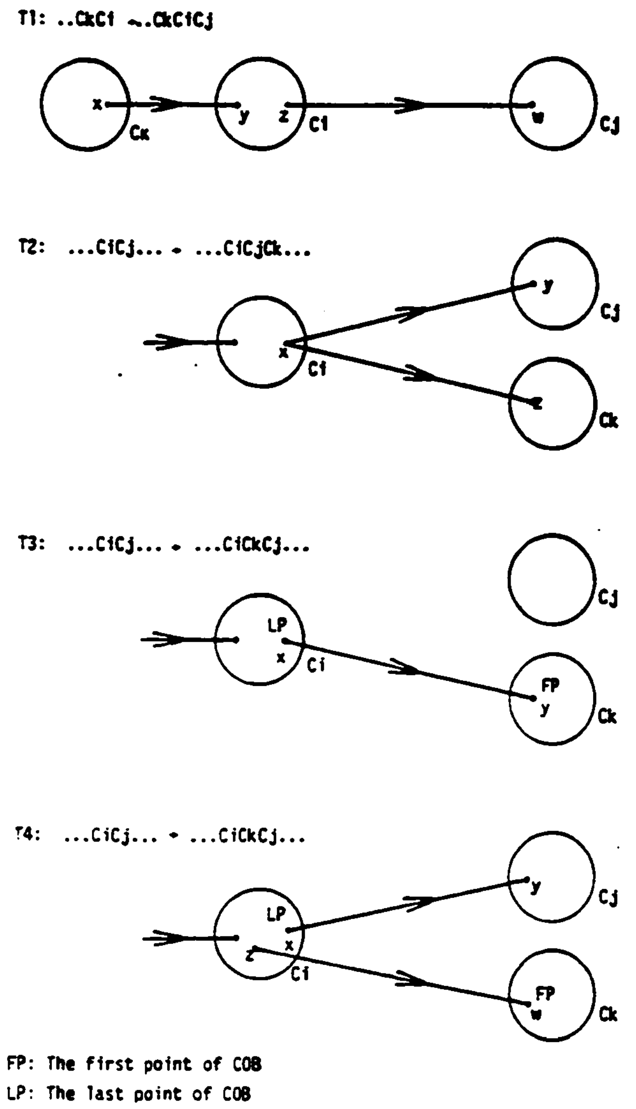


Fig. 3.1. The four basic transformations used to form computable paths.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with 0 attachable COBs, then choose the path as the m-th COB of dimension r and go to step 4.

(b) Find if possible, computable paths V1,V2,..,Vn such that more COBs can be attached to path Vi if input edges of attached COBs coming from COBs on path Vi-1 are disregarded for i = 2,..,n-1 and more COBs may be attached to path V1 if the input edges of attached COBs coming from COBs on the path Vn are disregarded. Choose path V1 as the m-th COB of dimension r.

(c) Find computable paths V1,V2,..,Vn such that more COBs can be attached to path Vi if input edges of attached COBs coming from COBs on path Vi-1 are disregarded for i = 2,..,n-1. Choose path V1 as the m-th COB of dimension r and go to step 4.

Step 4(Deleting a r-dimensional COB)

Delete from set E" edges originating from the COBs on the chosen path. If E"= Ø, then the procedure terminates, otherwise, let m = m + 1 and go to step 2.

The assignment of the integer function n(.) in step 1 ensures the computational ordering within a COB.

The four transformations used to obtain a computable path in step 2 of the algorithm basically gurantee the computability of each path and also ensure that each path absorbs as many edges in E" as possible. It may also be noted that the four transformations are mutually exclusive. T1 is the only transformation which adds a new COB at the end of the current path. Only in T3, new COB is inserted between two unconnected COBs on the current path which are not connected. Transformations T2 and T4 would be identical only in the case when x is the last point of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Ci, y ε Cj, z is the first point of Ck, and (x,y) ε E´, (x,z) ε E". But in this case, since the only inputs to Ck are from the COBs on the current path till Ci, COBs Ci and Ck would not be separate COBs of (r-1)-dimension.

Step 3 of this algorithm may be reasoned out in exactly the same manner as step 3 of the algorithm for 1-dimensional COBs.

3.2 Example

In this section, implementations on various machines of the 4 point Fast Fourier Transform (FFT) graph shown in Fig. 3.2 are sketched. The 1-dimensional COB cover of this graph shown in Fig. 3.3 is obtained by the algorithm of Chapter 1 and used as an input for the algorithm of the earlier section. The following steps describe the formation of 2-dimensional COBs derived through the application of this algorithm.

Step 1: Graph G´ = (C´,E´) is constructed as shown in Fig. 3.4.

Integer function ni is assigned to each point for every COB. E´ is set of edges remaining outside of COBs in Fig. 3.4.

Steps 2 and 3 are shown in the following table for brevity.

Path	Step 2 COB Sequence	Step 2 Computable Path set E´ *	Step 3 Number of Attachable COBs
V1	C1C2	(1,1;2,2)	3
V2	C3	---	1
V3	C5C6C9	(5,1;6,2),(6,2;9,1)	2
V4	C10	---	1

* Notation (a,b;c,d) stands for an edge from the point b of COB a to the point d of COB c.

There is no path with 0 attachable COBs. But path V2 may be extended by COB C4 if inputs to C4 from path V1 ((2,4;4,5)) is disregarded.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

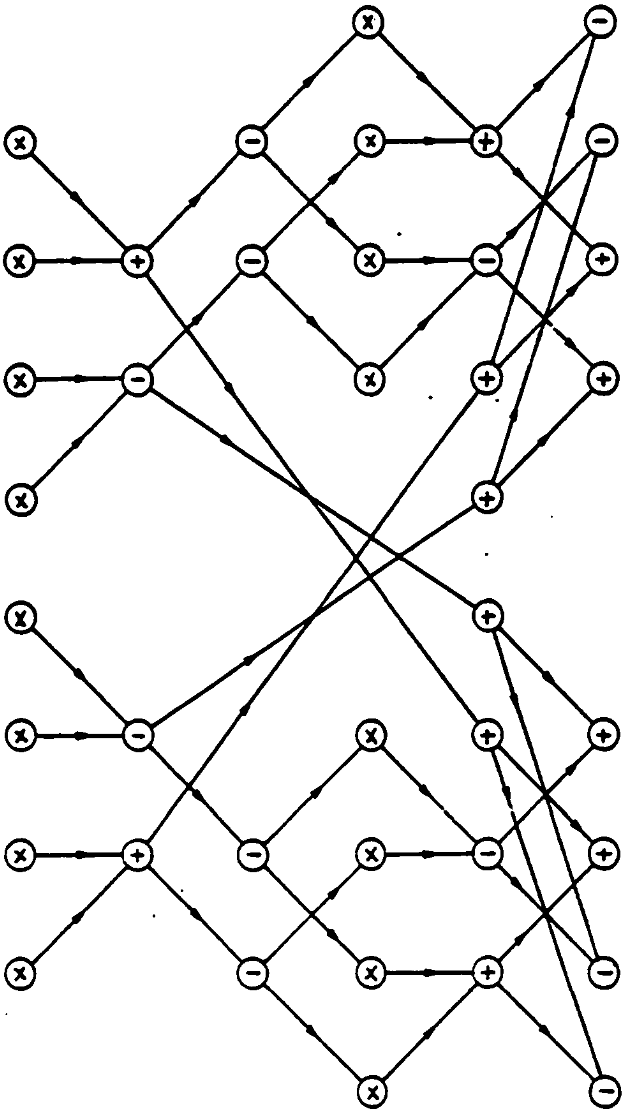


Fig. 3.2. Computational graph of 4-point FFT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

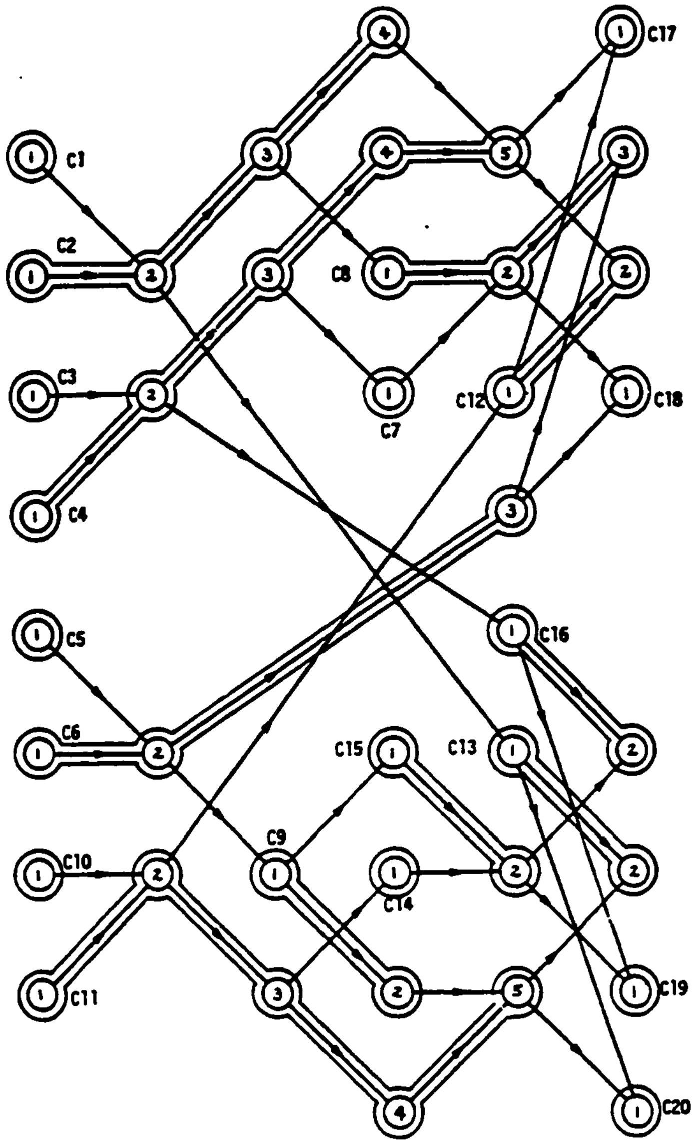


Fig. 3.3. 1-dimensional COB cover of the 4-point FFT graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

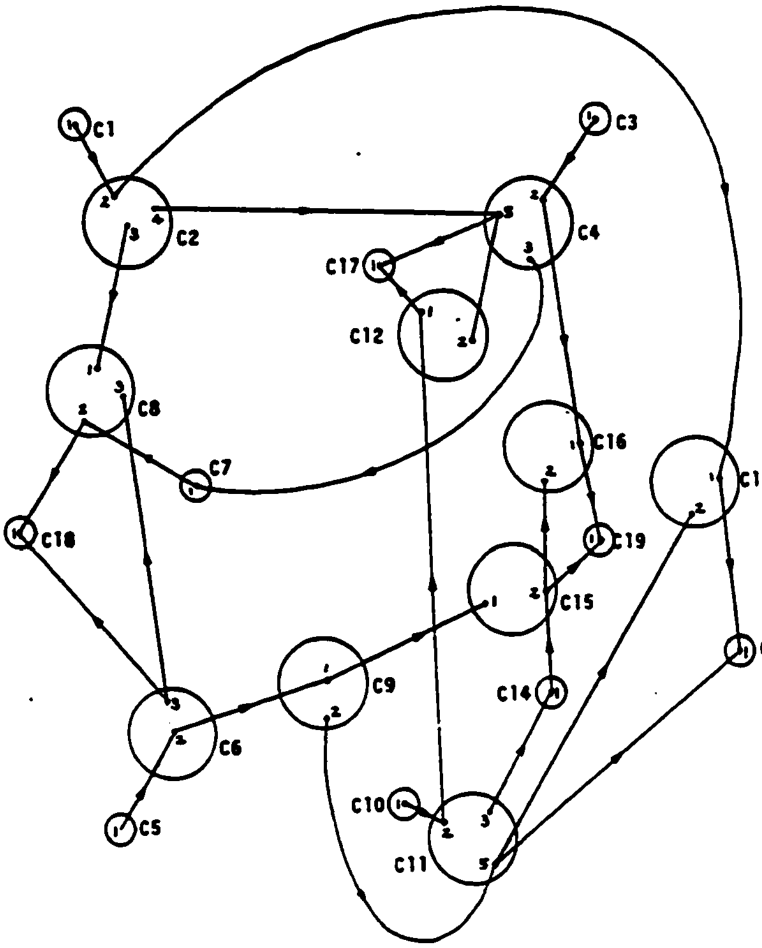


Fig. 3.4. Equivalent 1-register COB cover of the 4-point FFT graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Similarly path V4 may be extended by COB C11 if inputs to C11 from path V3 ((9,2;11,5)) are disregarded. Thus from condition (b) of step 3, one may choose either V1 or V3 as the first COB. Let V3 be the first 2-dimensional COB.

Step 4: All edges originating from C5, C6, and C9 are deleted from E".

Steps 2 and 3:

Path	COB Sequence	Step 2 Computable Path set E´	Step 3 Number of Attachable COBs
V1	C3	---	1
V2	C1C2	(1,1;2,2)	3
V3	C10C11C14C15	(10,1;11,2),(11,3;14,1),(14,1;15,2)	3

There is no path with 0 attachable COBs. But path V2 may be extended by COB C13 if the input to C13 from the path V3 ((11,5;13,2)) is disregarded. Thus from condition (b) of step 3, V3 is chosen as the second 2-dimensional COB.

Step 4: Edges originating from C10, C11, C14 and C15 are deleted from E".

Steps 2 and 3:

Path	COB Sequence	Step 2 Computable Path set E´	Step 3 Number of Attachable COBs
V1	C3	---	1
V2	C1C2C13C20	(1,1;2,2),(2,2;13,1),(13,1;20,1)	0

V2 is chosen as the third 2-dimensional COB from condition (a) of step 3.

Step 4: Edges originating from C1, C2, C13 and C20 are deleted from E".

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Steps 2 and 3:

Path	COB Sequence	Step 2 Computable Path set E´	Step 3 Number of Attachable COBs
V1	C3C4C12C17	(3,1;4,2),(4,5;12,2),(4,5;17,1)	1
V2	C3C4C7C8C18	(3,1;4,2),(4,3;7,1),(7,1;8,2),(8,2;18,1)	0
V3	C3C4C16C19	(3,1;4,2),(4,2;16,1),(16,1;19,1)	0

V2 is chosen as the fourth 2-dimensional COB from condition (a) of step 3.

Step 4: Edges originating from C3, C4, C7, C8 and C18 are deleted from E".

Steps 2 and 3:

Path	COB Sequence	Step 2 Computable Path set E´	Step 3 Number of Attachable COBs
V1	C12C17	(12,1;17,1)	0
V2	C16C19	(16,1;19,1)	0

V1 is chosen as the fifth 2-dimensional COB from condition (a) of step 3.

Step 4: Edges originating from C12 and C17 are deleted from E".

Steps 2 and 3:

Path	COB Sequence	Step 2 Computable Path set E´	Step 3 Number of Attachable COBs
V1	C16C19	(16,1;19,1)	0

V1 is chosen as the sixth 2-dimensional COB.

Step 4: After edges originating from C16 and C19 are deleted from E",

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

E" = Ø. Therefore procedure terminates.

The resultant 2-dimensional COB cover is shown in Fig. 3.5. In order to obtain 3-dimensional COB cover, the r-register algorithm is applied to Fig. 3.5. The result is 3 3-dimensional COBs, as shown in Fig. 3.6. Applying the r-register algorithm repeatedly, 4 to 9-register COBs are found, as shown in Fig. 3.7.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

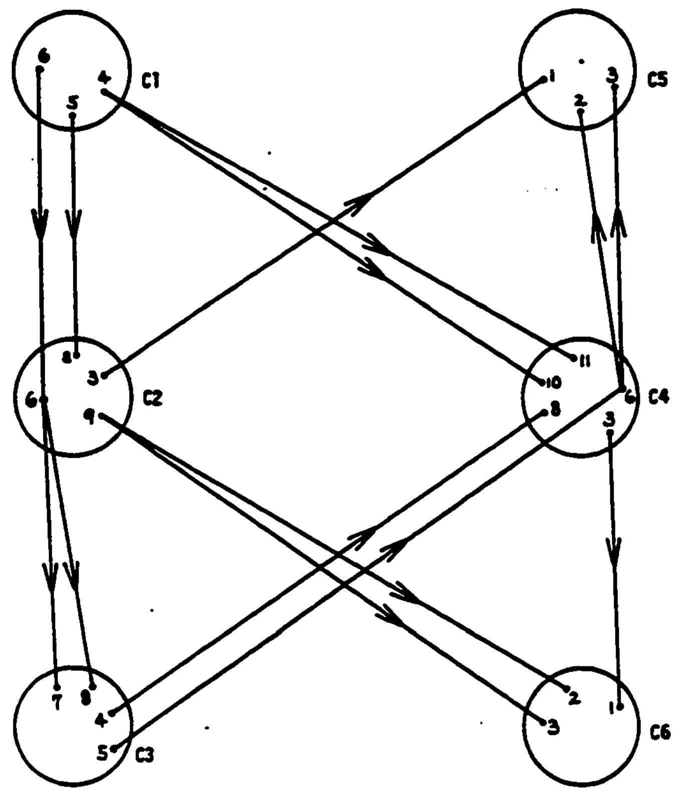


Fig. 3.5. 2-dimensional COB cover of the 4-point FFT graph.

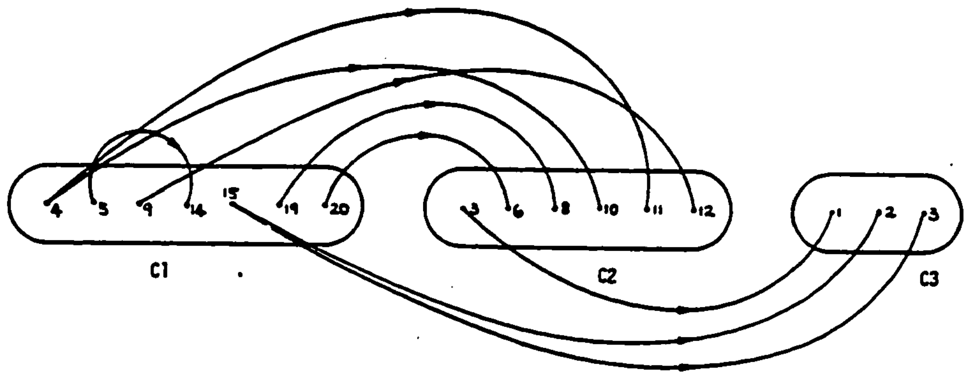
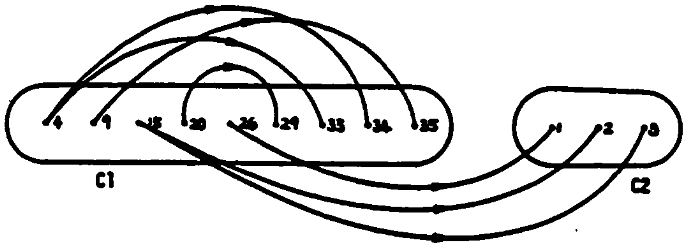
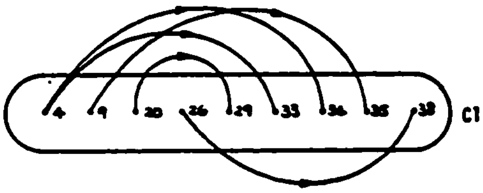


Fig. 3.6. 3-dimensional COB cover of the 4-point FFT graph.

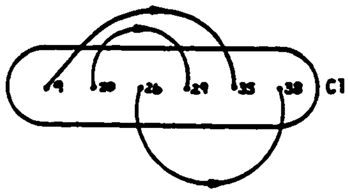
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



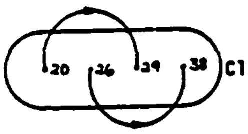
4-dimensional COB cover



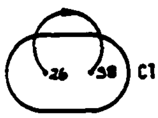
5-dimensional



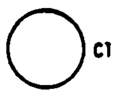
6-dimensional



7-dimensional



8-dimensional



9-dimensional

Fig. 3.7. 4- through 9-dimensional COB covers of the 4-point FFT graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

APPLICATIONS

The intent of this chapter is to illustrate the concept of a primitive COB and its integration with the principles developed in earlier chapters. A primitive COB is defined and illustrated by an example in Section 4.1. In Sections 4.2, various primitive COBs suitable for Hadamard transform (HT), and their codes using the algorithms developed in Chapters 2 and 3 are obtained. In Section 4.3, HT implementations using these primitive COBs are investigated. Sections 4.4 and 4.5 repeat this exercise for fast Fourier transform (FFT).

4.1 Primitive COB

Many signal processing algorithms have graphs which may be partioned into a set of identical subgraphs. This property greatly simplifies the software implementation of signal processing algorithms. As Morris illustrates in [19], automatic generation of digital signal processing software is possible by making use of the regular structure of the algorithm. In such software generation, a computational kernel is identified and is used repeatedly to compute the complete algorithm. This computational kernel is usually the smallest repeatable subgraph possible.

A primitive COB is a computational kernel, but not necessarily the smallest repeatable subgraph. A given graph may be covered using many

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

different primitive COBs. A computational graph may also be implemented using different primitive COBs simultaneously. The following example illustates this idea through the implementation of a binary computational graph of 63 points (shown in Fig. 4.1) using a set of primitive COBs.

The procedure begins by finding a set of primitive COBs as shown in Fig. 4.2. The complete graph can be implemented in two different ways. One way is to use the primitive COB of 3 points and another way is to use the primitive COB of 7 points. The results of these two different implementations are shown in Fig. 4.3. In addition to different implementations, each primitive COB can be implemented on machines with different numbers of registers to compare the execution time for the complete graph. These implementations and their complexities are shown in Fig. 4.4 and Table 4.1.

Table 4.1. Dependence of the complexities of two different implementations upon the number of registers in the machine.

Implementation using 3 point primitive COB

# of registers	Time/COB	Eta	# of COBs	Total Time for the graph
1	16	5.33	21	336
2	13	4.33	21	273
3	13	4.33	21	273

Implementation using 7 point primitive COB

# of registers	Time/COB	Eta	# of COBs	Total Time for the graph
1	36	5.14	9	324
2	30	4.29	9	270
3	27	3.86	9	243

An implementation of the complete graph may also be devised using the algorithm developed earlier. The time complexity of this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

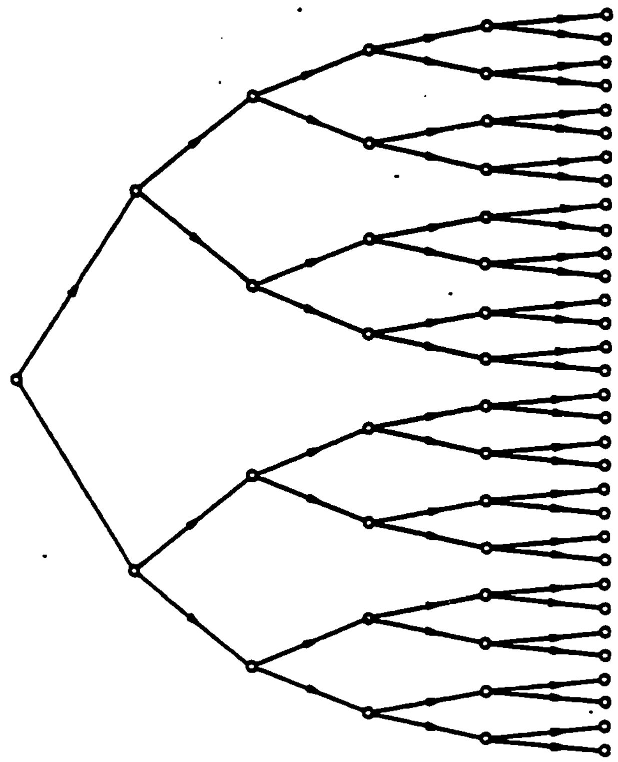


Fig. 4.1. A computational graph with 63 points.

Fig. 4.2. Primitive COBs for implementation of the graph in Fig. 4.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

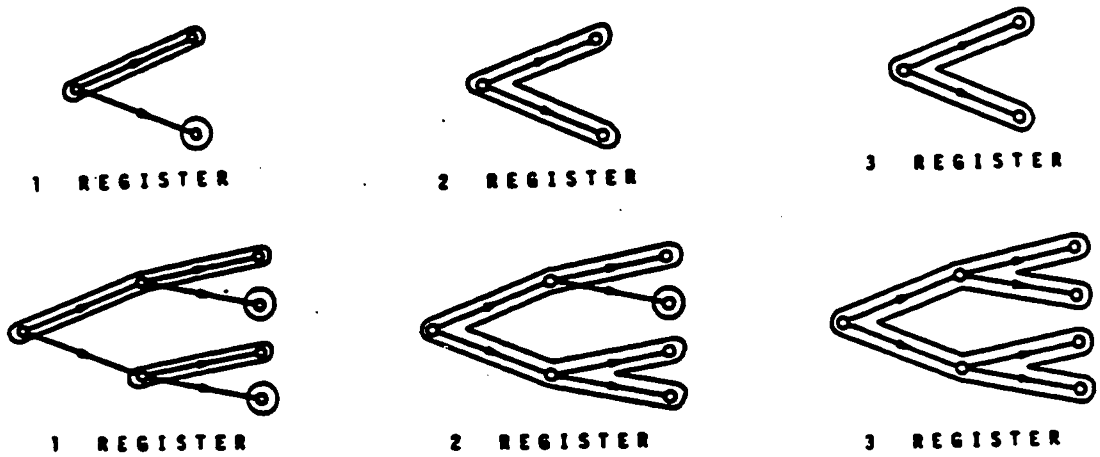


Fig. 4.3. Various implementations of 3- and 7-point primitive COBs.

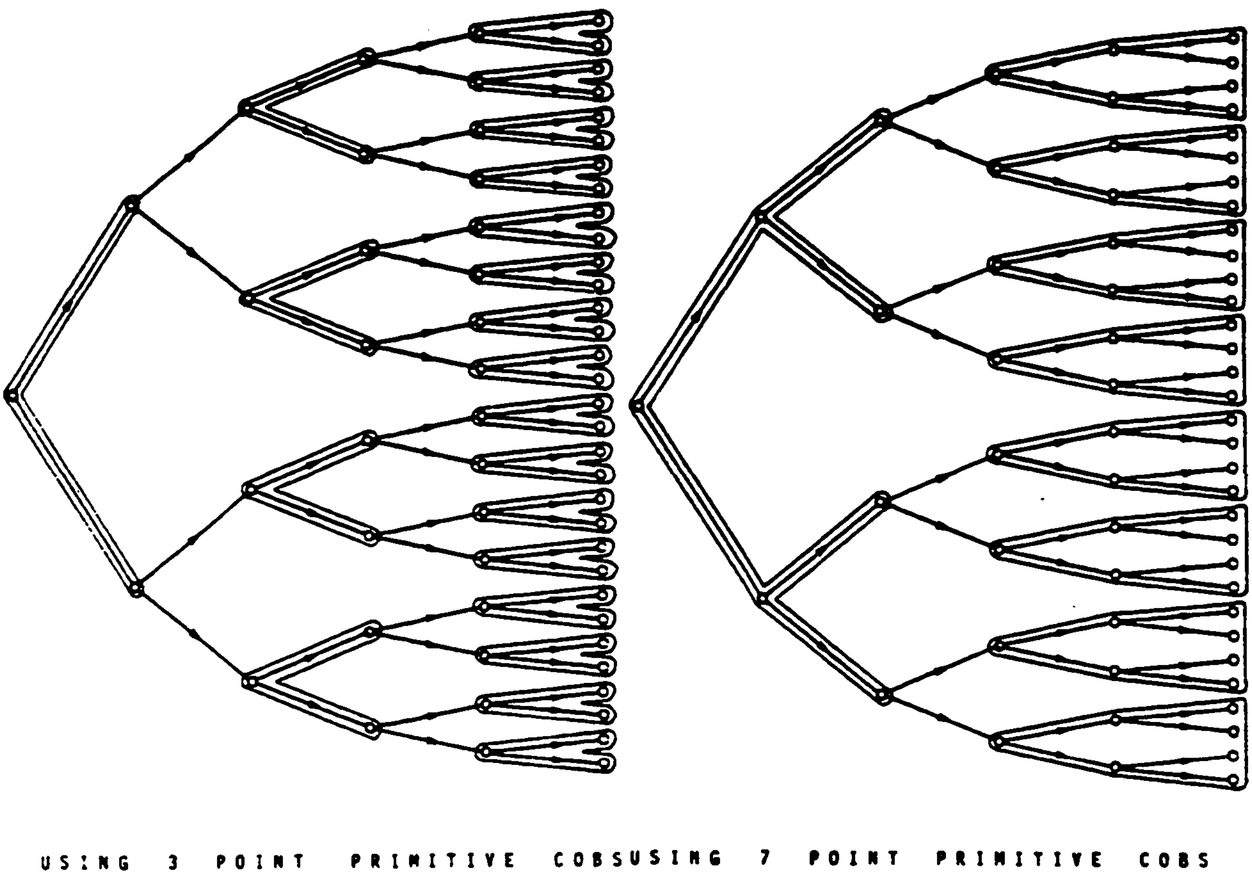


Fig. 4.4. Cover of complete graph using 3- and 7-point primitive COBs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

implementation will generally be smaller than those with primitive COBs. This is because the usage of the primitive COBs artificially severs some links in the graph without caring for its global implications. However, the increase in time is marginal as Table 4.2 shows.

Table 4.2. Comparison of implementations with and without primitive COBs.

# of registers	Time for implementation without primitive COBs	% increase in time using COB of 3 points	7 points
1	316	6.33	2.53
2	264	3.41	2.27
3	241	13.28	0.83

One may note that increasing the number of registers generally reduces the time gap between the implementations with and without primitive COBs. The only exception to this occurs when the primitive COB is too small to fully utilize all the available registers.

In actual software implementation, time complexities due to decision-making and arithmetic operations for loop control are assumed to be eliminated by the use of in-line-code. Therefore, whether primitive COB approach is used or not, the code sizes are approximately the same. However, the design of a large non-structural errorless software for an algorithm may be a time consuming task without primitive COBs. With primitive COBs, the software can be generated automatically and with ease since the portion of the software related to the primitive COB can be used repeatedly to form a complete code.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2 Hadamard Transform(HT)

In this section, a description of efficient 1-,2-, and 3-register implementations for primitive COBs of 2X2, 4X3, 8X4, and 16X5 points useful for computation of HT is presented. These primitive COBs are then used to compute a 2^{12}-point HT.

4.2.1 1-Register Implementation of Primitive COBs

Primitive COBs of 2X2, 4X3, 8X4, and 16X5 points which would be used here for implementing the Hadamard transform are shown in Fig. 4.5. These primitive COBs were chosen for their superior performance (with reference to their time complexity) from many different primitive COBs that might be useful for implementing a Hadamard transform. Figure 4.5 also shows a 1-dimensional COB cover obtained through algorithm of Chapter 2 and lists the associated codes for a machine with only one accumulator. Using the formula derived in Chapter 2, one obtains the total number of operations in the case of a 2^n length HT as:

Total # of operation= # of COBs + # of points + # of points with outdegree $\geq$ 2 + # of COBs with terminal point outdegree > 2

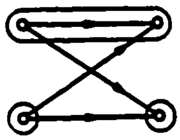
$$= (2+n)2^{n-1} + (n+1)2^n + n2^n + 2^n$$

$$=(5n+6)2^{n-1}.$$

Since execution time for Tload, Tmop(+,-), and Tstore are assumed to be 2 units each, total execution time is

$$\text{Total Time}=(5n+6)2^{n-1} \times 2 = (5n+6)2^n.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



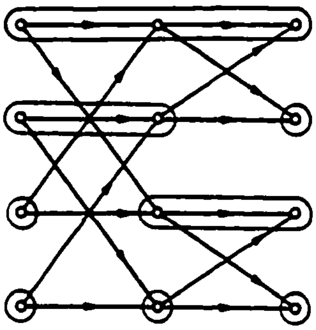
computation

```
R1 ← I1
R1 ← R1 + I3
T0 ← R1
R1 ← I0
R1 ← R1 + I2
T1 ← R1
R1 ← R1 + T0
O0 ← R1
R1 ← T1
R1 ← R1 - T0
O1 ← R1
```

code

2X2 primitive COB

Rn: n-th register
In: n-th input data from memory
On: n-th output data to memory
Tn: n-th temporary scratch pad memory location



computation

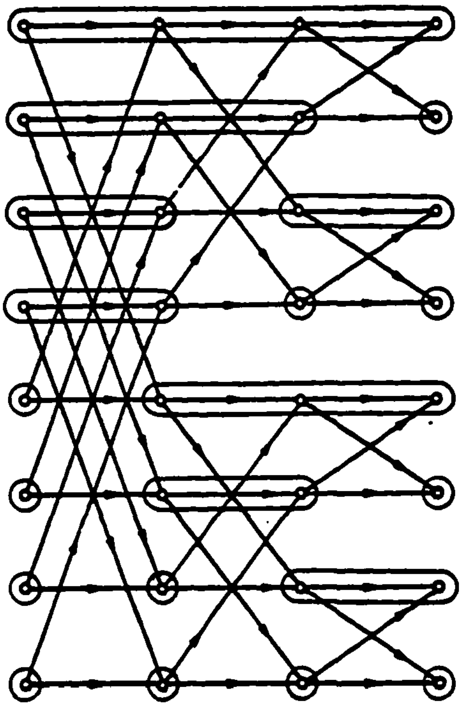
```
R1 ← I3
R1 ← R1 + I7
T0 ← R1
R1 ← I1
R1 ← R1 + I5
T1 ← R1
R1 ← R1 + T0
T2 ← R1
R1 ← T1
R1 ← R1 - T0
T0 ← R1
R1 ← I2
R1 ← R1 + I6
T1 ← R1
R1 ← I0
R1 ← R1 + I4
T3 ← R1
R1 ← R1 + T1
T4 ← R1
R1 ← R1 + T2
O0 ← R1
R1 ← T4
R1 ← R1 - T2
O1 ← R1
R1 ← T3
R1 ← R1 - T1
T2 ← R1
R1 ← R1 + T0
O2 ← R1
R1 ← T1
R1 ← R1 - T0
O3 ← R1
```

code

4X3 primitive COB

Fig. 4.5a. 1-register implementation of HT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



computation

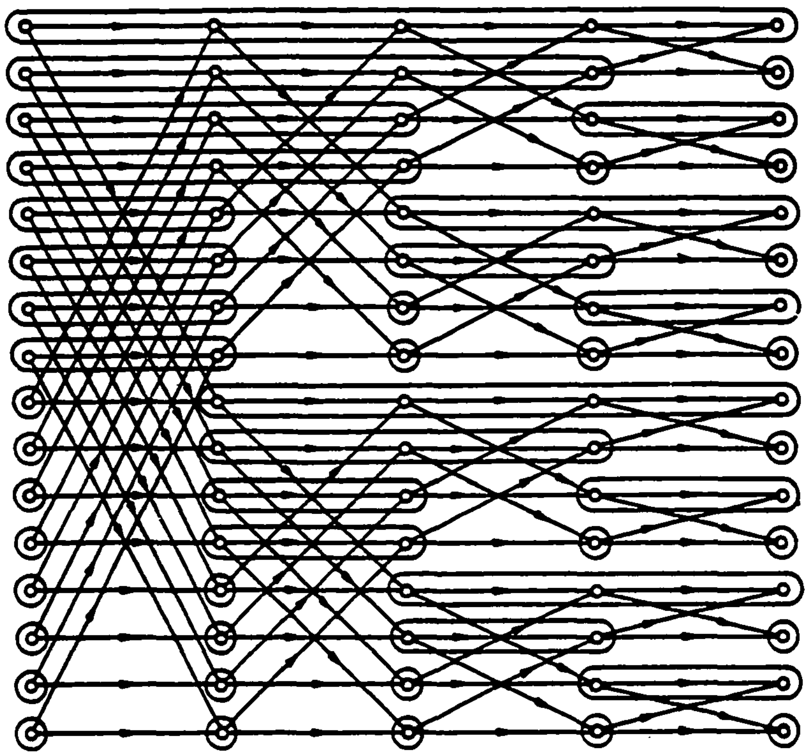
```
R1  ← I7
R1  ← R1 + I15
T0  ← R1
R1  ← I3
R1  ← R1 + I11
T1  ← R1
R1  ← R1 + T0
T2  ← R1
R1  ← T1
R1  ← R1 - T0
T0  ← R1
R1  ← I5
R1  ← R1 + I13
T1  ← R1
R1  ← I1
R1  ← R1 + I9
T3  ← R1
R1  ← R1 + T1
T4  ← R1
R1  ← R1 + T2
T5  ← R1
R1  ← T3
R1  ← R1 - T1
T1  ← R1
R1  ← R1 + T0
T3  ← R1
R1  ← T1
R1  ← R1 - T0
T0  ← R1
R1  ← I6
R1  ← R1 + I14
T1  ← R1
R1  ← I2
R1  ← R1 + I10
T6  ← R1
R1  ← R1 + T1
T7  ← R1
R1  ← T6
R1  ← R1 - T1
T1  ← R1
R1  ← I4
R1  ← R1 + I12
T6  ← R1
R1  ← I0
R1  ← R1 + I8
T8  ← R1
R1  ← R1 + T6
T9  ← R1
R1  ← R1 + T7
T10 ← R1
R1  ← R1 + T5
O0  ← R1
R1  ← T10
R1  ← R1 - T5
```

```
O1  ← R1
R1  ← T4
R1  ← R1 - T2
T2  ← R1
R1  ← T9
R1  ← R1 - T7
T4  ← R1
R1  ← R1 + T2
O2  ← R1
R1  ← T4
R1  ← R1 - T2
O3  ← R1
R1  ← T8
R1  ← R1 - T6
T2  ← R1
R1  ← R1 + T1
T4  ← R1
R1  ← R1 + T3
O4  ← R1
R1  ← T4
R1  ← R1 - T3
O5  ← R1
R1  ← T2
R1  ← R1 - T1
T1  ← R1
R1  ← R1 + T0
O6  ← R1
R1  ← T1
R1  ← R1 - T0
O7  ← R1
```

code

Fig. 4.5b. 1-register implementation of HT(continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



16X5 primitive COB computation

Fig. 4.5c. 1-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
R1  - I15
R1  - R1 + I31
T0  - R1
R1  - I7
R1  - R1 + I23
T1  - R1
R1  - R1 + T0
T2  - R1
R1  - T1
R1  - R1 + T0
T0  - R1
R1  - I11
R1  - R1 + I27
T1  - R1
R1  - I3
R1  - R1 + I19
T3  - R1
R1  - R1 + T1
T4  - R1
R1  - R1 + T2
T5  - R1
R1  - T4
R1  - R1 - T2
T2  - R1
R1  - T3
R1  - R1 - T1
T1  - R1
R1  - R1 + T0
T3  - R1
R1  - T1
R1  - R1 - T0
T0  - R1
R1  - I13
R1  - R1 + I29
T1  - R1
R1  - I5
R1  - R1 + I21
T4  - R1
R1  - R1 + T1
T6  - R1
R1  - T4
R1  - R1 - T1
T1  - R1
R1  - I9
R1  - R1 + I25
T4  - R1
R1  - I1
R1  - R1 + I17
T7  - R1
R1  - R1 + T4
T8  - R1
R1  - R1 + T6
T9  - R1
R1  - R1+T5

T10 - R1
R1  - T9
R1  - R1 - T5
T5  - R1
R1  - T8
R1  - R1 - T6
T0  - R1
R1  - R1 + T2
T8  - R1
R1  - T6
R1  - R1 - T2
T2  - R1
R1  - T7
R1  - R1 - T4
T4  - R1
R1  - R1 + T1
T6  - R1
R1  - R1 + T3
T7  - R1
R1  - T6
R1  - R1 - T3
T3  - R1
R1  - T4
R1  - R1 - T1
T1  - R1
R1  - R1 + T0
T4  - R1
R1  - T1
R1  - R1 + T0
T0  - R1
R1  - I14
R1  - R1 + I34
T1  - R1
R1  - I6
R1  - R1 + I22
T6  - R1
R1  - R1 + T1
T9  - R1
R1  - T6
R1  - R1 - T1
T1  - R1
R1  - I10
R1  - R1 + I26
T6  - R1
R1  - I2
R1  - R1 + I18
T11 - R1
R1  - R1 + T6
T12 - R1
R1  - R1 + T9
T13 - R1
R1  - T12
R1  - R1 - T9
T9  - R1

R1  - T11
R1  - R1 - T6
T6  - R1
R1  - R1 + T1
T11 - R1
R1  - T6
R1  - R1 - T1
T1  - R1
R1  - I12
R1  - R1 + I28
T6  - R1
R1  - I4
R1  - R1 + I20
T12 - R1
R1  - R1 + T6
T14 - R1
R1  - T12
R1  - R1 - T6
T6  - R1
R1  - I8
R1  - R1 + I24
T12 - R1
R1  - I0
R1  - R1 + I16
T15 - R1
R1  - R1 + T12
T16 - R1
R1  - R1 + T14
T17 - R1
R1  - R1 + T13
T18 - R1
R1  - R1 + T10
O0  - R1
R1  - T18
R1  - R1 - T10
O1  - R1
R1  - T17
R1  - R1 - T13
T10 - R1
R1  - R1 + T5
O2  - R1
R1  - T10
R1  - R1 - T5
O3  - R1
R1  - T16
R1  - R1 - T14
T5  - R1
R1  - R1 + T9
T10 - R1
R1  - R1 + T8
O4  - R1
R1  - T10
R1  - R1 - T8
O5  - R1

R1  - T5
R1  - R1 - T9
T5  - R1
R1  - R1 + T2
O6  - R1
R1  - T5
R1  - R1 - T2
O7  - R1
R1  - T15
R1  - T12
T2  - R1
R1  - R1 + T6
T5  - R1
R1  - R1 + T11
T8  - R1
R1  - R1 + T7
O8  - R1
R1  - T8
R1  - R1 - T7
O9  - R1
R1  - T5
R1  - R1 - T11
T5  - R1
R1  - R1 + T3
O10 - R1
R1  - T5
R1  - R1 - T3
O11 - R1
R1  - T2
R1  - R1 - T6
T2  - R1
R1  - R1 + T1
T3  - R1
R1  - R1 + T4
O12 - R1
R1  - T3
R1  - R1 + T4
O13 - R1
R1  - T2
R1  - R1 + T1
T1  - R1
R1  - R1 + T0
O14 - R1
R1  - T1
R1  - R1 + T0
O15 - R1
```

16X5 primitive COB code

Fig. 4.5d. 1-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We denote the time complexity of a COB implementation per point by Eta. Eta is a measure of the efficiency of the implementation. A smaller Eta indicates a better implementation. In a 1-register implementation of a primitive COB of $2^n X(n+1)$ points, Eta = 5 + 1/(n+1).

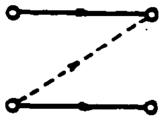
4.2.2 2-Register Implementation of Primitive COBs

Primitive COBs shown earlier may also be covered using 2-dimensional COBs and implemented on a machine using 2 accumulators efficiently. The results, obtained from the algorithm of Chapter 3, are shown in Fig. 4.6. To compute the execution time of these implementations, an inspection of their structure is in order. The odd and even indexed points of the first n-1 stages of these highly regular implementations are mere duplicates of one lower size implementation. The last stage of the implementation is made up of three different types of butterflies shown in Fig. 4.7. These butterflies occur in a regular cycle of Types-1,2,1,3,1,2,1,3,.... A Type-1 butterfly computation involves only one Load, but two Mop(+) and Stores each. Its complexity (complexity of computing the two end-points) is thus 10 time units. Type-2 butterfly involves a Rop(+), a Mop(+) and two Stores. It also saves the storage of one of the source points. Its effective complexity is thus 5 time units. Finally, the Type-3 butterfly involves two Mop(+) and Stores but it converts the Store of source point into a Copy thus having an effective complexity of 7 time units.

From the above discussion, the time complexity of $2^n X (n+1)$ point primitive COB, C(n), is given by:

$$C(n) = 2C(n-1) + 10x2^{n-2} + 5x2^{n-3} + 7x2^{n-3} \quad ; \quad n > 2.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

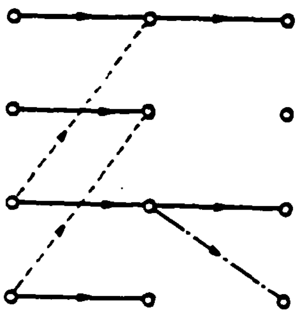


computation

```
R1 ← I1
R1 ← R1 + I3
R2 ← I0
R2 ← R2 + I2
T0 ← R2
R2 ← R2 + R1
O0 ← R2
R1 ← R1 - T0
O1 ← R1
```

code

2 X 2 primitive COB



computation

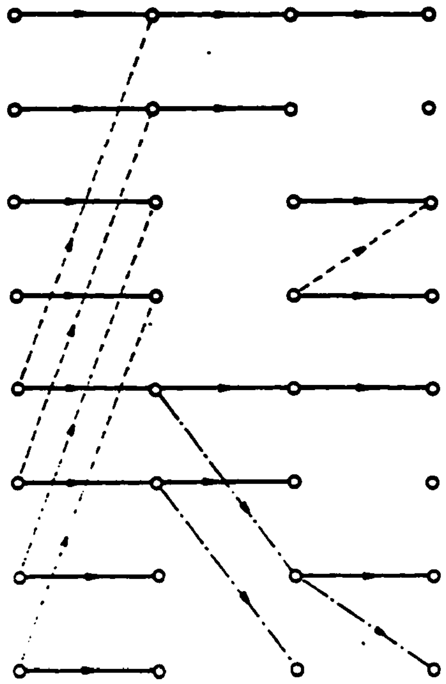
```
R1 ← I3
R1 ← R1 + I7
R2 ← I1
R2 ← R2 + I5
T0 ← R2
R2 ← R2 + R1
T1 ← R2
R1 ← R1 - T0
T1 ← R1
R1 ← I2
R1 ← R1 + I6
R2 ← I0
R2 ← R2 + I4
T2 ← R2
R2 ← R2 +R1
R1 ← R1 - T2
T2 ← R2
R2 ← R2 + T1
O0 ← R2
R2 ← T2
R2 ← R2 - T1
O1 ← R2
R2 ← R1
R1 ← R1 + T0
O2 ← R1
R2 ← R2 - T0
O3 ← R2
```

code

4 X 3 primitive COB

Fig. 4.6a. 2-register implementation of HT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



computation

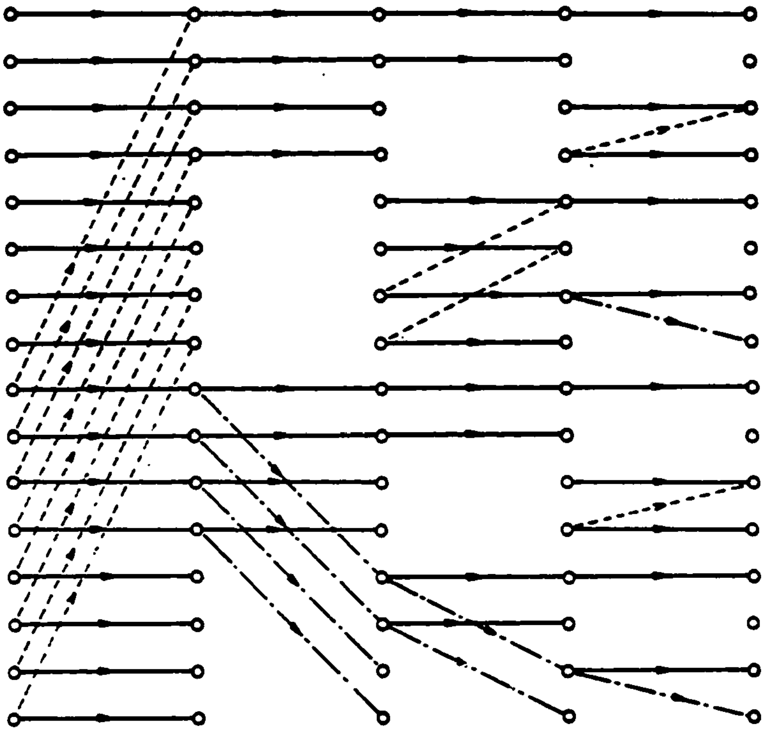
```
R1 ← I7
R1 ← R1 + I15
R2 ← I3
R2 ← R2 + I11
T0 ← R2
R2 ← R2 + R1
R1 ← R1 + T0
T1 ← R2
T0 ← R1
R1 ← I5
R1 ← R1 + I13
R2 ← I1
R2 ← R2 + I9
T2 ← R2
R2 ← R2 + R1
R1 ← R1 - T2
T2 ← R2
R2 ← R2 + T1
T3 ← R2
R2 ← R1
R1 ← R1 + T0
R2 ← R2 - T0
T0 ← R1
T4 ← R2
R1 ← I6
R1 ← R1 + I14
R2 ← I2
R2 ← R2 + I10
T5 ← R2
R2 ← R2 + R1
R1 ← R1 - T5
T5 ← R1
T0 ← R2
R1 ← I4
R1 ← R1 + I12
R2 ← I0
R2 ← R2 + I8
T7 ← R2
R2 ← R2 + R1
R1 ← R1 - T7
T7 ← R2
R2 ← R2 + T6
T8 ← R2
R2 ← R2 + T3
O0 ← R2
R2 ← R1
R1 ← R1 + T5
R2 ← R2 - T5
T5 ← R1
R1 ← R1 + T0
O4 ← R1
R1 ← R2
R2 ← R2 + T4
O6 ← R2
R1 ← R1 - T4
O7 ← R1
R1 ← T8
R1 ← R1 - T3
O1 ← R1
R1 ← T7
R1 ← R1 - T6
R2 ← T2
R2 ← R2 + T1
T1 ← R1
R1 ← R1 + R2
O2 ← R1
R2 ← R2 - T1
O3 ← R2
R1 ← T5
R1 ← R1 - T0
O5 ← R1
```

code

8X4 primitive COB

Fig. 4.6b. 2-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



16X5 primitive COB computation

Fig. 4.6c. 2-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
R1  - I15
R1  - R1 + I31
R2  - I7
R2  - R2 + I23
T0  - R2
R2  - R2 + R1
R1  - R1 - T0
T0  - R2
T1  - R1
R1  - I11
R1  - R1 + I27
R2  - I3
R2  - R2 + I19
T2  - R2
R2  - R2 + R1
R1  - R1 - T2
T2  - R2
R2  - R2 + T0
T0  - R2
R2  - R1
R1  - R1 + T1
R2  - R2 - T1
T1  - R1
T3  - R2
R1  - I13
R1  - R1 + I29
R2  - I5
R2  - R2 + I21
T4  - R2
R2  - R2 + R1
R1  - R1 - T4
T4  - R2
T5  - R1
R1  - I9
R1  - R1 + I25
R2  - I1
R2  - R2 + I17
T6  - R2
R2  - R2 + R1
R1  - R1 - T6
T6  - R2
R2  - R2 + T4
T7  - R2
R2  - R2 + T8
T9  - R2
R2  - R1
R1  - R1 + T5
R2  - R2 - T5
T5  - R1
R1  - R1 + T1
T11 - R1
R1  - R2
R2  - R2 + T3

R1  - R1 - T3
T3  - R2
T12 - R1
R1  - I14
R1  - R1 + I30
R2  - I6
R2  - R2 + I22
T13 - R2
R2  - R2 + R1
R1  - R1 - T13
T13 - R2
T14 - R1
R1  - I10
R1  - R1 + I26
R2  - I2
R2  - R2 + I18
T15 - R2
R2  - R2 + R1
R1  - R1 - T15
T15 - R2
R2  - R2 + T13
T16 - R2
R2  - R1
R2  - R2 + T14
R1  - R1 - T14
T14 - R2
T17 - R1
R1  - I12
R1  - R1 + I28
R2  - I4
R2  - R2 + I20
T18 - R2
R2  - R2 + R1
R1  - R1 - T18
T18 - R2
T19 - R1
R1  - I8
R1  - R1 + I24
R2  - I0
R2  - R2 + I16
T20 - R2
R2  - R2 + R1
R1  - R1 - T20
T20 - R2
R2  - R2 + T18
T21 - R2
R2  - R2 + T8
T22 - R2
R2  - R2 + T9
O0  - R2
R2  - R1
R2  - R2 + T19
R1  - R1 - T19
T19 - R2

R2  - R2 + T1
T23 - R2
R2  - R2 + T11
O8  - R2
R2  - R1
R1  - R1 + T17
R2  - R2 - T17
T17 - R1
R1  - R1 + T3
O12 - R1
R1  - R2
R2  - R2 + T12
R1  - R1 - T12
O14 - R2
O15 - R1
R1  - T21
R1  - R1-T16
R2  - T7
R2  - R2 - T8
T7  - R1
R1  - R1 + R2
R2  - R2 - T7
O2  - R1
O3  - R2
R1  - T2
R1  - R1 - T0
R2  - T6
R2  - R2 - T4
T2  - R2
R2  - R2 + R1
R1  - R1 - T2
T2  - R2
T4  - R1
R1  - T15
R1  - R1 - T13
R2  - T20
R2  - R2 - T18
T6  - R2
R2  - R2 + R1
R1  - R1 - T6
T6  - R2
R2  - R2 + T2
O4  - R2
R2  - R1
R1  - R1 + T4
R2  - R2 - T4
O6  - R1
O7  - R2
R1  - T5
R1  - R1 - T1
R2  - T19
R2  - R2 - T14
T1  - R2
R2  - R2 + R1

R1  - R1 - T1
O10 - R2
O11 - R1
R1  - T22
R1  - R1 - T9
O1  - R1
R1  - T6
R1  - R1 - T2
O5  - R1
R1  - T23
R1  - R1 - T11
O9  - R1
R1  - T17
R1  - R1 - T3
O13 - R1
```

16X5 primitive COB code

Fig. 4.6d. 2-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

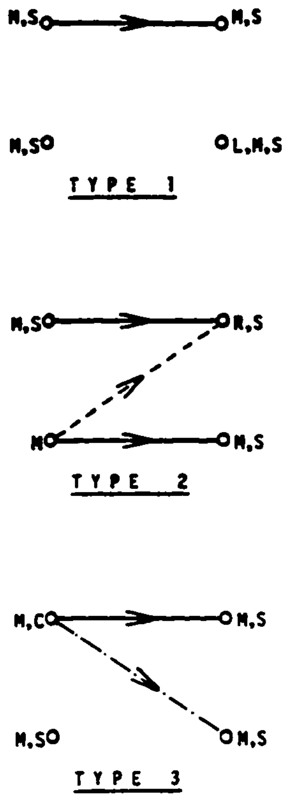


Fig. 4.7. The three types of butterfly implementations prevalent in the 2-register implementation of HT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The solution of this difference equation yields the following closed form expression for the time complexity of the two register implementation.

$$C(n) = (4n + 4.75)2^{n} \qquad ; \quad n > 2.$$

Also in this case, Eta = 4 + 0.75/(n+1).

4.2.3 3-Register Implementation of Primitive COBs

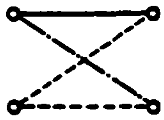
The 3-dimensional COB cover of the primitive COBs under consideration and the associated implementations on a machine with 3 accumulators are shown in Fig. 4.8.

4.2.4 0-Register and Infinite-Register Implementations

If an implementation computes each graph point independently, without any regard for the graph structure, we call it a 0-register implementation here. Each HT computational point is calculated by first loading an operand, then adding to or subtracting from it an operand located in memory, and storing the result back into the memory, taking a total of 6 units of time. Each computational point, in this case, is a COB. Since a 0-register implementation is constructed without any effort to minimize memory related operation, its execution time is the worst possible.

Since every computational point takes 6 units of time, total time for a computational graph may be obtained by merely multiplying the number of computational points in the graph by 6.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

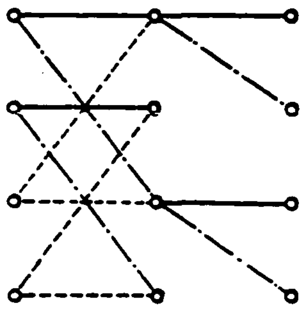


computation

```
R1 ← I1
R1 ← R1 + I3
R2 ← I0
R2 ← R2 + I2
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
O0 ← R2
O1 ← R1
```

code

2X2 primitive COB



computation

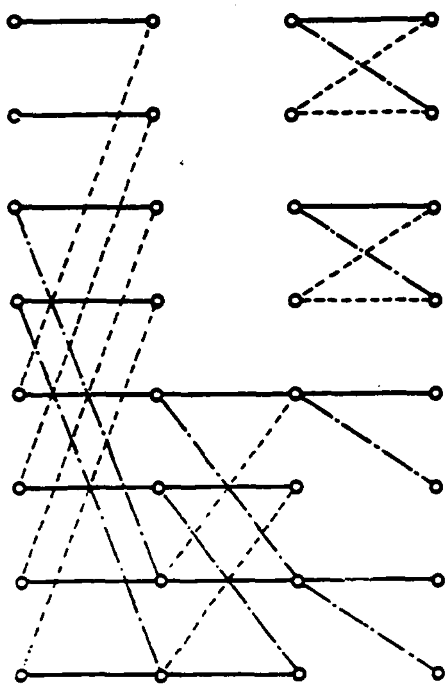
```
R1 ← I3
R1 ← R1 + I7
R2 ← I1
R2 ← R2 + I5
R3 ← R2
R2 ← R2 + R1
R1 ← R1 + R3
T0 ← R2
T1 ← R1
R1 ← I2
R1 ← R1 + I6
R2 ← I0
R2 ← R2 + I4
R3 ← R2
R2 ← R1 + R1
R1 ← R1 - R3
R3 ← R2
R2 ← R2 + T0
R3 ← R3 - T0
O0 ← R2
O1 ← R3
R3 ← R1
R1 ← R1 + T1
R3 ← R1 + T1
O2 ← R1
O3 ← R3
```

code

4X3 primitive COB

Fig. 4.8a. 3-register implementation of HT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



computation

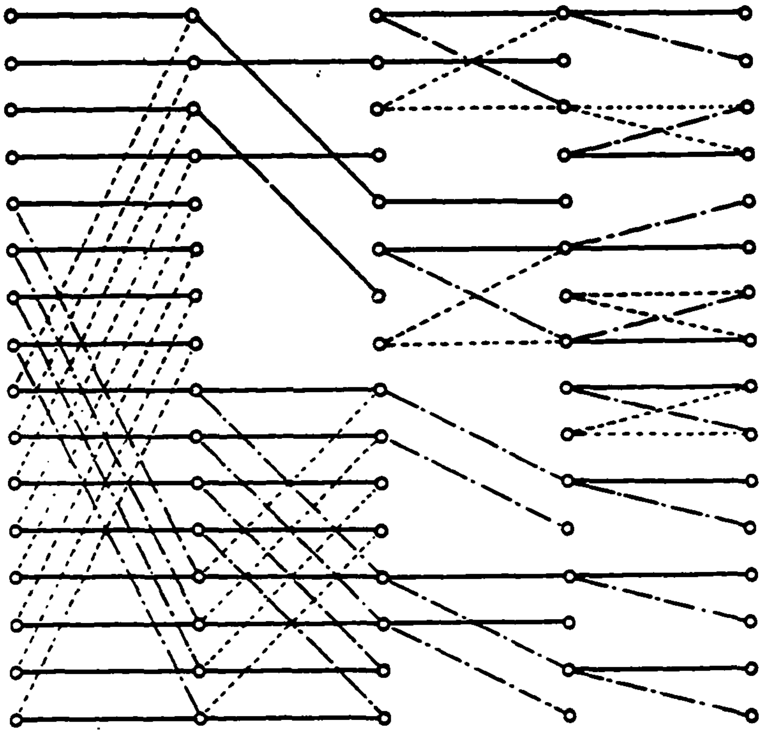
8X4 primitive COB

```
R1 ← I7
R1 ← R1 + I15
R2 ← I3
R2 ← R2 + I11
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
T0 ← R2
R2 ← I5
R2 ← R2 + I13
R3 ← I1
R3 ← R2 + I9
T1 ← R3
R3 ← R3 + R2
R2 ← R2 - T1
T1 ← R3
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
T1 ← R2
T2 ← R1
R1 ← I6
R1 ← R1 + I14
R2 ← I2
R2 ← R2 + I10
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
T4 ← R2
R2 ← I4
R2 ← R2 + I12
R3 ← I0
R3 ← R3 + I8
T5 ← R3
R3 ← R3 + R2
R2 ← R2 - T5
T5 ← R3
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
R3 ← R2
R2 ← R2 + T2
R3 ← R3 - T2
O4 ← R2
O5 ← R3
R3 ← R1
R1 ← R1 + T3
T3 ← R3 - T3
O6 ← R1
O7 ← R3
R1 ← T1
R1 ← R1 - T0
R2 ← T5
R2 ← R2 - T4
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
O2 ← R2
O3 ← R1
R1 ← T1
R1 ← R1 + T0
R2 ← T5
R2 ← R2 + T4
R3 ← R2
R2 ← R2 + R1
R1 ← R1 - R3
O0 ← R2
O1 ← R1
```

code

Fig. 4.8b. 3-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



16X5 primitive COB computation

Fig. 4.8c. 3-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
R1  ← I15
R1  ← R1 + I31
R2  ← I7
R2  ← R2 + I23
R3  ← R2
R2  ← R1 + R1
R1  ← R1 - R3
T0  ← R2
R2  ← I11
R2  ← R2 + I27
R3  ← I3
R3  ← R3 + I19
T1  ← R3
R3  ← R3 + R2
R2  ← R2 - T1
T1  ← R3
R3  ← R3 + T0
T2  ← R3
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T3  ← R2
T4  ← R1
R1  ← I13
R1  ← R1 + I29
R2  ← I5
R2  ← R2 + I21
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T5  ← R2
R2  ← I9
R2  ← R2 + I25
R3  ← I1
R3  ← R3 + I17
T6  ← R3
R3  ← R3 + R2
R2  ← R2 - T6
T6  ← R3
R3  ← R3 + T5
T7  ← R3
R3  ← R3 + T2
T8  ← R3
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T9  ← R2
R2  ← R2 + T3
T10 ← R2
R2  ← R1
R1  ← R1 + T4
R2  ← R2 - T4
T4  ← R1

T11 ← R2
R1  ← I4
R1  ← R1 + I30
R2  ← I6
R2  ← R2 + I22
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T12 ← R2
R2  ← I10
R2  ← R1 + I26
R3  ← I2
R3  ← R3 + I18
T13 ← R3
R3  ← R3 + R2
R2  ← R2 - T13
T13 ← R3
R3  ← R3 - T12
T14 ← R3
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T15 ← R2
T16 ← R1
R1  ← I12
R1  ← R1 + I28
R2  ← I4
R2  ← R2 + I20
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
T17 ← R2
R2  ← I8
R2  ← R2 + I24
R3  ← I0
R3  ← R3 + I16
T18 ← R2
R3  ← R3 + R2
R2  ← R2 - T18
T19 ← R3
R3  ← R3 - T17
T20 ← R3
R3  ← R3 - T7
T21 ← R3
R3  ← R2
R2  ← R2 + R1
R1  ← R2 - R3
T22 ← R2
R2  ← R2 + T15
R3  ← R2
R2  ← R2 + T10
R3  ← R3 - T10
O8  ← R2
O9  ← R3

R2  ← R1
R1  ← R1 + T16
R2  ← R2 - T16
R3  ← R1
R1  ← R1 + T4
R3  ← R3 - T4
O12 ← R1
O13 ← R3
R3  ← R2
R2  ← R2 + T11
R3  ← R3 - T11
O14 ← R2
O15 ← R3
R1  ← T13
R1  ← R1 + T12
R2  ← T19
R2  ← R2 + T17
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
R3  ← R2
R2  ← R2 + T8
R2  ← R3 - T8
O0  ← R2
O1  ← R3
R2  ← T7
R2  ← R2 - T2
R3  ← R2
R2  ← R2 - R1
R1  ← R1 + R3
O2  ← R1
O3  ← R2
R1  ← T1
R1  ← R1 - T0
R2  ← T16
R2  ← R2 + T5
R3  ← R2
R2  ← R2 + R1
R1  ← R1 - R3
R3  ← R2
R2  ← R2 - T21
R3  ← R3 + T21
R2  ← T20
R2  ← R2 - T14
R3  ← R1
R1  ← R1 - R2
R2  ← R2 + R3
O6  ← R2
O7  ← R1
R1  ← T9
R1  ← R1 - T3
R2  ← T22
R2  ← R2 - T15
R3  ← R2

R2  ← R2 + R1
R1  ← R1 - R3
O10 ← R2
O11 ← R1
```

16X5 primitive COB code

Fig. 4.8d. 3-register implementation of HT (continued).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The time complexity and the Eta value for the 0-register implementation of a $2^n X(n+1)$ point primitive COB is given by:

$$\text{Total time} = 6(n+1)2^n \quad , \quad \text{Eta} = 6.$$

When an infinite number of registers is available, three different types of butterfly computations exist. Each initial stage butterfly is computed using 2 Loads, 1 Copy, and 2 Mop(+,-). Each final stage butterfly is computed using 1 Copy, 2 Rop(+,-), and 2 Stores. Each of the remaining butterflies is computed using 1 copy and 2 Rop(+,-). These computations are shown in Fig. 4.9. Thus, for 2^n length primitive COB, 2^n Loads, 2^n Mop(+,-), $n2^n$ Rop(+,-), $n2^{(n-1)}$ Copies, and 2^n Stores are required. Accordingly, the total time for a $2^n X(n+1)$ point primitive COB implementation on an infinite accumulator machine is:

$$\text{Total time} = 6(2^n) + 3n(2^{n-1}) \quad , \quad \text{Eta} = 1.5 + 4.5/(n+1).$$

4.2.5 Consolidation of Results

Comparing the Eta values of 1-, 2- and infinite-register implementations with that of 0-register implementation, one can note that for large values of n, by merely structuring the order of computation, one can obtain savings of 16.7%, 33% and 75% respectively, in the HT execution time compared to non-structured 0-register case.

Table 4.3 lists the complexities of various implementations of primitive COBs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 4.3 Complexities of various implementations of HT primitive COBs.

COB Size = 2 X 2 # of computational points: 4

# R	Load	Mop(+,-)	Store	Copy	Rop(+,-)	Time	Eta
0	4	4	4	0	0	24	6.0
1	3	4	4	0	0	22	5.5
2	2	3	3	0	1	17	4.25
3-∞	2	2	2	1	2	15	3.75

COB Size = 4 X 3 # of computational points: 12

# R	Load	Mop(+,-)	Store	Copy	Rop(+,-)	Time	Eta
0	12	12	12	0	0	72	6
1	8	12	12	0	0	64	5.33
2	5	10	9	1	2	51	4.25
3	4	8	6	4	4	44	3.67
5-∞	4	4	4	4	8	36	3.00

COB Size = 8 X 4 # of computational points: 32

# R	Load	Mop(+,-)	Store	Copy	Rop(+,-)	Time	Eta
0	32	32	32	0	0	192	6
1	20	32	32	0	0	168	5.25
2	12	27	24	3	5	134	4.18
3	12	18	16	8	14	114	3.56
9-∞	8	8	8	12	24	84	2.63

COB size= 16 X 5 # of computational points: 80

# R	Load	Mop(+,-)	Store	Copy	Rop(+,-)	Time	Eta
0	80	80	80	0	0	480	6
1	48	80	80	0	0	416	5.2
2	28	68	60	8	12	332	4.15
3	24	50	43	20	30	284	3.55
17-∞	16	16	16	32	64	192	2.40

As can be seen from Table 4.3, choosing a larger primitive COB improves the efficiency of algorithm. But in practice, one should consider both the improvement in time and the increase in code (program) size to determine the appropriate primitive COB. A primitive COB should be small enough so that the code for it can be generated without difficulty. At the same time, it should be large enough to utilize all

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

available registers efficiently. The following example illustrates the choice of a primitive COB in a machine with three accumulators.

Example: Suppose the target CPU contains 3 accumulators. In order to fully utilize all available registers, 3-register implementations of primitive COBs should be used. Based on the parameters listed in Table 4.4, an appropriate primitive COB may be chosen as follows.

Table 4.4 Change in the values of Eta for various primitive COBs

primitive COB	Time/COB	Eta	% decrease in Eta from previous line
2X2	15	3.75	--
4X3	44	3.67	2.13
8X4	114	3.56	3.00
16X5	284	3.55	0.28

The code size for a COB is directly proportional to the execution time for the COB. Thus as we go down the COBs listed in Table 4.4, the code size multiplies by a factor of approximately 1.5 each time. An inspection of Table 4.4 now shows that a primitive COB of 8X4 points is probably the best in these circumstances. If the size of this COB is further increased, it has a marginal effect on Eta but the code size increases by 149%.

4.3 Implementation of a complete HT through primitive COBs

This section discusses the issues involved in the implementation of a complete graph through an example of 2^{12} length HT. If the primitive COBs of types discussed earlier with $2^t X(t+1)$ points are used to cover a 2^n length HT, then a total of $n2^{(n-1)}/(t+1)$ primitive COBs would be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

required. Thus the odd divisors of (t+1) should divide n. In the present case, it rules out the 16X5 primitive COB. The 2048 X 12 point primitive COB also need not be considered because of its excessive code size.

If one uses the 2X2 primitive COBs, the resultant implementation has six computing stages, each with 2048 COBs. All six stages may be made identical by rearranging the graph of HT [20],[21]. Thus the code for each stage is identical except for the memory locations of input and output data. Further, every pair of consecutive stages may have an in-place code. Therefore, software for the entire 2^{12} length HT may consist of the code for the first 2 stages placed in a loop, thus reducing the code size by approximately 66.7%.

Use of 4X3 primitive COBs similarly results in 4 identical stages, each with 1024 COBs. Use of a loop reduces the code size by approximately 50%.

Use of 8X4 primitive COBs implies 3 identical stages each with 512 COBs. Use of a loop is not beneficial in this case.

Finally, if 32X6 primitive COBs are used for the implementation, there are only 2 identical stages each with 128 COBs. As in the earlier case, a loop is not useful.

The execution time of the complete HT depends upon both the size of the primitive COB used and the number of registers available to implement each primitive COB. Table 4.5 and Fig.4.9 display the results obtained. While calculating the code sizes, the possibility of using the in-place algorithm is kept in mind. One may conclude from these that the computational time of HT is largely independent of the choice of primitive COB. Also, using a machine with more than three registers

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

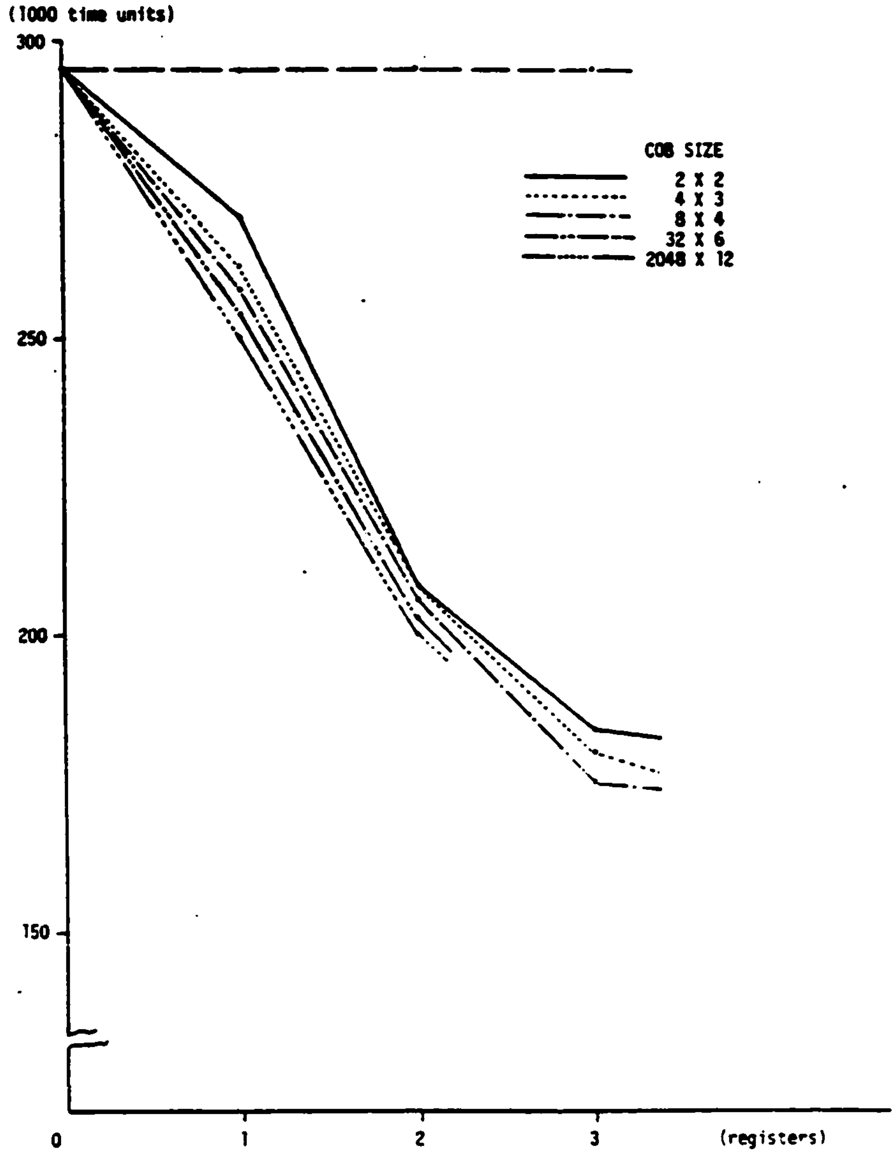


Fig. 4.9. Time complexity of various implementations of 2^{12} length HT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is not justified in this case. A good trade off between the time and the code size is obtained when one uses a 3-register machine and a 2X2 primitive COB.

Table 4.5. Implementation of 2^{12} length HT

0-register implementation

prim. COB	# of primitive COBs within HT	Time per prim. COB	Total Time for HT	Eta of Prim. COB	Code Size for HT
2X2	12288	24	294912	6.00	98304
4X3	4096	72	294912	6.00	147456
8X4	1536	192	294912	6.00	294912
32X6	256	1152	294912	6.00	294912
2048X12	2	147456	294912	6.00	294912

1-register implementation

2X2	12288	22	270336	5.50	90112
4X3	4096	64	262144	5.33	131072
8X4	1536	168	258048	5.25	258048
32X6	256	992	253952	5.17	253952
2048X12	2	124928	249856	5.08	249856

2-register implementation

2X2	12288	17	208896	4.25	69632
4X3	4096	51	208896	4.25	104448
8X4	1536	134	205824	4.18	205824
32X6	256	792	202752	4.13	202752
2048X12	2	99846	199692	4.06	199692

3-register implementation

2X2	12288	15	184320	3.75	61440
4X3	4096	44	180224	3.67	90112
8X4	1536	114	175104	3.56	175104

4.4 Fast Fourier Transform(FFT)

In this section, two primitive COBs for FFT are presented and implemented using 0 to infinite number of registers. They are then applied to implement a 2^8 length FFT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4.1 2-Point Primitive COB

The graph shown in Fig. 4.10 computes two complex points in the FFT graph and hence is termed as the 2-point primitive COB. The 1- and 2-register implementations and the associated codes are shown in Fig. 4.11. The implementation of this small primitive COB does not change if the number of registers is increased beyond 2.

4.4.2 4-Point Primitive COB

The graph of a 4-point primitive COB is shown in the Fig. 3.2. Figures 3.3 through 3.11 then show its 1- through 9-register implementations. A further increase in the number of registers does not affect the implementation of this COB.

4.4.3 Consolidation of Results

The complexities of the two FFT COBs and, in particular, their dependence on the number of registers in the machine is shown in Table 4.6. These results indicate that while using the 2-point COB, a 2-register machine will perform optimally and even for the 4-point COB increasing the number of registers beyond 5 has very little effect on the time complexity.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

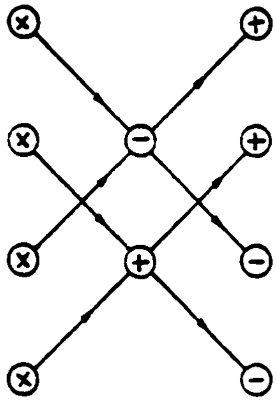
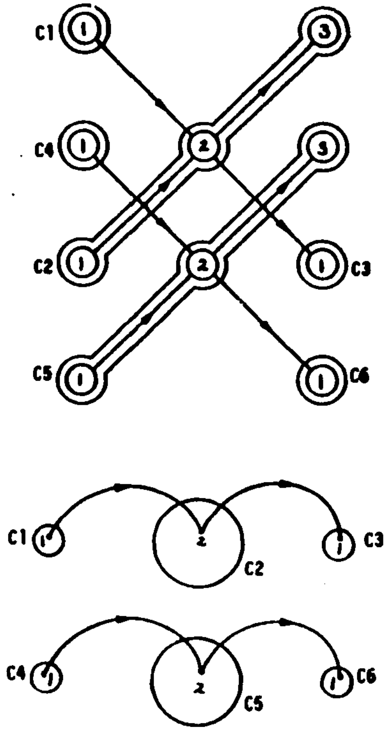


Fig. 4.10. Computational graph of 2-point FFT.

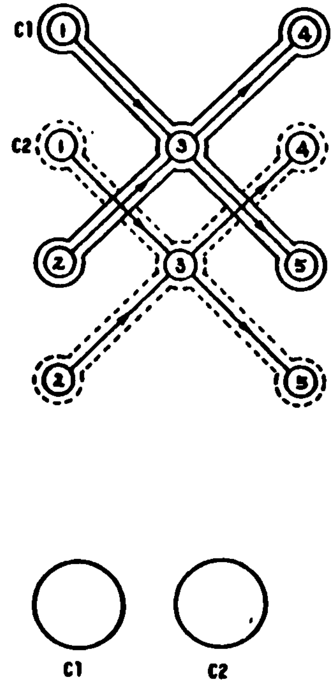
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.




```
R1 ← I2
R1 ← R1 X I4
T0 ← R1
R1 ← I3
R1 ← R1 X I5
R1 ← R1 - T0
T0 ← R1
R1 ← R1 + I0
O0 ← R1
R1 ← I1
R1 ← R1 - T0
O2 ← R1
R1 ← I3
R1 ← R1 X I4
T0 ← R1
R1 ← I2
R1 ← R1 X I5
R1 ← R1 + T0
T0 ← R1
R1 ← R1 + I1
O1 ← R1
R1 ← I1
R1 ← R1 - T0
O3 ← R1
```

code

1-dimensional COB cover




```
R1 ← I2
R1 ← R1 X I4
R2 ← R1
R1 ← I3
R1 ← R1 X I5
R1 ← R1 - R2
R2 ← R1
R1 ← R1 + I0
O0 ← R1
R1 ← I1
R1 ← R1 - R2
O2 ← R1
R1 ← I3
R1 ← R1 X I4
R2 ← R1
R1 ← I2
R1 ← R1 X I5
R1 ← R1 + R2
R2 ← R1
R1 ← R1 + I1
O1 ← R1
R1 ← I1
R1 ← R1 - R2
O3 ← R1
```

code

2-dimensional COB cover

Fig. 4.11. 1- and 2-dimensional COB cover of 2-point FFT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 4.6 Complexities of various implementations of FFT primitive COBs.

COB size = 2 X 1   # of complex computational points = 2

R	Load	Mop(+,-)	Mop(x)	Rop(+,-)	Copy	Store	Time	Eta	% dec.in Eta
0	10	6	4	0	0	10	68	34	-
1	6	6	4	0	0	8	56	28	17.65
2-	4	4	4	2	2	4	44	22	21.43

COB size = 4 X 2   # of complex computational points = 8

R	Load	Mop(+,-)	Mop(x)	Rop(+,-)	Copy	Store	Time	Eta	% dec.in Eta
0	40	24	16	0	0	40	272	34.000	-
1	20	24	16	0	0	32	216	27.000	20.59
2	13	17	16	7	6	19	175	21.875	18.98
3	12	14	16	10	8	16	166	20.750	5.14
4	12	11	16	13	8	14	159	19.875	4.22
5	12	8	16	16	8	12	152	19.000	4.40
6	11	8	16	16	9	11	149	18.625	1.97
7	10	8	16	16	10	10	146	18.250	2.01
8	9	8	16	16	11	9	143	17.875	2.05
9-	8	8	16	16	12	8	140	17.500	2.10

4.5 Implementation of 2^8 Length FFT

An implementation of 2^8 length FFT using 2-point primitive COBs results in 8 identical computational stages of 128 COBs each. As for the case of HT, a pair of these stages may be calculated in-place [20,21]. The size of code may therefore be reduced by 75% by using the loop as described in Section 4.3. Similarly, use of 4-point primitive COBs produces 4 identical stages of 64 COBs each. Use of a loop, in this case, will reduce the code size by 50%. Table 4.7 and Fig. 4.11 display various factors affected by the choice of a particular implementation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 4.7 Implementation of 2^8 length FFT

0-register implementation

Prim. COB	# of prim.COBs within FFT	Time per Prim.COB	Total time for FFT	Eta of prim.COB	Code size for FFT
2X1	1024	68	69632	34	17408
4X2	256	272	69632	34	34816

1-register implementation

2X1	1024	56	57344	28	14336
4X2	256	216	55296	27	27648

2-register implementation

2X1	1024	44	45056	22	11264
4X2	256	175	44800	21.875	22400

3-register implementation

2X1	1024	44	45056	22	11264
4X2	256	166	42496	20.75	21248

4-register implementation

2X1	1024	44	45056	22	11264
4X2	256	159	40704	19.875	20352

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

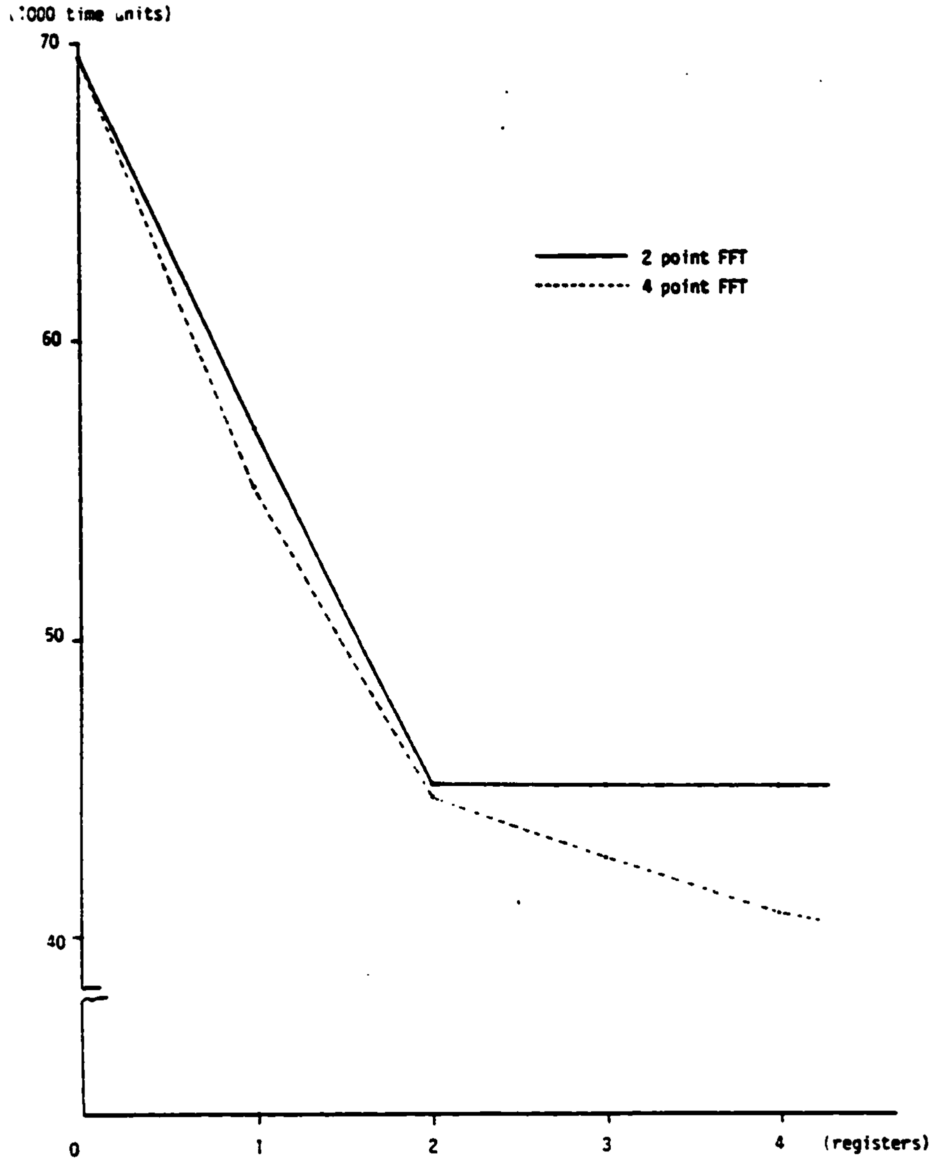


Fig. 4.12. Time complexity of various implementations of 2^8 length FFT.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

CONCLUSIONS

This chapter reviews the results obtained during this work. After summarizing the useful results in Section 5.1, and their applications in Section 5.2, future research areas are identified in Section 5.3.

5.1 Summary of Selected Results

This work for the first time provides the means to design an implementation of a given arbitrary computational graph, while taking into account the number of accumulators available in the processor. The 1-register algorithm of Chapter 2 can be applied to form a time efficient algorithm for the graph implemented on a one accumulator processor. Since most of the general purpose microprocessors available today have one accumulator, the results obtained here are universally useful. This 1-register algorithm is extended to r-register algorithm in Chapter 3. Given a machine containing n general purpose registers, any computational graph can be subjected to 1- and r-register algorithms to form a time efficient implementation. Furthermore, since most signal processing algorithms contain regular structures, a computational kernel, called a primitive COB here, may be used repeatedly to cover the complete graph, as shown in Chapter 4. The primitive COB may be subjected to the algorithms derived in this thesis to obtain its efficient code for any given processor. By repeating this basic code, one may then obtain an efficient code for the complete graph.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The results obtained in Chapter 4 point out several important facts. First, for a given computational graph, the time complexity decreases exponentially as the number of registers increases (See Figs. 4.9 and 4.11). This result implies that the increase in the number of registers after a certain point does not yield a profitable decrease in time complexity. (For Hadamard transform, this is a modest three accumulator architecture). Consequently, an arbitrary increase in the number of accumulators in processor design is not justified since the cost of hardware inflates very rapidly as the number of accumulators increases. Another important result obtained is that the size of primitive COB does not affect the time complexity significantly, as long as it is large enough to fully utilize all available registers. One may thus choose a small and efficient primitive COB, so that writing the code for it is a trivial task.

5.2 Significance of the Results

The importance of this work is mainly due to the wide applicability of the algorithms developed in Chapters 2 and 3. These algorithms enable one to design a time efficient code by giving due consideration to the hardware architecture, in particular, the number of registers contained in the CPU. These algorithms enable one to utilize the hardware capabilities to their fullest extent, thus improving the performance without any additional cost.

Another potential application of this research is to provide means to evaluate various architectures with respect to a given algorithm. The procedures of Chapters 2 and 3 allow one to systematically study the trade offs between various factors such as the time complexity, hardware

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

complexity and code size. This enables one to choose a good engineering design in most practical situations.

Finally, this work also brings out the concept of a primitive COB. A primitive COB can be used for automatic generation of software for large signal processing problems and to reduce the code size of an algorithm without sacrificing time efficiency. It may also have a significant impact on the design of special purpose parallel processing hardware for signal processing applications.

5.3 Suggestions for Further Work

The verification on an actual multi-accumulator machine of the various implementations obtained here is highly desirable. It was not possible to carry this out mainly due to the time limitation and also because of the lack of good multi-accumulator processors. Since most of the available microprocessors have architectures geared towards high-level language implementations rather than numerical applications, it is necessary to design a multi-accumulator hardware for this verification. Such a hardware design would use bit-slice microprocessors AM2901 or AM2903 [22-24], since they have a sufficient number of registers for our purpose and belong to a family that has a large number of support ICs.

Another potential area for future research is the investigation of the relationship between a graph structure and its ultimate implementation on a finite register SISD machine. In particular, one may be able to restructure the computational graph without affecting the final results, such that the restructured graph may have a highly efficient implentation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally it should be mentioned that the r-dimensional COB model may not yield optimum results in some cases and merits further attention.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

[1] S. Winograd, "On computing the Discrete Fourier Transform," Proc. Nat. Acad. Sci., U.S.A., vol. 73, pp. 1005-1006, Apr. 1976.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of Complex Fourier Series," Math. of Com., vol. 19, pp. 296-301, 1965.

[3] L. R. Morris, "A comparative study of time efficient FFT and WFTA programs for general purpose computers," IEEE Trans. Acoust., Speech and Signal Processing, vol. ASSP-26, no.2, pp. 141-150, Apr. 1978.

[4] H. D. Toong and A. Gupta, "An architectural comparison of contemporary 16-bit microprocessors," IEEE Micro, vol. 1, pp. 26-37, May 1981.

[5] Component Data Catalog, Intel Corporation, Santa Clara, CA, 1982.

[6] Z80 Microcomputer Data Book, Mostek Corp., Carrollton, TX, 1981.

[7] Electronic Device Division Data Catalog, Rockwell International, Anaheim, CA, 1981.

[8] Microprocessor Data Manual, Motorola Inc., Austin, TX, 1981.

[9] J. P. Anderson, "A note on some compiling algorithms," Comm. ACM, vol. 7, no. 3, pp. 149-150, Mar. 1964.

[10] I. Nakata, "On compiling algorithms for arithmetic expressions," Comm. ACM, vol. 10, no. 8, pp. 492-494, Aug. 1967.

[11] N. M. Brenner, "Fast Fourier Transform of externally stored data," IEEE Trans. Audio Electroacoust., vol. AU-17, no. 2, pp. 128-132, June 1969.

[12] P. S. Naidu, "FFT of externally stored data," IEEE Trans. Acoust., Speech, and Signal Processing, vol. ASSP-26, no. 5, pp. 473, 1970.

[13] J. O. Eklundh, "A fast computer method for matrix transposition," IEEE Trans. Computers, vol. C-21, no. 7, pp. 801-803, July 1972.

[14] P. S. Naidu, "Fast matrix transpose computer implementation," Signal Processing, North Holland Publishing Company, pp. 457-459, Mar. 1982.

[15] H. Nawab and J. H. McClellan, "Bounds on the minimum number of data transfers in WFTA and FFT programs," IEEE Trans. Acoust., Speech and Signal Processing, vol. ASSP-27, no. 4, pp. 394-398, Aug. 1979.

[16] M. J. Flynn, "Very high-speed computing system," IEEE Proc., vol. 54, no. 12, pp. 1901-1909, Dec. 1966.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[17] Peter M. Kogge, The Architecture of Pipelined Computers, New York: McGraw-Hill, Inc., 1981.

[18] J. L. Pfaltz, Computer Data Structures, New York: McGraw-Hill, Inc., 1977.

[19] L. R. Morris, "Automatic generation of time efficient digital signal processing software," IEEE Trans. Acoust., Speech and Signal Processing, vol. ASSP-25, no. 1, pp. 74-79, February 1977.

[20] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing, Englewood Cliff, NJ: Prentice-Hall, 1975.

[21] L. R. Rabiner and B. Gold, Theory and Application of Signal Processing, Englewood Cliff, NJ: Prentice-Hall, 1975.

[22] J. Mick and J. Brick, Bit-Slice Microprocessor Design, New York: McGraw-Hill, Inc., 1980.

[23] G. J. Myers, Digital System Design with LSI Bit-Slice Logic, New York: Wiley Interscience, 1980.

[24] D. E. White, Bit-Slice Design: Controller and ALUs, New York: Garland STPM Press, 1981.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.