This is to certify that the

thesis entitled

Computation Design Alternatives
with
Microprocessor-Based Systems

presented by

Sigurd Leland Lillevik

has been accepted towards fulfillment
of the requirements for

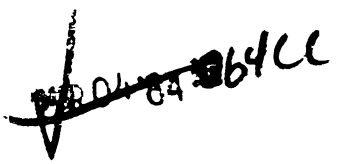
Ph.D. degree in Electrical
Engineering

David Fisher

Major professor

Date July 27, 1978

O-7639



COMPUTATIONAL DESIGN ALTERNATIVES WITH MICROPROCESSOR-BASED SYSTEMS

By

Sigurd Leland Lillevik

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering
and
Systems Science

1978

ABSTRACT

COMPUTATIONAL DESIGN ALTERNATIVES
WITH
MICROPROCESSOR-BASED SYSTEMS

By

Sigurd Leland Lillevik

For a microprocessor-based system, the computational section reduces to combinations of four elemental computational design alternatives (CDA's). The described research examines and characterizes these CDA's and develops a structured approach to computational section design which incorporates a rigorous, theoretic foundation.

The DIRECT CDA incorporates a single microprocessor (μP) and memory, so it executes adds and multiplies in software. Next, the second CDA contains a μP, memory, and arithmetic unit (AU). This AU CDA, then, performs multiplies in hardware. For the third, a μP, memory, and calculator chip (CALC) comprise the CALC CDA, and it executes adds and multiplies with the CALC chip. Finally, several μP's and memories in a Master/Slave arrangement implement the multiple-μP (mμP) CDA. It applies the concept of parallel execution, but performs all adds and multiplies in software (like the DIRECT).

A common set of attributes facilitates comparison of CDA's. Using such attributes, Multiattribute Utility Theory (MUT) assesses a numeric quantity, the utility, to represent each CDA's usefulness. Thus, design involves selecting the CDA with the greatest utility.

When attributes satisfy eight axioms plus utility and preferential independence definitions, utility assessment embodies an additive utility function; i.e.,

$$u(\bar{x}) = \sum_{i=1}^{N} k_i u_i(x_i) = \text{utility}$$

$$\bar{x} = (x_1, x_2, x_3, \ldots, x_N) = \text{consequence}$$

$$N = \text{number of attributes}$$

$$u_i(x_i) = \text{marginal utility of attribute } x_i$$

$$k_i = \text{scaling constants, where } \sum_{i=1}^{N} k_i = 1.$$

Application of MUT to three examples--linear regression, matrix inversion, and fast Fourier transform computations--illustrated the use of the additive utility function and led to CDA characterization. Here, a consequence consists of an ordered-triplet whose components correspond to the precision, execution-time, and cost of a CDA; thus, N = 3. For instance, the first example produced the following marginal utilities: DIRECT = (60, 00, 100), AU = (00, 100, 53), CALC = (100, 19, 00), and mμP = (33, 97, 62). The above function with weight vector $\bar{k}$ = (0.3, 0.3, 0.3) finds a weighted average of marginal utilities; DIRECT = 55, AU = 51, CALC = 40, and mμP = 64; hence, the "best" design consists of the mμP CDA. In characterizing the CDA's, the examples revealed that the AU exhibited fastest execution-time. Also, the mμP's parallel execution surpassed the DIRECT's speed even though they both implement computations entirely in software. But this increase varies significantly with the type of example and the severity of the Master's overhead program. The CALC executes slower than the DIRECT or mμP for single/double-precision, but faster for triple/quadruple-precision. All costs climb

with greater problem dimensions due to increased data (not program) memory, yet the mμP costs grow more rapidly because of repetitive hardware and software. Finally, costs vary little with changes in precision.

So designers will be able to synthesize more advanced μP-based systems through judicious use of these elemental CDA's, their characteristics, and MUT. Also, these characteristics indicate that new LSI devices should include enhanced computational features. With MUT methodology developers can now examine and characterize future μP-based systems as technology advances.

ACKNOWLEDGMENTS

No single individual has influenced my professional growth and development more than Dr. P. David Fisher, my major professor. For the past several years he has guided and motivated my career through his exemplary enthusiasm, hard work, aggressiveness, and superior knowledge of the field. Mere words can not express my appreciation. Perhaps I can best thank him by extending this type of friendship to others throughout my professional career.

Also, I wish to express my gratitude to my committee members for their guidance of my doctoral program: Dr. S. R. Crouch, Dr. J. J. Forsyth, Dr. J. B. Kreer, and Dr. D. K. Reinhard. And to my wife, Sandi, I extend my appreciation for her typing of the dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I INTRODUCTION

Technological advances made within the electronics industry during the early and mid 1970's stimulated the rapid development and broad acceptance of microprocessor-based systems. Today, these systems span a wide range of applications with considerable variation in computational requirements. To meet these requirements, the electronics industry introduced a host of large-scale integrated (LSI) devices that perform diverse computations; for example, linear regression, matrix inversion, and fast Fourier transform computations. Even though a plethora of hardware/software computational design alternatives confront today's system designer, no structured approach exists for them to sythesize optimal computational sections. So the purpose of the research reported here is to investigate and characterize computational design alternatives (CDA's), and to develop a more rigorous and structured approach to design which incorporates a firm, theoretic foundation.

Using the results of this research, designers of microprocessor-based (μP-based) systems will be able to synthesize advanced systems with improved performance. Second, these results will benefit LSI chip designers by influencing what properties they will embody in the next generation of devices. Since some features assist computations while others detract, these new LSI devices will contain the desirable characteristics and minimize the action of the undesirable ones. Third, the results of this study will assist developers of μP-based systems by providing them with

the methodology to investigate and characterize future systems as technology advances. Because the theoretic development begins at the axiom level, they may easily modify the methodology to fit their particular goals.

Throughout the research reported here the study achieves several objectives as delineated below:

1) survey the present semiconductor market for LSI devices that facilitate computations,
2) generalize the properties of such devices,
3) identify a basic set of elemental computational design alternatives,
4) determine common attributes for each member in the set,
5) find techniques to evaluate such attributes,
6) develop a decision mechanism for comparing CDA's which uses these attributes,
7) illustrate the use of this decision mechanism with representative and contemporary engineering applications, and
8) use the examples to characterize the properties of each CDA in the elemental set.

To meet these objectives, the study only considers elementary hardware/software tradeoffs because of straightforward and tractable analysis. But since the overall methodology remains the same with optimal, or advanced, hardware devices and software algorithms, no loss in generality occurs.

In Chapter II a survey of today's semiconductor market identifies those devices and their characteristics that facilitate computations. Specifically, the discussion focuses on μP's, solid-state memories, and

LSI support chips. These results establish the motivation and rationale for the entire investigation. With these devices, Chapter III defines a basic set of elemental CDA's which when combined may realize the computational section of a generalized μP-based system. To facilitate comparison of each member in this set, it determines attributes common to the members that reflect the important features, qualities, and characteristics of each CDA, and it describes techniques to evaluate them. Next, Chapter IV presents a decision mechanism for selecting the "best" CDA which uses the attributes found in the previous chapter. And this involves eight axioms upon which the theoretic foundation rests, several independence definitions, and a central theorem which delineates a specific decision function. Finally, the three examples in Chapter V--linear regression, matrix inversion, and fast Fourier transform computations--use the results of the previous two chapters to clarify the techniques used to evaluate attributes, and to elucidate the specific decision function in selecting the "best" CDA. Additionally, this chapter explores the effect of alternative algorithms. Not only do these examples typify representative and contemporary engineering problems, but they vary widely in their mathematical complexity. During this process the applications exemplify the overall procedure of this investigation and, moreover, they lead to conclusions which characterize each CDA.

CHAPTER II CHARACTERISTICS OF COMPUTATIONAL DEVICES

Recent advances in processing technology led to LSI devices that exhibit enhanced features at low cost. Consequently, μP-based systems now function in a myriad of applications which range from medicine[1] to communications[2] to agriculture[3-5] to business.[6] And other examples abound. Many μP-based systems require advanced mathematical capabilities and a multitude of computational design alternatives may accomplish these calculations. In this chapter a survey of today's semiconductor market identifies those LSI devices and their characteristics that facilitate computations. Specifically, the discussion focuses on the performance trends of μP's, solid-state memories, and LSI support chips in the light of present technology. These results establish the rationale and motivation for the investigation reported in the remaining Chapters.

2.1 Microprocessors

In 1972, the Intel Corp. introduced a 4-bit word length, 10 μsec. cycle-time, p-channel μP to the electronics industry.[7] Other firms soon followed suit with μP's of their own. This original device required three distinct power-supplies and a two-phased clock to execute its 45 instructions, and a complete microcomputer entailed about 4 chips. Soon, a second generation device entered the semiconductor market which exhibited nearly twice the performance of its predecessor.[8] By using the then recently developed n-channel technology,

which achieved higher density and increased speed, it displayed a 5 μsec. cycle-time, an 8-bit word length, and an instruction set of 80. Further advantages included single power-supply operation, and only a single-phase clock. And a mere 2 chips configured a complete microcomputer. Presently, the semiconductor market maintains a third generation μP which, again, doubles its forerunner's performance.[9, 10] This circuit uses advanced n-channel or integrated-injection logic (I^2L) technology, attains a 1 μsec. cycle-time for an 8 or 16-bit word length, and sells for about $10.00; it includes an instruction repertoire of a minicomputer (well over 100). Yet a more important addition involves on-chip memory and input/output (I/O) logic which significantly reduces package count and, thus, costs. Consequently, today's semiconductor market truly supports a "one-chip" microcomputer with many advanced features.

An analoguous trend which parallels these metal-oxide semiconductor (MOS) advances saw bipolar "bit-slice" μP's exploit the speed-power characteristics of low-power Schottky transistor-transistor logic (TTL). Unlike MOS μP's these devices do not contain a microprogram programmable logic array (PLA) and, hence, the user provides the microprogram;[11-13] this allows the use of any desired instruction set. With pipelined architectures they produce 125 nsec. cycle-times, and many small computer firms now offer bit-slice, μP-based minicomputers as either new products or upgraded replacements.[14]

From this discussion it becomes clear that the μP business is extremely competitive, which generally forces product lines to expand rapidly. Often, by the time a μP-based system reaches the marketing phase it contains obsolete parts and components.

2.2 Solid-State Memories

As with μP's and bit-slice devices, technological advances have remarkably altered both the price and performance of solid-state memories. Over the past several years average prices (millicents-per-bit) dropped at an annual rate of 35% and densities (bits-per-chip) increased 60% annually.[15, 16] For example, memories cost about $0.01 per bit and a typical chip contains from 1,000 to 8,000 bits. Consequently, memory system prices plunged while speed increased (e.g., 1 μsec. access times). Memories fall into two main divisions depending on the "accessibility" of an arbitrary location. Random access (or parallel) implies that any memory address may follow the other, but for sequential (or series) memories each address occurs in a specific order.[17, 18] For sequential memories, the CCD and bubble show much promise and may eventually replace the electromechanical serial memories such as disks, tapes, drums, etc. Although they now cost slightly more than the others, they contain no moving parts to wear out and exhibit faster access times.[19]

Three additional subdivisions further classify memory types: ROM/RAM, static/dynamic, and MOS/bipolar. First, ROM stands for read only memory and RAM for random access memory but, unfortunately, common terminology connotes RAM with read/write memory. So both ROM and RAM consist of random access memories where a μP can only read from a ROM, and can either read from, or write to, a RAM. Also, ROMs differ from RAMs in that they retain their contents during power-down (nonvolatile), whereas RAMs lose their contents (volatile). Additionally, ROMs occur in both mask and field programmable types.[20] For the first, before manufacturers "cook" the semiconductor they alter the circuit to contain the correct contents. With field programmable ROMs the end user applies

specific voltages and currents to the chip that define its contents. In general, this process purposely destroys part of the circuit, but ROMs with a new kind of FAMOS memory cell can be reprogrammed.[21] They contain a quartz window which when exposed to the correct ultra Violet (uV) light resets the memory for the next programming cycle.

Second, with static memories information remains in bistable flip-flops, but with dynamic memories information becomes the charge, or lack of charge, on a semiconductor capacitor and, hence, needs continued refresh due to nonideal leakage resistance. Although dynamic memories operate at higher speeds and consume less power than statics, still they may necessitate more costly auxiliary components such as multiple power supplies or critical clock circuits.[22] Third, the MOS/bipolar classification refers to the process technology used to manufacture the memory. With p-channel MOS memories their characteristics include low cost, excellent packing density, but slow speed; n-channel MOS properties basically differ in speed; i.e., slightly faster.[23] Another MOS technology, complementary-metal-oxide semiconductor (CMOS), employs both p and n-channel transistors and exhibits reduced power consumption with an increased speed-power product, but at the expense of less density and greater costs. As with all bipolar processes, Schottky TTL enjoys fast circuit operation and plays a dominant role in the digital industry.[24] But it dissipates considerable power and occupies much silicon area for decreased densities and high cost. Perhaps an excellent compromise technology, bipolar I^2L reveals good speed and density at less power dissipation and cost than Schottky TTL.[25] Therefore, many types of solid-state memories exist with widely differing characteristics, and to select one for an application involves careful consideration of several tradeoffs.

2.3 LSI Support Chips

The μP's described in Section 2.1 offer only general purpose instruction sets that often contain few sophisticated (or specialized) operations: particularly, I/O and arithmetic. To incorporate these auxiliary functions would require additional pins on the chip, and most manufacturers limit themselves to 40 (for economical yields) which eliminates such special operations (time-multiplexing the pins causes reduced speed and complex timing). Yet some firms concentrate on standard "support chips" that facilitate these external functions; e.g., data communication, data conversion, operator interaction, computation, etc.

Specifically, computational support chips deal with the arithmetic and trigonometric functions not found in μP's today. For example, new and creative algorithms[26, 27] helped semiconductor firms produce multiply circuits with 8-bit, 100 nsec., 1W operation that sell for about $100.00.[28, 29] And with an eye toward floating-point formats and operations, a 24-bit, 200 nsec., 5W multiply chip obviates the need for multiple-precision algorithms.[30] A remarkably versatile support chip contains both 16 and 32-bit, fixed and floating-point, arithmetic and trigonometric functions where all transfers occur over an 8-bit, bidirectional bus and in one 24-pin package.[31]

The ubiquitous calculator chip, when interfaced to a μP, provides extremely powerful processing power with literally no software development.[32, 33] Additionally, they incorporate both fixed and floating-point formats, but function slowly (typically 50 msec. for an add). Because they often require multiple power supplies, logic-level shifting circuits, and special encoding/decoding hardware, implementation costs involve around $200.00.[34]

The support chips for computations just described only illustrate the wave of new LSI function modules created by technology advances. Furthermore, many people predict the concept of LSI software where ROMs contain complete program packages tailored for some specific goal.[35, 36] Here, program ROMs matched with unique hardware arrangements drastically reduce software development time.

2.4 Summary

From the preceding survey of today's semiconductor market it becomes clear that many LSI devices facilitate computations. Representative characteristics of μP's, solid-state memories, arithmetic units, and calculator chips (see Table 2.1) illustrate the enhanced features and low cost of present devices. So with these characteristics a designer of μP-based systems may synthesize CDA's to perform advanced mathematical computations. Thus, this table of specific values defines the characteristics of LSI devices used throughout the remaining chapters.

Table 2.1 Representative characteristics of computational devices

a) μP's

8-bit word length

8 μsec. cycle-time

$10.00 cost

b) Solid-state memories

1-8k bits/chips

1 μsec. assess-time

$0.01/bit cost

c) Calculator devices

multiple-precision word-length

50 msec. execution-time

$200.00 cost

d) Arithmetic units

8-bit word length

100 nsec. execution-time

$100.00 cost

CHAPTER III COMPUTATIONAL DESIGN ALTERNATIVES

With the numerous LSI devices available to designers today, synthesis of advanced CDA's involves careful and tedious evaluation of several tradeoffs. This chapter defines a basic set of elemental CDA's which use the representative devices delineated in Chapter II. To facilitate comparison of each member in this set, it determines attributes common to each member that reflect the important features, qualities, and characteristics of each CDA, and it develops techniques to evaluate them. So synthesis of advanced CDA's now reduces to judicious combinations of these basic members.

3.1 Elemental CDA's

A generalized μP-based system may consist of several sections, each with specific and unique responsibilities (see Fig. 3.1). In such a system each section communicates to the others over a common system bus. The "computation" section functions to accomplish some distinct arithmetic calculation which the particular application requires. Because of technological progress, various combinations of hardware and software may implement the needed computation. Thus, a computational design alternative (CDA) represents: the specific combination of hardware and software required to accomplish a particular computation.

All "computational" sections of a μP-based system can be decomposed into four "elemental" CDA's, each with their own different

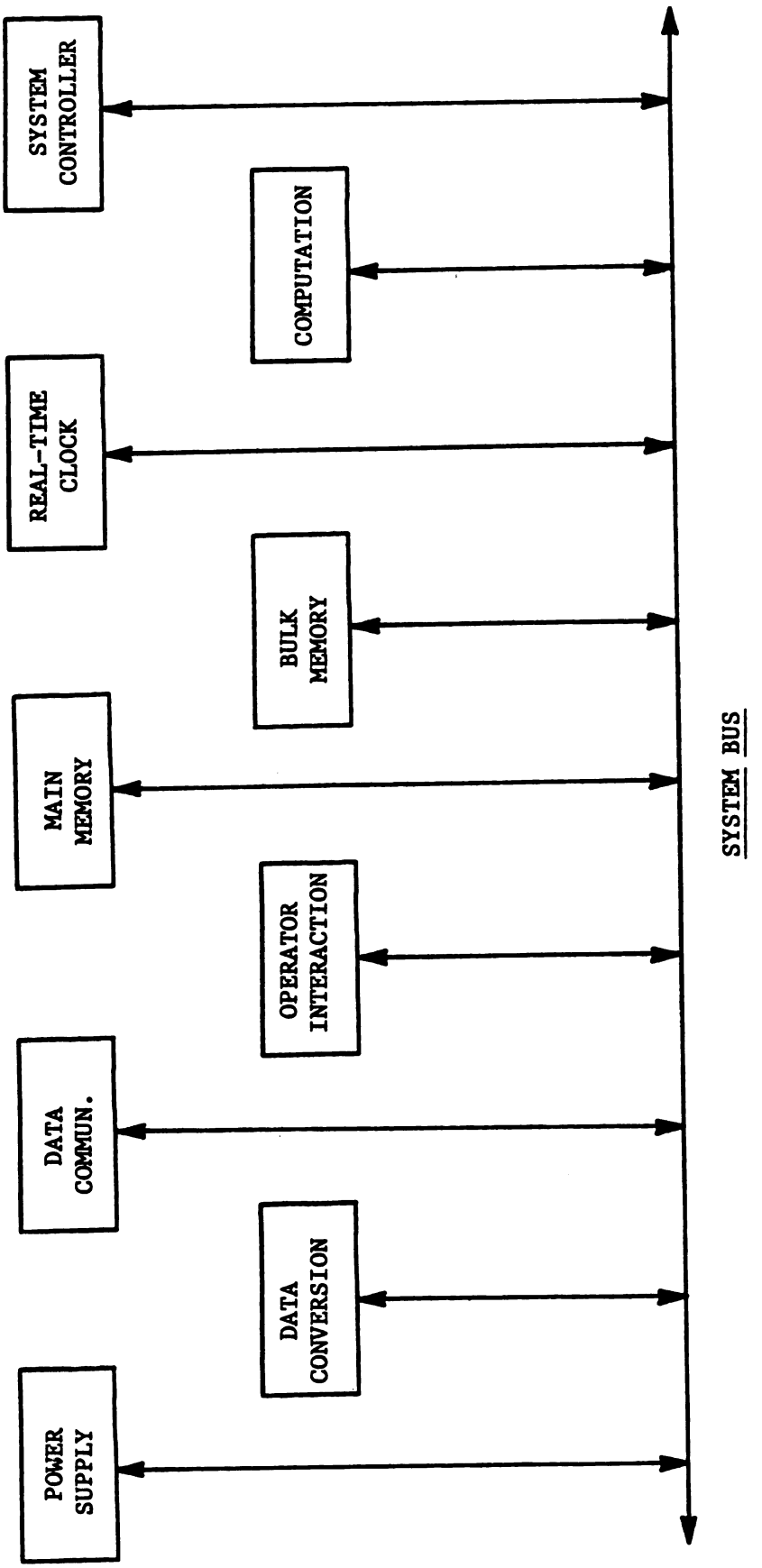


Figure 3.1 Block diagram of a generalized Microprocessor-Based System.

combination of hardware and software to accomplish computations. First, the DIRECT CDA consists of only a μP and memory connected via the system bus as shown in Figure 3.2. This CDA describes the simplest technique to execute arithmetic operations. Since the memory contains arithmetic (add, subtract, multiply, and divide) subroutines, the DIRECT CDA performs all computations in software. Next, the second CDA employs a μP, memory, and an arithmetic unit (AU) (see Fig. 3.3), all joined by the system bus. The AU CDA differs from the DIRECT CDA in how it achieves multiplies and divides. For the AU CDA it writes the two operands (numbers) into AU buffers, starts the multiply or divide operation, and simply reads back the result. But the AU CDA, like the DIRECT CDA, performs both add and subtract instructions totally in software. The third CDA incorporates a calculator chip to accomplish arithmetic computations as depicted in Figure 3.4. Here, the CALC CDA consists of a μP, memory, and the calculator chip, all linked by the system bus. Within the memory its program must simulate depressing the keys as with a hand held calculator; it must also read back the results and decode them when the requested function finishes.

The final elemental CDA applies the concept of simultaneous, or parallel, execution: it executes sections of the problem simultaneously, then adds the partial results together for the completed answer.[37-40] Because of this concept the multiple-microprocessor (mμP) CDA involves several μP's and memories in a Master-Slave arrangement as illustrated in Figure 3.5. Each Slave memory contains a replica of the DIRECT CDA memory, arithmetic subroutines, and performs all computations in software (like the DIRECT CDA). Yet the Master's memory includes no arithmetic subroutines; instead, it holds an "overhead" program

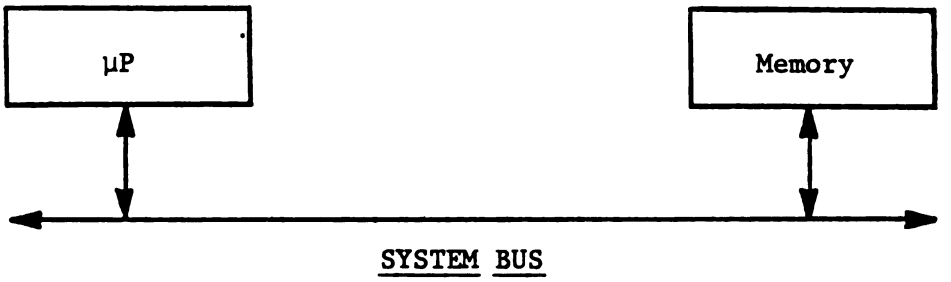


Figure 3.2 Block diagram of the DIRECT computational design alternative.

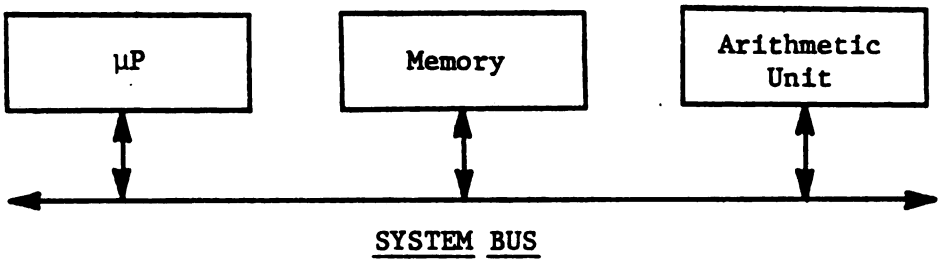


Figure 3.3 Block diagram of the AU computational design alternative.

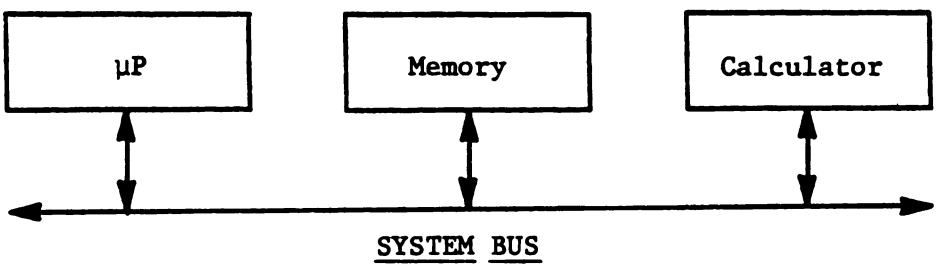


Figure 3.4 Block diagram of CALC computational design alternative.

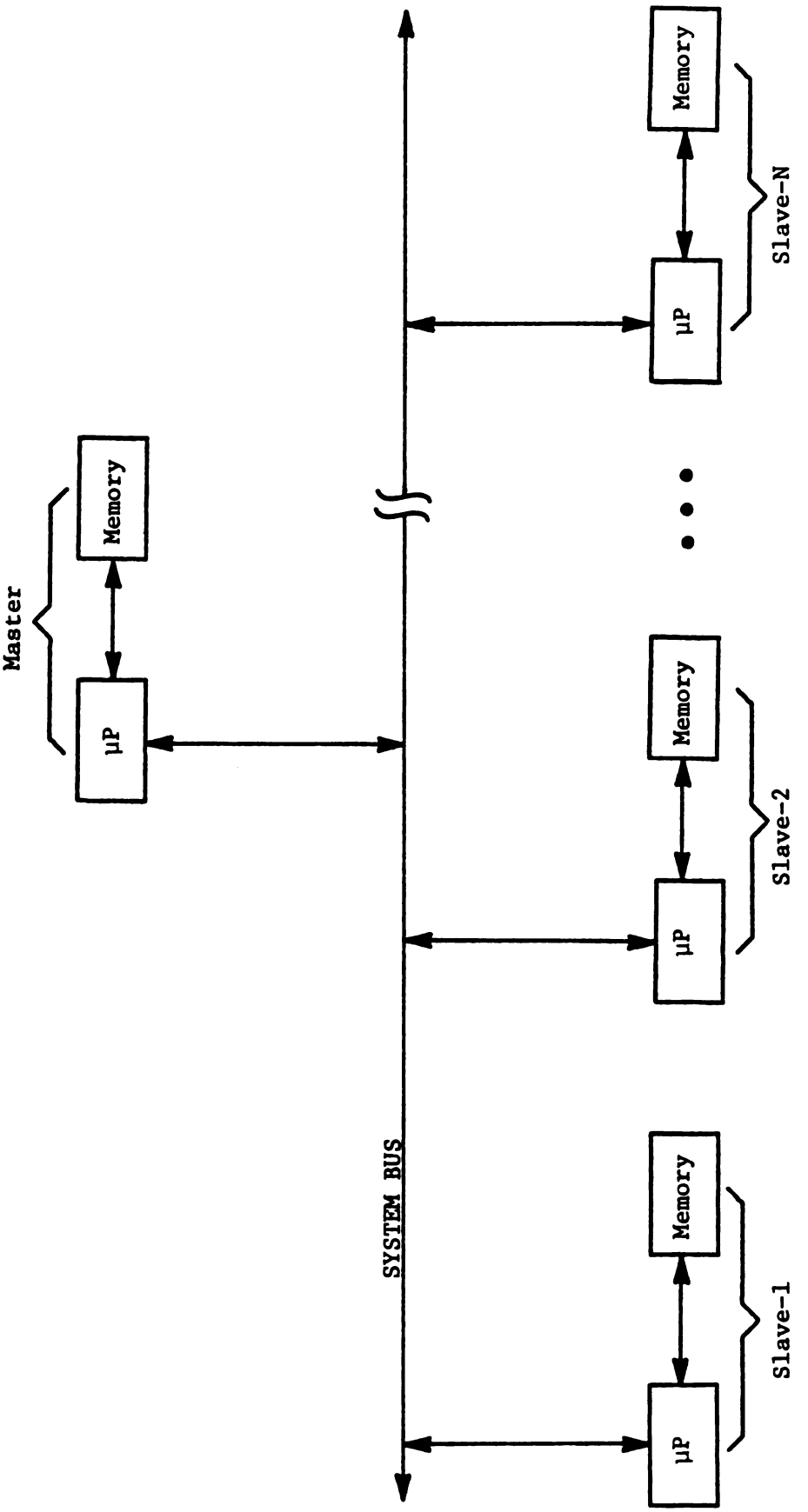


Figure 3.5 Block diagram of the Multiple Microprocessor (mµP) computational design alternative.

responsible for routing data and preliminary results from Slave to Slave, and for forming the completed result.

The above mμP CDA Master/Slave arrangement does not necessarily produce optimal results; an architecture designed for a unique problem, or class of problems, can potentially decrease execution-time and reduce costs. But the proposed mμP CDA does present a simple, general-purpose architecture that solves many problems well and, additionally, lends itself to straightforward analysis.

For each elemental CDA a unique mixture of hardware and software accomplish computations. Thus, the properties of each CDA vary considerably. So judicious combinations of the four CDA's (DIRECT, AU, CALC, and mμP) will realize the computation section of a μP-based system.

3.2 Attributes

Comparison of a group of objects occurs through evaluation of a common subset of characteristics, or properties. Moreover, the selected subset of attributes must reflect the important features and qualities of each member. When comparing a set of CDA's, several attributes satisfy these requirements: precision, execution-time, cost, power dissipation, circuit complexity, programming language, circuit reliability, packaging demands, maintenance schedule, etc. Too few attributes results in incomplete examination of the objects while too many attributes may cause unnecessary confusion. For the above list, all but the first three attributes depend strongly on the remainder of the μP-based system and, thus, fail to depict attributes indigenous to the CDA's. But precision, execution-time, and cost represent convenient and illustrative attributes for demonstrating the principal characteristics of CDA's.

3.3 Precision

Two common number representations, fixed-point and floating-point, both identify discrete values on the real-number line. For the fixed-point format an implied binary-point always lies between two specific bits in the word, and with floating-point numbers the binary-point varies (this requires storage of the position). In the research reported here, all numbers conform to the fixed-point format. Thus, precision involves the quantity of 8-bit memory words used to form a particular number; e.g., p = 1 for single-precision, p = 2 for double precision, etc. Since the μP's perform two's complement arithmetic, all numbers correspond to multiple-precision, fixed-point, two's complement quantities.

3.4 Execution-Times

Execution-time determines the number of seconds needed to complete a specific computation. Since addition/subtraction and multiplication/division both correspond to complementary arithmetic and digital logic functions, the execution-time of an addition roughly equals a subtraction and a multiplication roughly equals a division. These assumptions greatly facilitate analysis of execution-times. Thus, if the number of adds and multiplies can approximate the execution-time of an application program, then these values multiplied by the time to perform an add and multiply sum to give the execution-time. Symbolically, let

A_i = number of adds

M_i = number of multiplies

a_i = add time

m_i = multiply time

$$T_i = A_i a_i + M_i m_i = \text{execution-time} \qquad (3.1)$$

where,

$$i = \begin{cases} 1, \text{ DIRECT CDA} \\ 2, \text{ AU CDA} \\ 3, \text{ CALC CDA, and} \\ 4, \text{ m}\mu\text{P CDA.} \end{cases}$$

In Equation 3.1 the specific CDA architecture dictates expressions for a_i and m_i, while the algorithm of the application program configures constants A_i and M_i.

Multiple-precision arithmetic operates on each word of the number individually to produce the final result; e.g., consider the multiple-precision add flowchart in Figure 3.6. Here, corresponding words of the two operands add to produce the partial sum and the carry ripples through from word to word. Since each pass through the loop involves about 10 instructions, then for

t_o = cycle-time (8 μsec.)

p = precision, and

$$a_1 = 10pt_o. \tag{3.2}$$

Also, the AU and mμP CDA's employ identical add algorithms, so

$$a_2 = 10pt_o, \text{ and} \tag{3.3}$$

$$a_4 = 10pt_o. \tag{3.4}$$

But with the CALC CDA each operand requires decoding and encoding from binary to BCD formats, and each arithmetic operation demands "digit entry" (simulation of key depresses) and "function" time (see Fig. 3.7 and 3.8). Decoding/encoding tasks and the answer reads occur in μsecs., but when compared to the digit entry and function delays (msecs.) they contribute no significant time to the add, or multiply, execution-times. To determine the approximate number of digits sent to the calculator chip each two's complement number roughly ranges in magnitude $\pm 2^{8p-1}$,

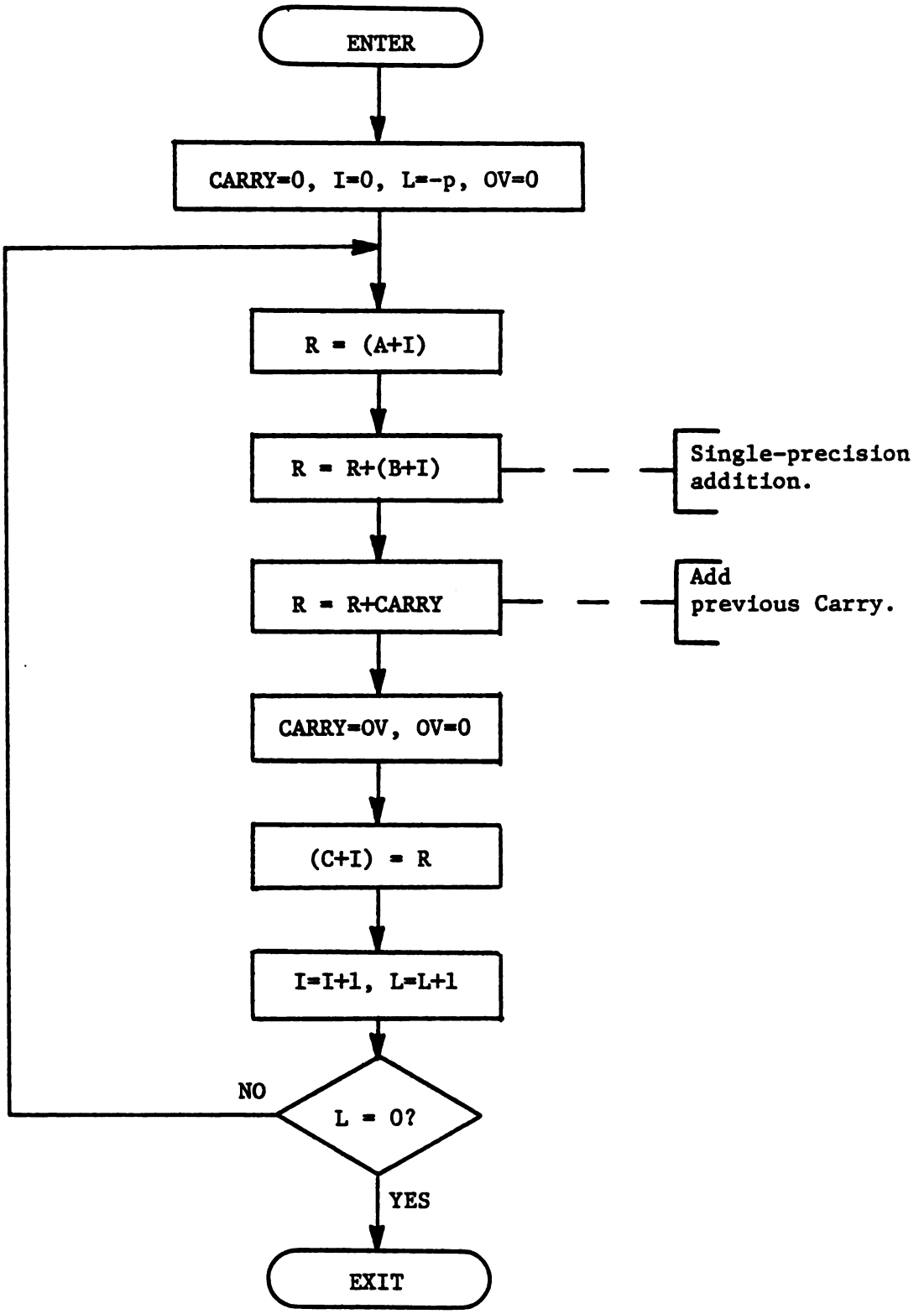


Figure 3.6 Multiple-precision addition flowchart.

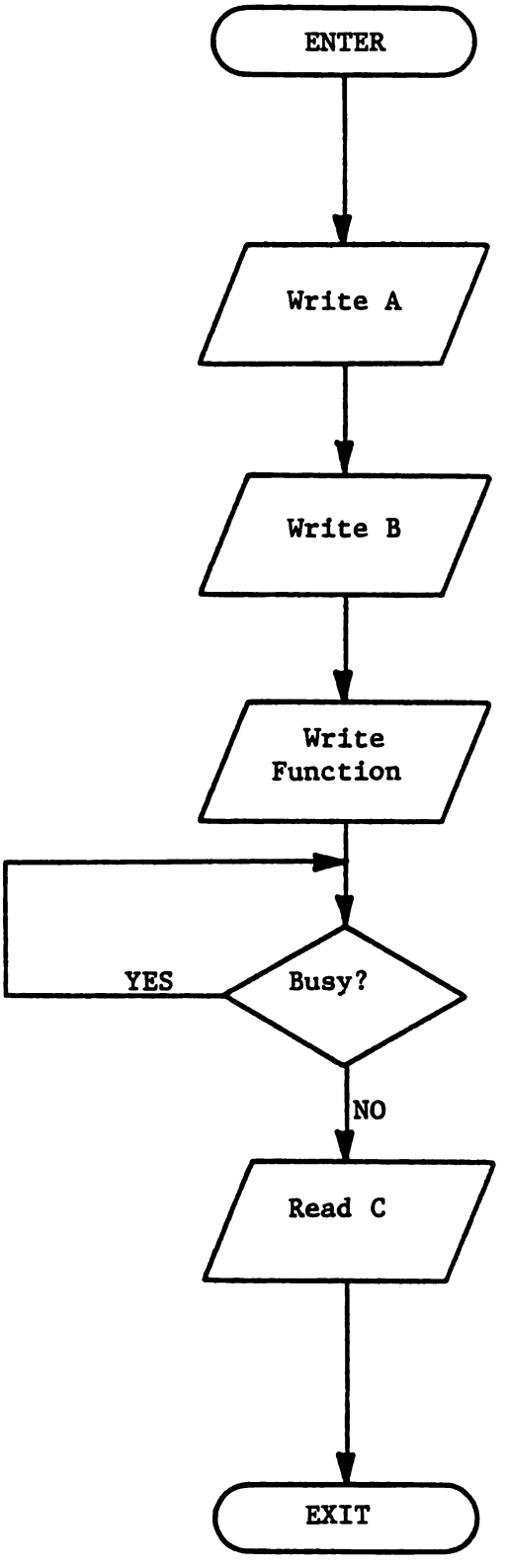


Figure 3.7 "Operation" flowchart for CALC CDA.

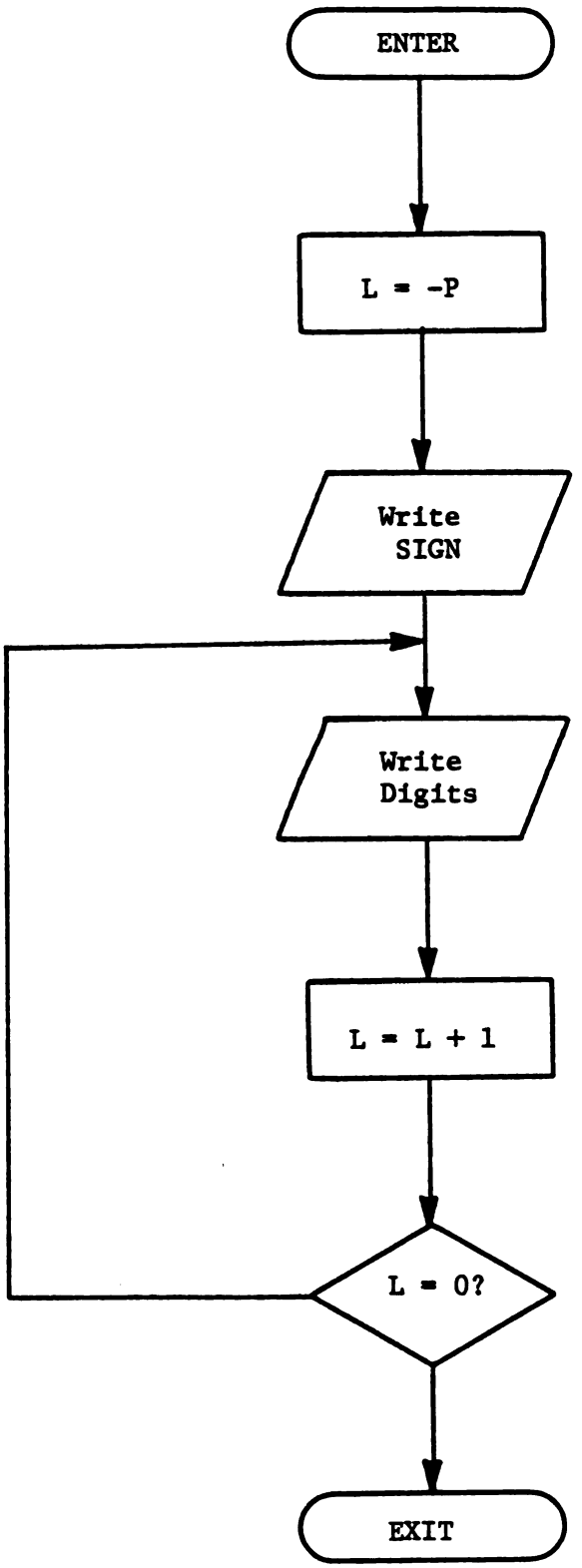


Figure 3.8 "Number-entry" flowchart for CALC CDA.

and on the average the number decoded falls in the middle, 2^{8p-2}. For $1 \leq p \leq 4$ these numbers contain around 2p decimal digits; e.g., if p = 1 and $2^{8p-2} = 2^6 = 64$, then the μP sends 2 decimal digits to the calculator chip. With these ideas the execution-time of a multiple-precision add instruction for the CALC CDA becomes

t_e = digit entry time (40 msecs.)

t_a = "add" function time (90 msec.), and

$$a_3 = 2(2p + 1)t_e + t_a. \tag{3.5}$$

Equations 3.2 through 3.5 yield estimates for the multiple-precision add times of all CDA's and Figure 3.9 illustrates these values for various precisions. In this figure the most noticable observation concerns the difference in add-time "magnitudes": nearly an order of 3.

By reducing the mathematical operation of multiplication to repetitive adds, the product becomes the multiplicand added to itself the multiplier number of times. For a p-precision, two's complement, fixed-point binary number multiplied by another, the result yields a 2p-precision product. Thus, the DIRECT CDA (without a hardware multiply) can use the algorithm depicted in Figure 3.10. The 2p-precision adds contribute, using Equation 3.2, $20pt_o$ time and on the average (see the development of Eq. 3.5) this occurs 2^{8p-2} times each multiplication; i.e., the multiple-precision multiply

$$m_1 = (20pt_o)(2^{8p-2})$$

$$= 80p2^{4p-1}t_o. \tag{3.6}$$

And since the mμP CDA uses the same software,

$$m_4 = 80p2^{4p-1}t_o. \tag{3.7}$$

With the AU CDA the problem changes dramatically due to its 8-bit by

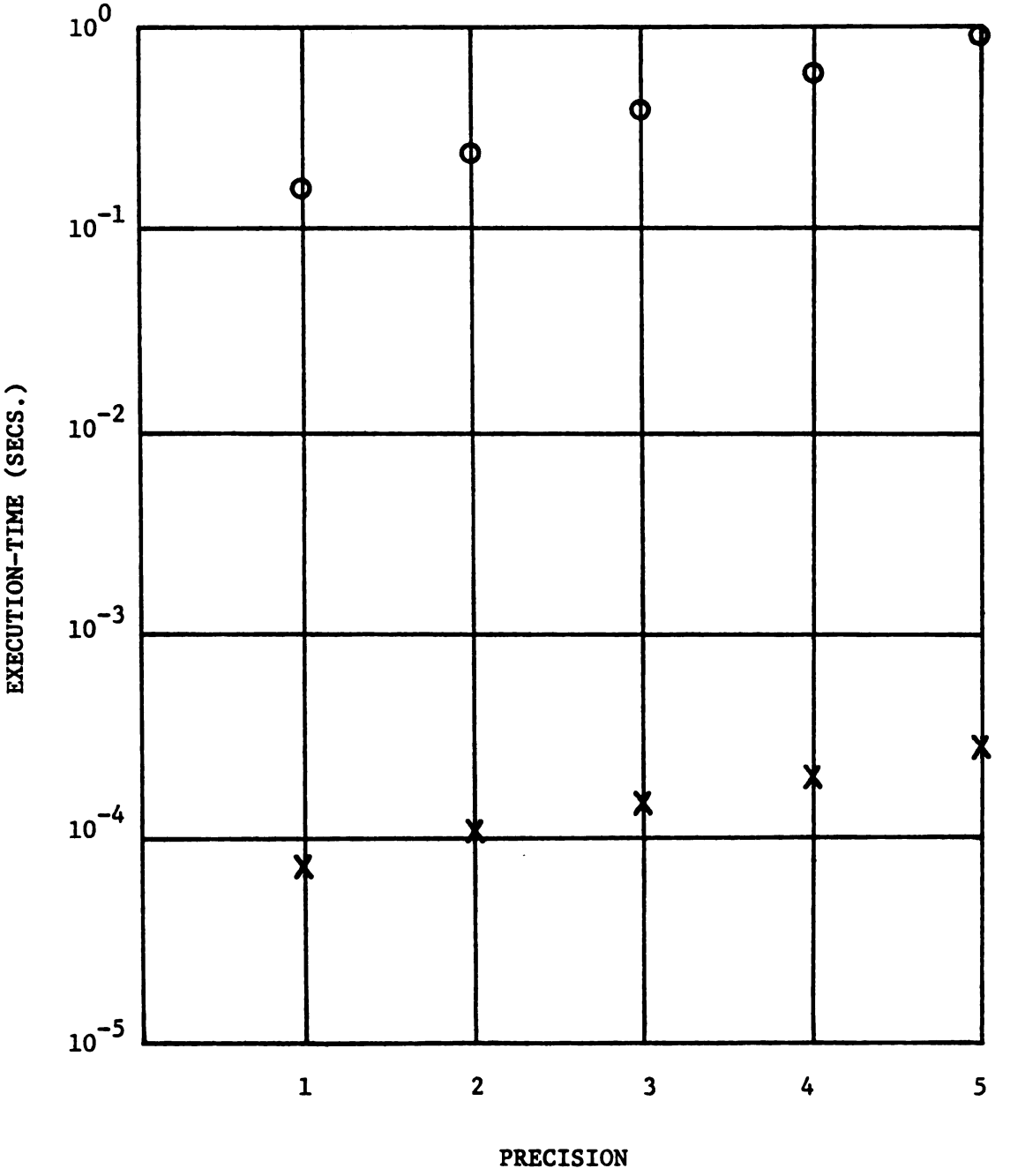


Figure 3.9 Multiple-precision addition time for various CDA's.

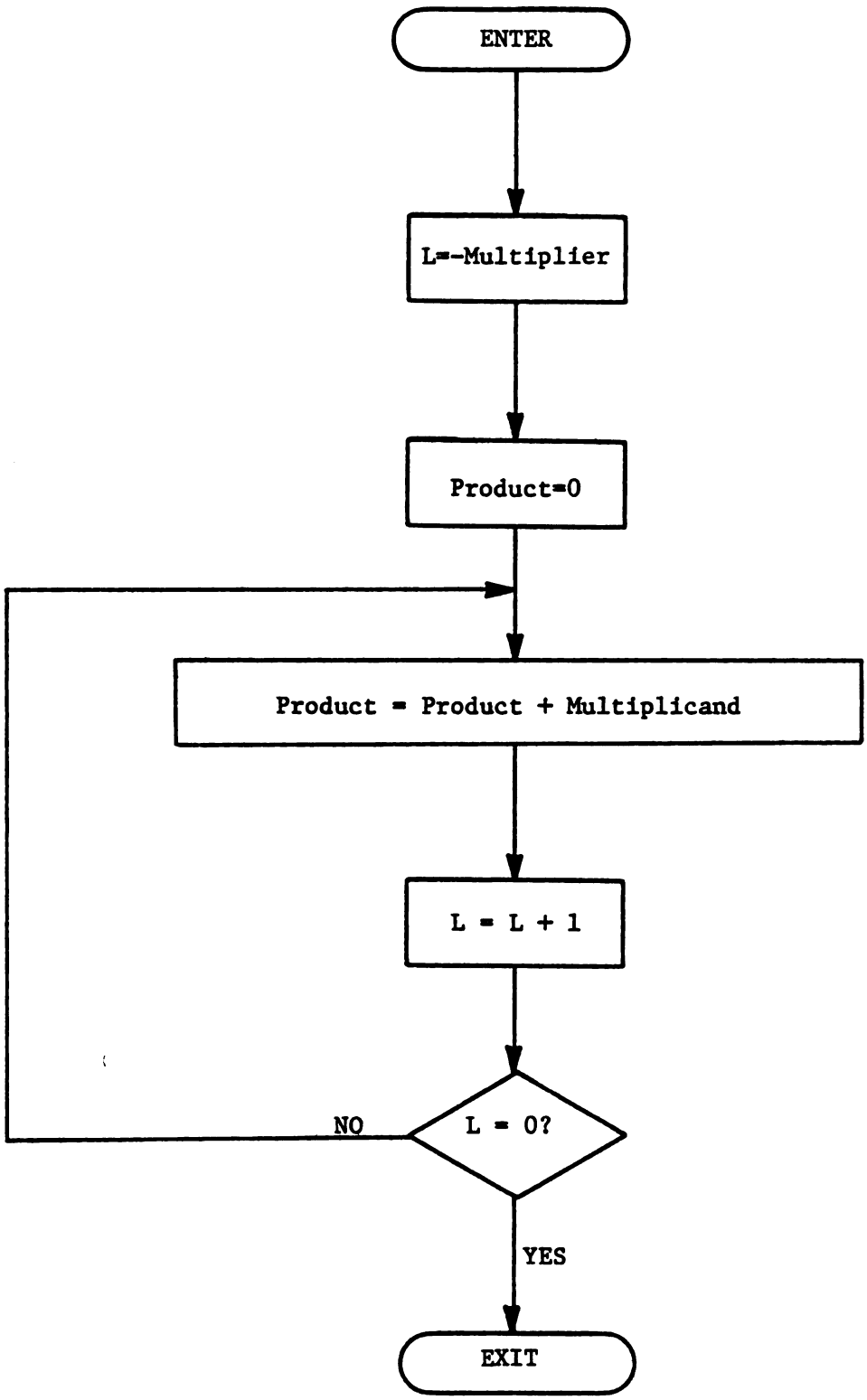


Figure 3.10 Multiple-precision multiply flowchart for DIRECT CDA.

8-bit hardware multiply. If the μP needs two I/O instructions to write the parameters into the AU, one to wait for the multiply to finish, and two to read the 16-bit product, then one single-precision multiply requires ($5t_o$) time. But for multiple-precision numbers the task becomes much more difficult; consider the notation:

$$A = (a_p + a_{p-1} + \ldots + a_1) = \text{multiplier}$$

$$B = (b_p + b_{p-1} + \ldots + b_1) = \text{multiplicand, and}$$

$$C = AB$$

$$= (a_p b_p + a_p b_{p-1} + \ldots + a_p b_1)$$

$$+ (a_{p-1} b_p + a_{p-1} b_{p-1} + \ldots + a_{p-1} b_1) + \ldots$$

$$+ (a_1 b_p + a_1 b_{p-1} + \ldots + a_1 b_1),$$

$$= \text{product.}$$

Thus, multiple-precision multiplies for AU CDA involve p^2 single-precision multiplies (partial products) and p^2 2p-precision adds (see Fig. 3.11); combining,

$$m_2 = p^2(5t_o) + p^2(20pt_o)$$

$$= 5p^2(1 + 4p)t_o. \qquad (3.8)$$

For the CALC CDA the analysis of Equation 3.5 directly applies to finding the multiple-precision multiply time; only the "multiply" function replaces the "add" function time; i.e.,

t_e = digit entry time (40 msec.)

t_m = "multiply" function time (120 msec.), and

$$m_3 = 2(2p + 1)t_e + t_m. \qquad (3.9)$$

So Equations 3.6 through 3.9 give estimates for the multiple-precision multiply times of all CDA's, and Figure 3.12 illustrates these values for various precisions. From this figure the most striking feature pertains to the rapid rise of the DIRECT multiply time;

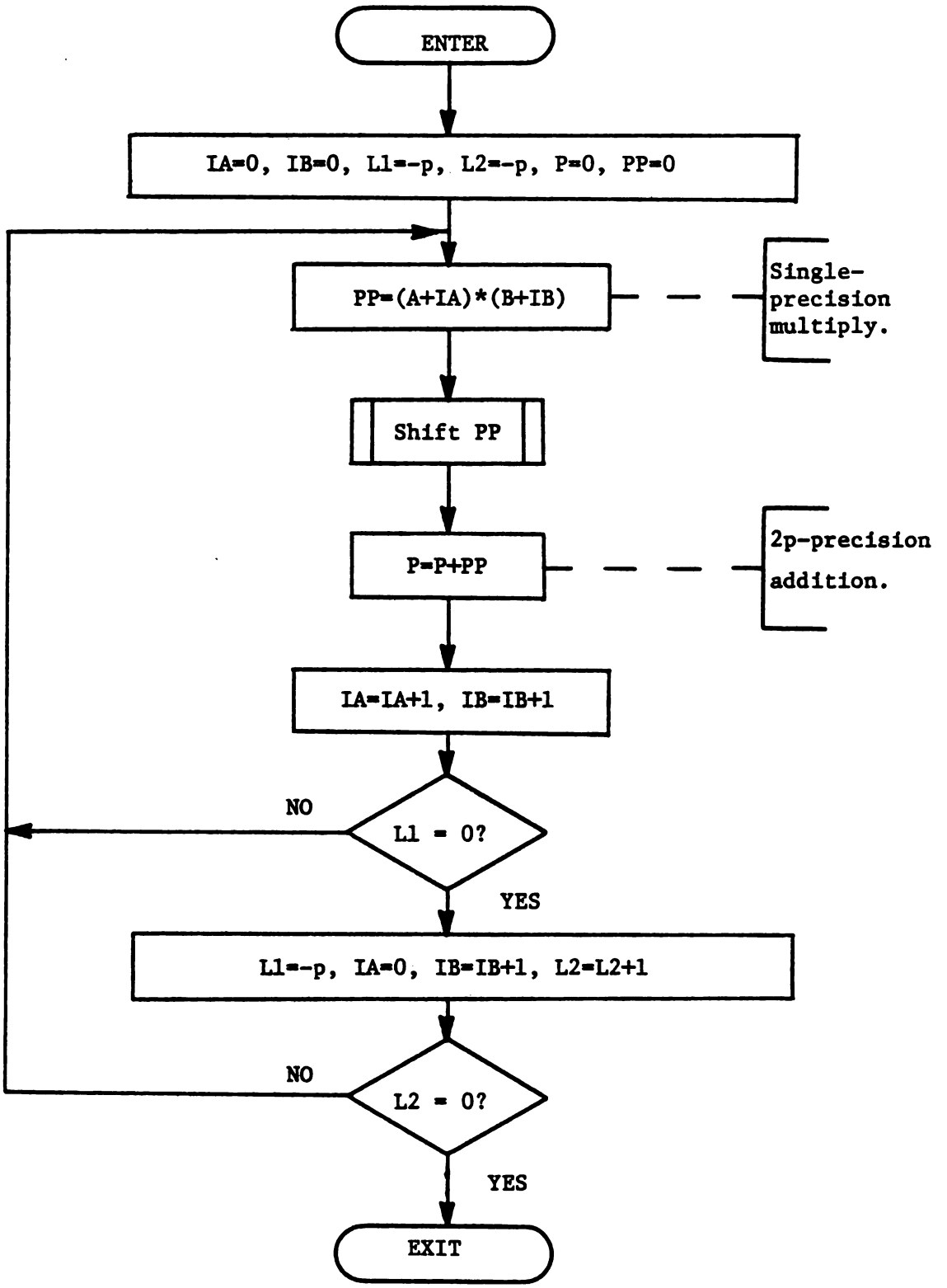


Figure 3.11 Multiple-precision multiply flowchart for AU CDA.

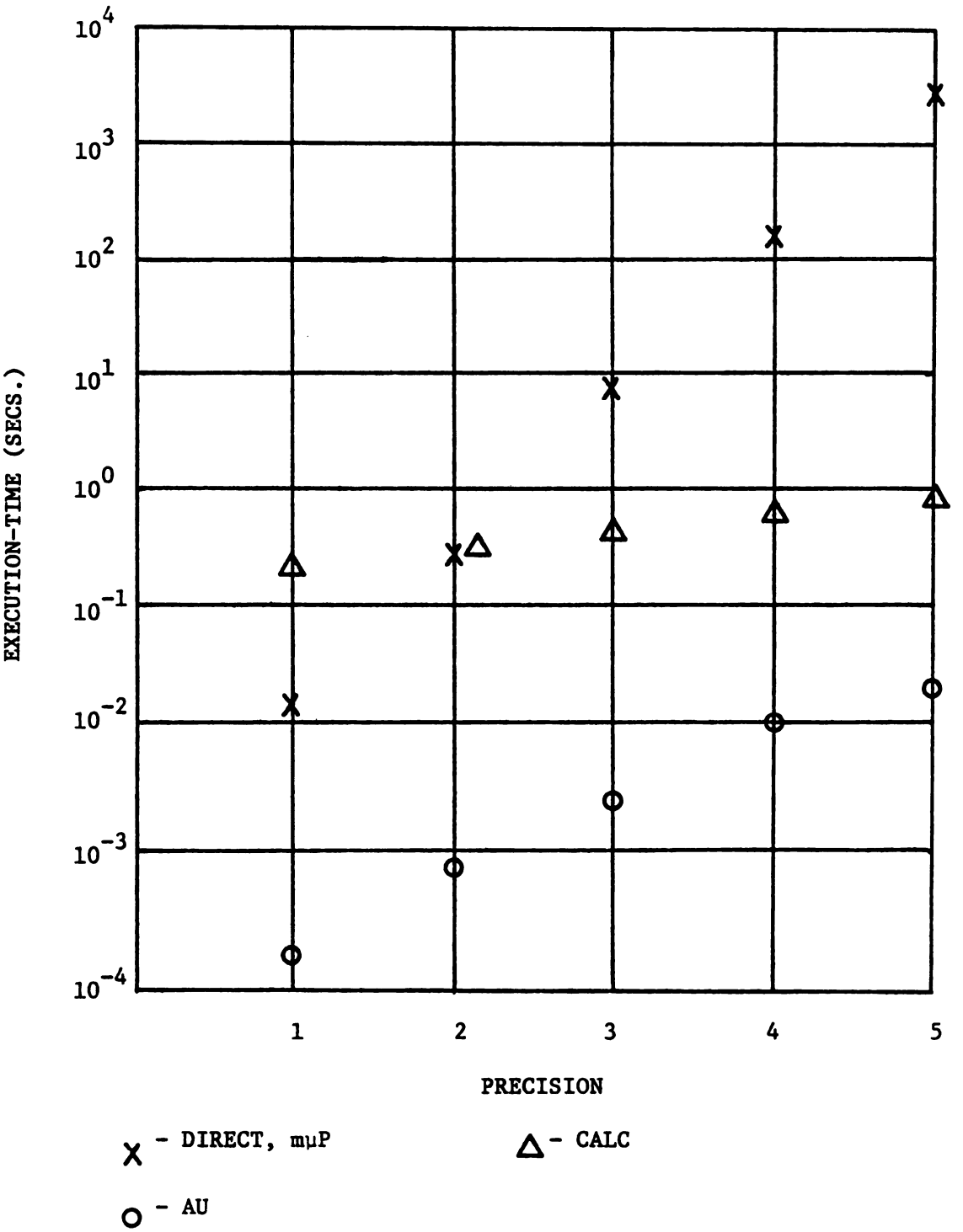


Figure 3.12 Multiple-precision multiply time for various CDA's.

for p = 1 it lies below the CALC, by p = 2 it equals the CALC, and for $p \geq 3$ it exceeds the CALC considerably. Such action results from the repetitive adds which comprise the DIRECT multiply time. Also, the AU multiply time remains the fastest because of its hardware multiply, but it increases quicker than the CALC. This event occurs due to the larger number of AU multiplies required with greater precision, than the increment in digit entries for the CALC CDA. Now, once the constants A_i and M_i have been found for an application, Equation 3.1 with Table 3.1 can give an estimate of the execution-time for each CDA.

In this section the flowcharts for various multiple-precision add and multiply instructions provide only the basic outline. Among the functions added to an implemented algorithm include initialization, zero and sign checks, overflow/underflow detection, etc. But these extra duties do not contribute significantly to the overall execution-time, nor do they represent more than second and third-order terms in the execution-time Equations 3.2 to 3.9.

3.5 Costs

The monetary expense incurred with each CDA could include several components, many of which depend on the remainder of the μP-based system, so this development focuses on the semiconductor parts cost. For all CDA's two principal terms add to give the total cost, one corresponds to an elemental (or basic) CDA cost and the other to an application dependent cost:

$$\begin{aligned} C_{ei} &= \text{elemental CDA cost} \\ C_{ai} &= \text{application cost, and} \\ C_i &= C_{ei} + C_{ai};\ 1 \leq i \leq 4; \\ &= \text{total cost (in \$'s).} \end{aligned} \tag{3.10}$$

First, the elemental cost term C_{ei} relates to the basic expense of procuring a CDA and its mathematical subroutine's memory. And three main factors comprise C_{ei},

IO_i = I/O device cost

R_i = mathematical subroutine's cost

CPU_i = μP cost, and

$$C_{ei} = IO_i + R_i + CPU_i;\ 1 \leq i \leq 4. \tag{3.11}$$

Second, the application cost term C_{ai} pertains only to the added expense of the application memory. This involves two terms, one for program memory and one for data storage:

w_o = cost per word ($0.08)

P_i = program memory

D_i = data memory, and

$$C_{ai} = w_o(P_i + pD_i);\ 1 \leq i \leq 4. \tag{3.12}$$

Because term C_{ei} in Equation 3.10 does not depend on the application it can be determined now, yet term C_{ai} must be deferred until an application program is defined. Hence, the three terms IO_i, R_i, and CPU_i in Equation 3.11 need evaluation for each CDA.

For the DIRECT CDA the I/O devices and μP costs involve simple estimates; since no I/O devices reside on the bus IO_1 = $0.00 and, today, an estimate for CPU_1 = $10.00. But estimating R_1 involves much more analysis and approximation. From Figure 3.6, the multiple-precision add flowchart, if each step involves about 1 word of memory, then 30 words corresponds to a reasonable estimate of the number of words for the instruction. Similarly, the multiply instruction illustrated in Figure 3.10 uses the add instruction so the number of words roughly equals 20. A summation of these values yields 50 words and doubling this to account

for the subtract and divide subroutines results in 100 words. So 100 words at a cost of w_o = \$0.08 per word gives $R_1 = 100w_o$ = \$8.00.

With the AU CDA the expense of the AU changes IO_2; today, a reasonable estimate of IO_2 = \$100.00. Still, CPU_2 remains the same as CPU_1, or CPU_2 = \$10.00. Following the procedure which determined R_1, the add instruction contributes 30 words while the AU multiply instruction (see Fig. 3.11) attaches an addition 45 words. Together, they sum to 75 words which when doubled for the subtract and divide subroutines give 150 words; this sets $R_2 = 150w_o$ = \$12.00.

Next, the CALC CDA, like the AU CDA, embodies an expensive I/O device (the calculator chip) and a contemporary estimate for this term assigns IO_3 = \$200.00. Yet the CPU estimate continues at CPU_3 = \$10.00. With this CDA the exact same flowchart, see Figure 3.7 and 3.8, can accomplish all arithmetic subroutines because only the function code (a parameter) need change from instruction to instruction; e.g., from add to multiply. These figures suggest that about 50 words can retain the program, and $R_3 = 50w_o$ = \$4.00.

Analysis of the last CDA, the mμP alternative, for the C_{e4} term parallels the DIRECT CDA except in quantity. Since the mμP CDA replicates the DIRECT CDA hardware and software, and if it engages S Slaves, then IO_4 = \$0.00, $R_4 = SR_1$, and $CPU_4 = (S + 1)\ CPU_1$. The CPU_4 term multiplies CPU_1 by (S + 1) because the Master μP creates an additional term.

Thus, Table 3.2 delineates the estimates for IO_i, R_i, CPU_i, and C_{ei} for all CDA's, $1 \leq i \leq 4$, based on 1978 device costs. From this table the elemental cost of the mμP CDA grows rapidly as the number of Slaves increases. It surpasses the AU CDA by S = 6 and the CALC CDA by S = 12. Also, the contribution of R_i to C_{ei} for the AU and CALC CDA's is insignificant because of the expensive IO_i term.

3.6 Summary

The preceding sections defined a basic set of four elemental CDA's: DIRECT, AU, CALC, and mμP. In addition, they identified three important attributes--precision, execution-time, and cost--which facilitate comparison of these elemental CDA's. Precision involves the quantity of 8-bit words used to represent a number, execution-time the number of seconds to complete a computation, and cost the dollar value of LSI devices used. Equations 3.1 and Table 3.1 determine execution-time, while Equation 3.10 and Table 3.2 specify cost (both use precision as a parameter). For execution-time and cost, these values result from terms indigenous to each CDA and from application derived terms. Using elementary multiple-precision flowcharts for arithmetic operations, the chapter estimates these indigenous terms. Such an approach offers straightforward and tractable analysis. In Chapter V, the text presents techniques to estimate these application terms, and illustrates the effect of alternative algorithms on the DIRECT and mμP CDA's performance; but first, Chapter IV presents a decision mechanism for selecting the "best" CDA.

Table 3.1 Add and Multiply execution-times for various CDA's.

i	a_i	m_i
1	$10pt_o$	$80p2^{4p-1}t_o$
2	$10pt_o$	$5p^2(1+4p)t_o$
3	$2(2p+1)t_e + t_a$	$2(2p+1)t_e + t_m$
4	$10pt_o$	$80p2^{4p-1}t_o$

Table 3.2 Estimates of C_{ei} for various CDA's.

i	IO_i	R_i	CPU_i	C_{ei}
1	--	\$8.00	\$10.00	\$18.00
2	\$100.00	\$12.00	\$10.00	\$122.00
3	\$200.00	\$4.00	\$10.00	\$214.00
4	--	\$(S)9.00	\$(S+1)10.00	\$(S)19.00 + 10.00

CHAPTER IV MULTIATTRIBUTE UTILITY THEORY

Using the attributes--precision, execution-time, and cost--this chapter describes a decision mechanism for selecting the "best" CDA entitled Multiattribute Utility Theory (MUT). This begins with several definitions that explain the notation used throughout the remaining sections. Next, the chapter presents eight axioms upon which the theoretic foundation of MUT rests, and it shows that the above attributes satisfy these axioms. In addition, it verifies that the attributes also fulfill the assumptions of a specific decision function; i.e., an additive utility function. Finally, this chapter describes techniques to assess this specific decision function. Thus, these concepts provide a decision mechanism for selecting the "best" CDA using attributes precision, execution-time, and cost.

4.1 Multiattribute Utility Theory Definitions

As a guide to notation several definitions are explained that will be used in the sections that follow.[41]

Definition 4.1: Consequence Space

Let consequence space, denoted $\bar{X} = \bar{X}_1 \times \bar{X}_2 \times \bar{X}_3 \times \ldots \times \bar{X}_N$, represent a rectangular subset of a finite N-dimensional Euclidean space.

Definition 4.2: Consequence

Let the consequence $\bar{x}$, denoted $\bar{x} = (x_1, x_2, x_3, \ldots, x_N)$, depict a specific point in consequence space where

x_i belongs to $\bar{X}_i$.

Definition 4.3: Attribute

Let the attribute x_i correspond to a particular value of dimension $\bar{X}_i$; e.g., x_i belongs to $\bar{X}_i$.

Definition 4.4: Consequence Set

Let the consequence set C, denoted $C = \{\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_M\}$, consist of the set of M consequences.

Definition 4.5: Relation

For consequences $\bar{x}_i$ and $\bar{x}_j$, the relation $\bar{x}_i > \bar{x}_j$ means that $\bar{x}_i$ is preferred to $\bar{x}_j$. Similarly, $\bar{x}_i < \bar{x}_j$ means $\bar{x}_i$ is not preferred to $\bar{x}_j$, and $\bar{x}_i = \bar{x}_j$ means $\bar{x}_i$ is equally preferred to $\bar{x}_j$.

Definition 4.6: Operation

For mutually exclusive consequences $\bar{x}_i$ and $\bar{x}_j$ and probability α, $0 \leq \alpha \leq 1$, the operation $w = \alpha\bar{x}_i + (1 - \alpha)\bar{x}_j$ represents the consequence $\bar{x}_i$ with probability α and consequence $\bar{x}_j$ with probability $(1 - \alpha)$.

Definition 4.7: Utility

Let the utility of consequence $\bar{x}$, denoted $u(\bar{x})$, describe a scalar quantity which indicates the usefulness of consequence $\bar{x}$.

Definition 4.8: Marginal Utility

Let the marginal utility of attribute x_i, denoted $u_i(x_i)$, designate a scalar quantity which indicates the usefulness of attribute x_i.

4.2 Utility Existence Axioms

The theoretic structure of MUT rests on a foundation composed of eight axioms. Von Neumann and Morgenstern[42] have shown that if a consequence set satisfies these axioms, then there exists a numerical utility for each member of the set. Formally, let $\bar{x}_i$, $\bar{x}_j$, and $\bar{x}_k$ denote any three members of the consequence set, and for $0 \leq \alpha,\beta \leq 1$:

Axiom 4.1

One and only one of the following three relations must hold; $\bar{x}_i > \bar{x}_j$, $\bar{x}_i < \bar{x}_j$, or $\bar{x}_i = \bar{x}_j$.

Axiom 4.2

If $\bar{x}_i > \bar{x}_j$ and $\bar{x}_j > \bar{x}_k$, then $\bar{x}_i > \bar{x}_k$.

Axiom 4.3

If $\bar{x}_i > \bar{x}_j$, then $\bar{x}_i > \alpha\bar{x}_i + (1 - \alpha)\bar{x}_j$.

Axiom 4.4

If $\bar{x}_i < \bar{x}_j$, then $\bar{x}_i < \alpha\bar{x}_i + (1 - \alpha)\bar{x}_j$.

Axiom 4.5

If $\bar{x}_i > \bar{x}_j > \bar{x}_k$, then there exists an α such that $\alpha\bar{x}_i + (1 - \alpha)\bar{x}_k > \bar{x}_j$.

Axiom 4.6

If $\bar{x}_i < \bar{x}_j < \bar{x}_k$, then there exists an α such that $\alpha\bar{x}_i + (1 - \alpha)\bar{x}_k < \bar{x}_j$.

Axiom 4.7

The operation $\alpha\bar{x}_i + (1 - \alpha)\bar{x}_j = (1 - \alpha)\bar{x}_j + \alpha\bar{x}_i$.

Axiom 4.8

The operation $\alpha(\beta\bar{x}_i + (1 - \beta)\bar{x}_j) + (1 - \alpha)\bar{x}_j = \alpha\beta\bar{x}_i + (1 - \alpha\beta)\bar{x}_j$.

Theorem 4.1

If a consequence set satisfies the preceding eight axioms, then a numeric utility exists for each member of the set.

When MUT is applied to selecting CDA's the consequence set contains exactly four members (one for each CDA) with three common attributes (precision, execution-time, and cost). Thus, M = 4 and N = 3. Axiom 4.1 deals with the "completeness" of the consequence set. This assumption rules out the possibility that; i) neither of two CDA's is preferred, while ii) both CDA's are undesirable. In Axiom 4.2 the assertion considers the "transitivity" of preferences; e.g., if one CDA is preferred to a second which is preferred to a third, then the first CDA is preferred to the third. This property is quite plausible and commonly accepted. For Axiom 4.3 if CDA one is preferred to two, then it will always be preferred to the combined event because CDA two can occur with probability $(1 - \alpha)$ (Axiom 4.4 states the dual of Axiom 4.3). Now, if CDA one is preferred to two and two is preferred to CDA three as in Axiom 4.5, then for some α (sufficiently large) the combined event is preferred to CDA two. Here, α provides a likely base for the numerical estimate of the preference for CDA one to two, over CDA two to three (Axiom 4.6 states the dual of Axiom 4.5). These Axioms, 4.5 and 4.6, provide credible "continuity" assumptions. In Axiom 4.7 the statement claims that the order of combined events may vary; this follows, accordingly, since the constituents result from alternate events. Finally, Axiom 4.8 states that combination events composed of CDA constituents may proceed in two successive steps or one complete operation. Such a "distributive" property is usually accepted as standard practice.

4.3 Additive Utility Functions

Axioms 4.1 through 4.8 guarantee that utilities exist for all members of the consequence set C, still the exact functional form remains unknown. In general, a utility function involves multidimensional operations on the attribute values, but under certain conditions this function reduces to a much simpler form composed of many unidimensional functions; symbolically,

$$u(\bar{x}) = f\{u_1(\bar{x}), u_2(\bar{x}), u_3(\bar{x}), \ldots, u_N(\bar{x})\}. \tag{4.1}$$

Consider the following definition:

Definition 4.9: Utility Independence

Let $\bar{X}_{\bar{i}} = \bar{X}_1 \times \ldots \times \bar{X}_{i-1} \times \bar{X}_{i+1} \times \ldots \times \bar{X}_N$, and let $\bar{x}_{\bar{i}}$ be a member of $\bar{X}_{\bar{i}}$. Then, $\bar{X}_i$ is utility independent of $\bar{X}_{\bar{i}}$ if ones preference order over lotteries on $\bar{X}_i$ with $\bar{X}_{\bar{i}}$ held fixed does not depend on the fixed amount $\bar{X}_{\bar{i}}$.

Keeney[43] has shown that if $\bar{X}_i$ is utility independent of $\bar{X}_{\bar{i}}$ for all dimensions, then Equation 4.1 takes a "quasi-additive" form.

Theorem 4.2

If $\bar{X}_i$ is utility independent of $\bar{X}_{\bar{i}}$ for i = 1, 2, 3,..., N, then

$$u(\bar{x}) = \sum_{i=1}^{N} k_i u_i(x_i) + \sum_{i=1}^{N} \sum_{j=i+1}^{N} k_{ij} u_i(x_i) u_j(x_j) + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{m=j+1}^{N} k_{ijm} u_i(x_i) u_j(x_j) u_m(x_m) + \ldots, \tag{4.2}$$

where k_i, k_{ij}, k_{ijm},... are scaling constants.

For the three attributes--precision, execution-time, and cost--lotteries over any attribute will not vary as the specific values of the remaining attributes change; e.g., more precision is always preferred

to less, less execution-time to more, and less cost to more. Thus, since $\bar{X}_i$ is utility independent of $\bar{X}_{\bar{i}}$ for all dimensions, then Equation 4.2 holds.

Yet in Equation 4.2 seven scaling constants must be evaluated, many of which represent the affect of "cross-product" terms between dimensions. Again, Keeney[44] has shown that under an additional condition these "cross-product" terms drop out.

Definition 4.10: Preferential Independence

Let $\bar{X}_{\overline{ij}} = \bar{X}_1 \times \ldots \times \bar{X}_{i-1} \times \bar{X}_{i+1} \times \ldots \times \bar{X}_{j-1} \times \bar{X}_{j+1} \times \ldots \times \bar{X}_N$, and let $\bar{x}_{\overline{ij}}$ be a member of $\bar{X}_{\overline{ij}}$. Then $\bar{X}_i \times \bar{X}_j$ is preferentially independent of $\bar{X}_{\overline{ij}}$ if ones preference order over lotteries on $\bar{X}_i \times \bar{X}_j$ with $\bar{X}_{\overline{ij}}$ held fixed does not depend on the fixed amount of $\bar{x}_{\overline{ij}}$.

Theorem 4.3

If $\bar{X}_i$ is utility independent of $\bar{X}_{\bar{i}}$ for one i, and $\bar{X}_i \times \bar{X}_j$ is preferentially independent of $\bar{X}_{\overline{ij}}$ for all $j \neq i$, then

$$u(\bar{x}) = \sum_{i=1}^{N} k_i u_i(x_i) \quad \text{(additive), or} \tag{4.3}$$

$$1 + ku(x) = \prod_{i=1}^{N} \{1 + kk_i u_i(x_i)\} \quad \text{(multiplicative)}, \tag{4.4}$$

where k and the k_i are scaling constants, $0 < k_i < 1$ and $k > -1$.

In Equation 4.3 $\sum_{i=1}^{N} k_i = 1$, and in Equation 4.2 if $\sum_{i=1}^{N} k_i > 1$ then $-1 < k < 0$ and if $\sum_{i=1}^{N} k_i < 1$ then $k > 0$.

Equation 4.3 represents an additive utility function since the

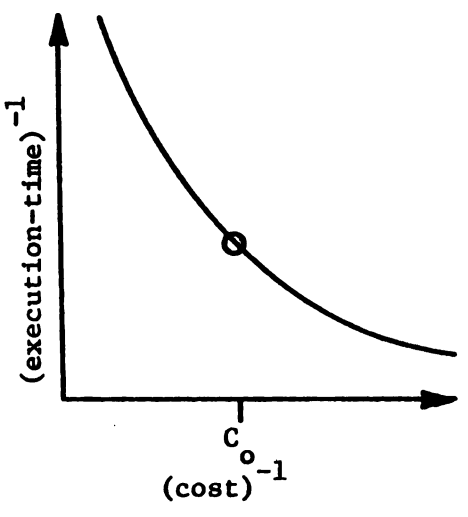
utility equals a weighted sum of marginal utilities, while Equation 4.4 depicts a multiplicative utility function because the utility equals a weighted product of marginal utilities. To distinguish between these two equations Keeney[45] offers the following corollary:

Corollary 4.1

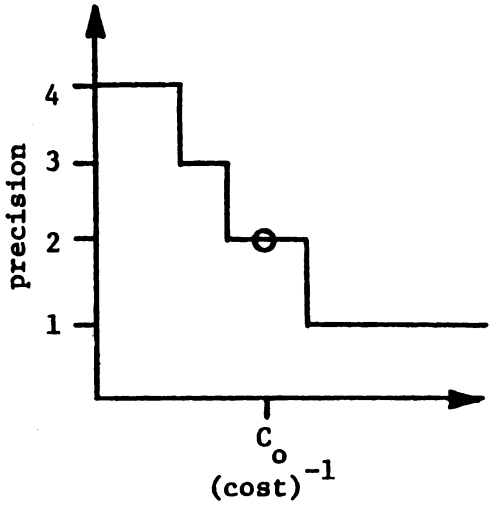
> Let $\bar{x}_i^1$, $\bar{x}_i^2$, $\bar{x}_j^1$, and $\bar{x}_j^2$ be distinct values of $\bar{X}_i$ and $\bar{X}_j$, and let $\bar{x}_{ij}$ take on some constant value. Define Gamble A as $(\bar{x}_i^1, \bar{x}_j^1, \bar{x}_{ij})$ or $(\bar{x}_i^2, \bar{x}_j^2, \bar{x}_{ij})$ each with probability 0.5, and Gamble B as $(\bar{x}_i^1, \bar{x}_j^2, \bar{x}_{ij})$ or $(\bar{x}_i^2, \bar{x}_j^1, \bar{x}_{ij})$ each with probability 0.5. If one is indifferent to Gamble A and B, then Equation 4.3 holds (additive).

For use of either Equation 4.3 or 4.4 the consequence set must satisfy the preferential independence assumption in addition to the utility independence assumption of Theorem 4.2. The preferential independence assumption states that the indifference curves between two dimensions do not vary as the other dimensions change. With indifference curves[46] a contour runs through a reference point which indicates the boundary between desired and undesired attribute pairs as in Figure 4.1. As the fixed amount $\bar{x}_{ij}$ changes the other attributes change, but the indifference curve remains constant. Hence, both conditions of Theorem 4.3 hold and the utility function adheres to the form of Equation 4.3 or 4.4.

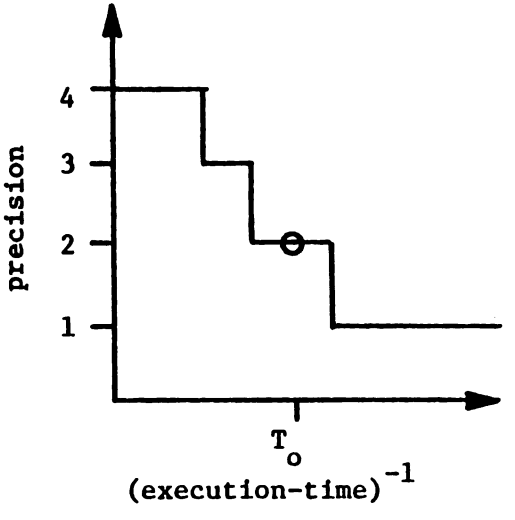
In Corollary 4.1 assume $\bar{x}_i^1 > \bar{x}_i^2$ and $\bar{x}_j^1 > \bar{x}_j^2$, then Gamble A is between a preferred consequence $(\bar{x}_i^1, \bar{x}_j^1, \bar{x}_{ij})$ and a non-preferred consequence $(\bar{x}_i^2, \bar{x}_j^2, \bar{x}_{ij})$ which both occur with probability 0.5. Similarly, Gamble B is between two consequences that both contain a preferred and non-preferred attribute and which occur with probability 0.5.



a) Execution-time and cost.



b) Precision and cost.



c) Execution-time and precision.

Figure 4.1 Indifference curves for computational design alternatives.

For attributes precision, execution-time, and cost neither Gamble A nor Gamble B represent a preferred wager and, consequently, Corollary 4.1 specifies the additive utility function of Equation 4.3. Thus, the utility function for the CDA's of Chapter III consist of a weighted sum of individual marginal utilities.

In this section the use of an additive utility function assumed both utility and preferential independence: two strong assumptions. Fortunately, Dawes, et al.[47] and Einhorn, et al.[48] point out that even modest deviations from utility and preferential independence rarely affect the ultimate number $u(\bar{x})$ and, even less, the rank ordering of the $u(\bar{x})$ values (utilities). They report that with monotonic attributes--where more is preferred to less, or less is preferred to more--the two independence assumptions cause little trouble. So for monotonic attributes precision, execution-time, and cost, the utility and preferential independence assumptions are plausible and, if not to an approximation, acceptable.

4.4 Marginal Utility Functions

As with the utility of a consequence, the marginal utility of an attribute indicates its usefulness relative to the other attribute values. Since the precision attribute describes a discrete variable its marginal utility must consist of a discrete function. And for the execution-time and cost attributes, both continuous variables, the marginal utilities require continuous functions. Other qualities desirable of marginal utility functions include a common numerical range; any interval will suffice but a convenient choice is $0 \leq u_i(x_i) \leq 100$. Finally, marginal utility functions should preserve the proportional distance

between attribute points; e.g., values twice as far apart should denote marginal utilities twice as far apart.

Using the previous ideas Edwards[49] has proposed the following procedure for determining marginal utility functions. First, the most and least desirable values and attribute are found and, then, a simple straight line is defined which yields 100 for the most desirable attribute and 0 for the least desirable attribute. This procedure yields the marginal utility functions for the attributes precision, execution-time, and cost.

Precision

Let p denote the precision where p = 1 for single-precision, p = 2 for double-precision, etc;

$$u_1 = (p - 1)\ 33.3;\ 1 \leq p \leq 4. \tag{4.5}$$

Execution-time

Let T signify the execution-time, and T_{max} and T_{min} the maximum and minimum values, respectively;

$$u_2 = \frac{(T_{max} - T)}{(T_{max} - T_{min})}\ 100. \tag{4.6}$$

Cost

Let C indicate the cost, and C_{max} and C_{min} the maximum and minimum values, respectively;

$$u_3 = \frac{(C_{max} - C)}{(C_{max} - C_{min})}\ 100. \tag{4.7}$$

Hence, for any set of CDA consequences Equations 4.5 through 4.7 convert the attribute values to marginal utilities, and Equation 4.3 forms the utility of each consequence as a weighted sum of its marginal utilities.

So the best member, or members, of the consequence set consist of those alternatives with the largest utility.

Not all attributes possess linear marginal utility functions as suggested in this section, and Raiffa[50] presents several techniques to assess nonlinear marginal utility functions. But Edwards[51] shows that for monotonic attributes the straight-line procedure produces a close, first-order approximations to the nonlinear approach with sample correlation coefficient 0.99. Since precision, execution-time, and cost all represent monotonic attributes, the linear approximation in this Section is credible and acceptable.

4.5 Summary

In this chapter, Multiattribute Utility Theory (MUT) adapted to selecting the "best" CDA, provides a decision mechanism which uses the attributes precision, execution-time, and cost. Here, MUT assigns a numeric quantity to each CDA which indicates its usefulness with respect to the other alternatives in the consequence set. Since the above attributes satisfy Axioms 4.1 through 4.8, Theorem 4.1 guarantees that utilities exist for all consequences. Furthermore, because the attributes also fulfill the utility and preferential independence assumptions of Theorem 4.3, the additive utility function holds which forms the utility of a CDA as a weighted sum of its marginal utilities (Eqs. 4.5 through 4.7 convert attributes to marginal utilities). In the next chapter, three examples illustrate the procedure for obtaining marginal utilities and, moreover, the use of this additive utility function. So the "best" alternative consists of the CDA with the largest utility.

CHAPTER V APPLICATIONS

In this chapter, three examples illustrate the results of the previous two chapters; they clarify the techniques used to determine attribute values and for typical consequence sets they elucidate use of an additive utility function. These examples--linear regression, matrix inversion, and fast Fourier transform computations--exemplify the overall procedure of this investigation and, moreover, they lead to conclusions which characterize each CDA. Not only do these applications typify representative and contemporary engineering problems, but they vary widely in their mathematical sophistication. First, a discussion of the general approach follows below.

5.1 General Approach

The general approach to using MUT applied to selecting CDA's involves five principal steps (see Fig. 5.1). First, the application must be defined; this includes stating all assumptions and conditions under which the computation remains valid. Then, the computation is expressed using detailed and complete notation. Since some parameters may vary their potential effect upon the computation must be considered. Next, the most difficult step in Figure 5.1 concerns determining the attribute values; i.e., precision, execution-time, and cost. To accomplish this, begin by preparing a program flowchart which implements the computation using a series approach. And by examining it for sections

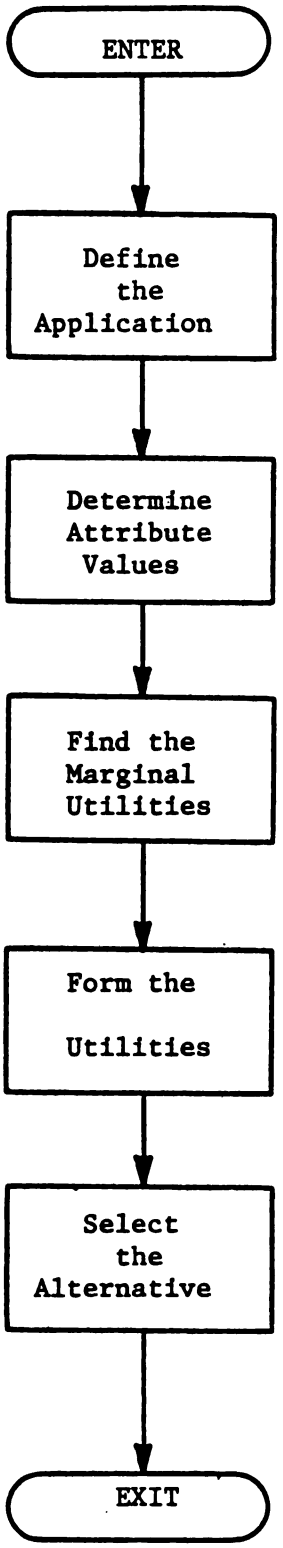


Figure 5.1 General approach to selecting a computational design alternative.

which may be executed simultaneously, or in parallel, construct a second program flowchart which implements a parallel approach (Master and Slaves). All but the mμP CDA uses the parallel approach to assist in finding attribute values. From these flowcharts the total number of elementary operations; e.g., adds/subtracts and multiplies/divides, or constants A_i and M_i, can be estimated for each CDA. By multiplying these terms by the multiple-precision add and multiply times (see Eq. 3.1 and Table 3.1), a_i and m_i, the execution-times are found as follows:

$$T_i = A_i a_i + M_i m_i.$$

Again, the two flowcharts can be used to estimate both the program and data-storage memory requirements, P_i and D_i respectively, for each CDA. They yield the application cost term (see Eq. 3.12), C_{ai}, as follows:

$$C_{ai} = w_o(P_i + pD_i),$$

and together with the elemental CDA cost term (see Eq. 3.1 and Table 3.2), C_{ei}, give the total cost of each alternative:

$$C_i = C_{ei} + C_{ai}.$$

To determine program storage memory requirements consider the contributions of an arbitrary loop and subroutine call:

```
loop
        .
        .
        .
        LDR         COUNT                   /REG = COUNT
        NEG                                 /REG = -COUNT
LOOP    ---
        .
        .
        .
        INC                                 /REG = REG + 1
```

```
        JNZ         LOOP                    /REG = 0?, NO JMP LOOP
          .
          .
          .
```

subroutine call

```
          .
          .
          .
        LD1         AD1                     /OPERAND 1 ADR
        LD2         AD2                     /OPERAND 2 ADR
        LD3         AD3                     /RESULT ADR
        CALL        SUB
          .
          .
          .
```

Thus, for estimating P_i each loop and each subroutine call contribute four words.

In the third step of Figure 5.1 the attribute values are converted to marginal utilities as explained in Chapter IV. Precision, a discrete variable, obeys a discrete function, while execution-time and cost, both continuous variables, follow a linear transformation. And for all attributes the marginal utilities lie within the same interval; i.e., [0, 100]. The next step invokes an additive utility function (shown to be valid in Chapter IV) upon the marginal utilities; it forms the utility of each CDA as a weighted sum of it's marginal utilities. Hence, prior to the use of this function a "Decision Maker" must assess the value of the weights, or k_i's, subject to the condition that $\sum_{i=1}^{N} k_i = 1$. Finally, the last step of Figure 5.1 concerns selecting the CDA with the largest total utility. Or if more than one alternative is

to be selected, then choose those with the greatest utility. By varying the consequence set, some alternatives may result in equal utilities which indicate the "break-even" point between those CDA's. Also, differing values of the k_i's produce new utility values which illustrate the "sensitivity" of a CDA to each attribute dimension; thus, the utilities in the dimension of emphasized k_i increases, while the others decrease.

5.2 Linear Regression Example

In this example a set of N ordered pairs $\{(x_i, y_i) \mid i = 1, 2, 3, \ldots, N\}$ are fit by the method of least-squares[52] to a straight line $y = b_o + b_1 x$. The solutions to the normal equations

$$nb_o + b_1 \Sigma x_i = \Sigma y_i,$$

$$b_o \Sigma x_i + b_1 \Sigma x_i^2 = \Sigma x_i y_i,$$

yield the unknown constants b_o and b_1; i.e.,

$$b_1 = \frac{n \Sigma x_i y_i - (\Sigma x_i)(\Sigma y_i)}{N \Sigma x_i^2 - (\Sigma x_i)^2} .$$

$$b_o = \frac{\Sigma y_i}{N} - b_1 \frac{\Sigma x_i}{N}.$$

For this example it is assumed the x's are fixed variables and the y's independent random variables having normal distributions and with common variance σ^2.

The series program flowchart contains six principal steps and no major loops (see Fig. 5.2). To estimate the total number of adds and multiplies, A_i and M_i, the contributions of each step can be summed as shown in Table 5.1. These terms, when multiplied by the multiple-precision add and multiply times, combine to give the series execution-time

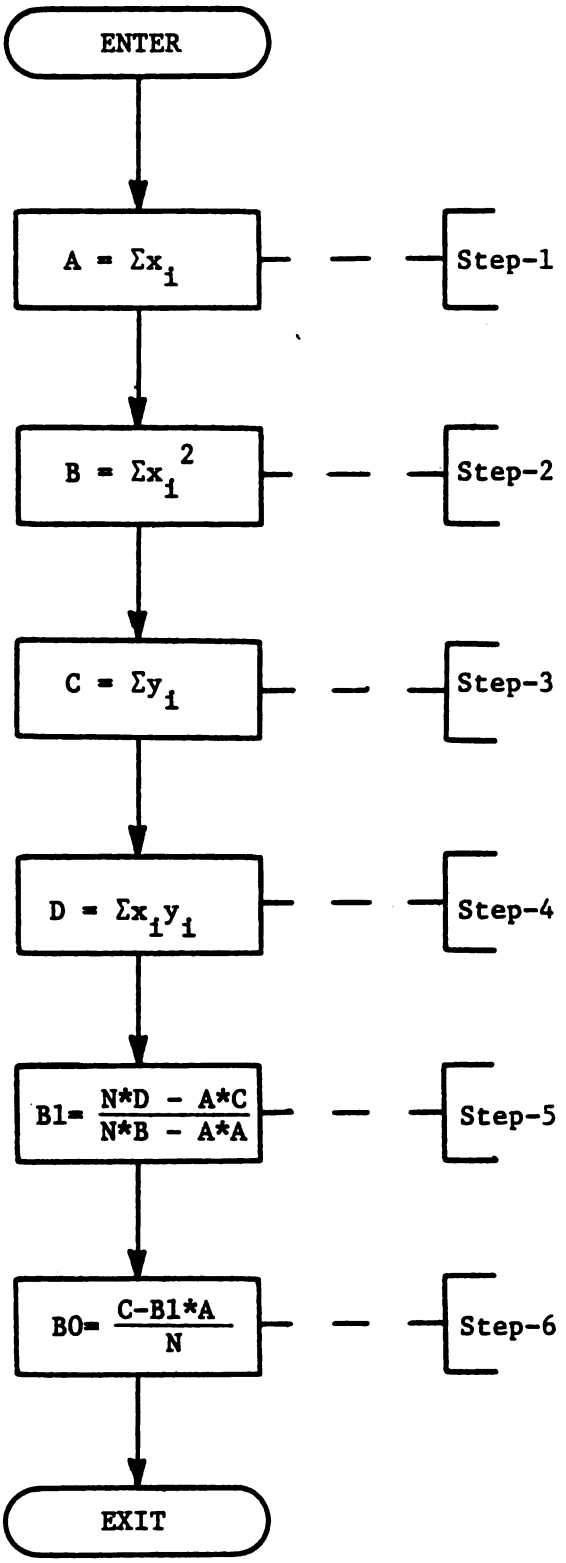


Figure 5.2 Linear regression series flowchart.

Table 5.1 Estimates of A_i, M_i ($i \neq 4$) for Example-1.

Step	Adds	Multiplies
1	N	--
2	N	N
3	N	--
4	N	N
5	2	5
6	1	2
Totals	$A_i = (4N + 3)$	$M_i = (2N + 7)$

Table 5.2 Estimates of A_4, M_4 for Example-1.

Step	Adds	Multiplies
1	3N	--
2	2N	--
3	N	N
4	4	--
5	2	5
6	1	2
Totals	$A_4 = (6N + 7)$	$M_4 = (N + 7)$

$$T_s = T_i = (4N + 3)a_i + (2N + 7)m_i;\ i \neq 4. \tag{5.1}$$

By examining Figure 5.2 it is possible to identify those sections of the series flowchart which can be executed simultaneously, or in parallel. Specifically, four slaves can perform Steps 1 to 4 once they receive the correct x's and/or y's. The responsibility of the Master then becomes data routing, not data computing as shown in Figure 5.3 and 5.4. Again, the total number of adds and multiplies can be found by summing the contribution of each step in the parallel flowchart as detailed in Table 5.2. By multiplication of these terms by the multiple-precision add and multiply times the parallel execution-time is found as

$$T_p = T_4 = (6N + 7)a_4 + (N + 7)m_4. \tag{5.2}$$

(Note: the "SLAVES BUSY" term can be deduced from Figure 5.5)

An important result occurs by defining R, the ratio of series to parallel execution-time as

$$R = \frac{T_s}{T_p},$$

and assuming $m_i = 8a_i$.[53] From Equations 5.1 and 5.2

$$R = \frac{(4N + 3)a_i + (2N + 7)\ (8a_i)}{(6N + 7)a_i + (N + 7)\ (8a_i)},$$

$$= \frac{20N + 59}{14N + 63},$$

and as N grows large

$$\lim_{N \to \infty} R = \frac{20}{14} = 1.43;$$

this represents a maximum decrease in execution-time by a factor of 1.43 due to simultaneous, or parallel, execution. Although the Slaves

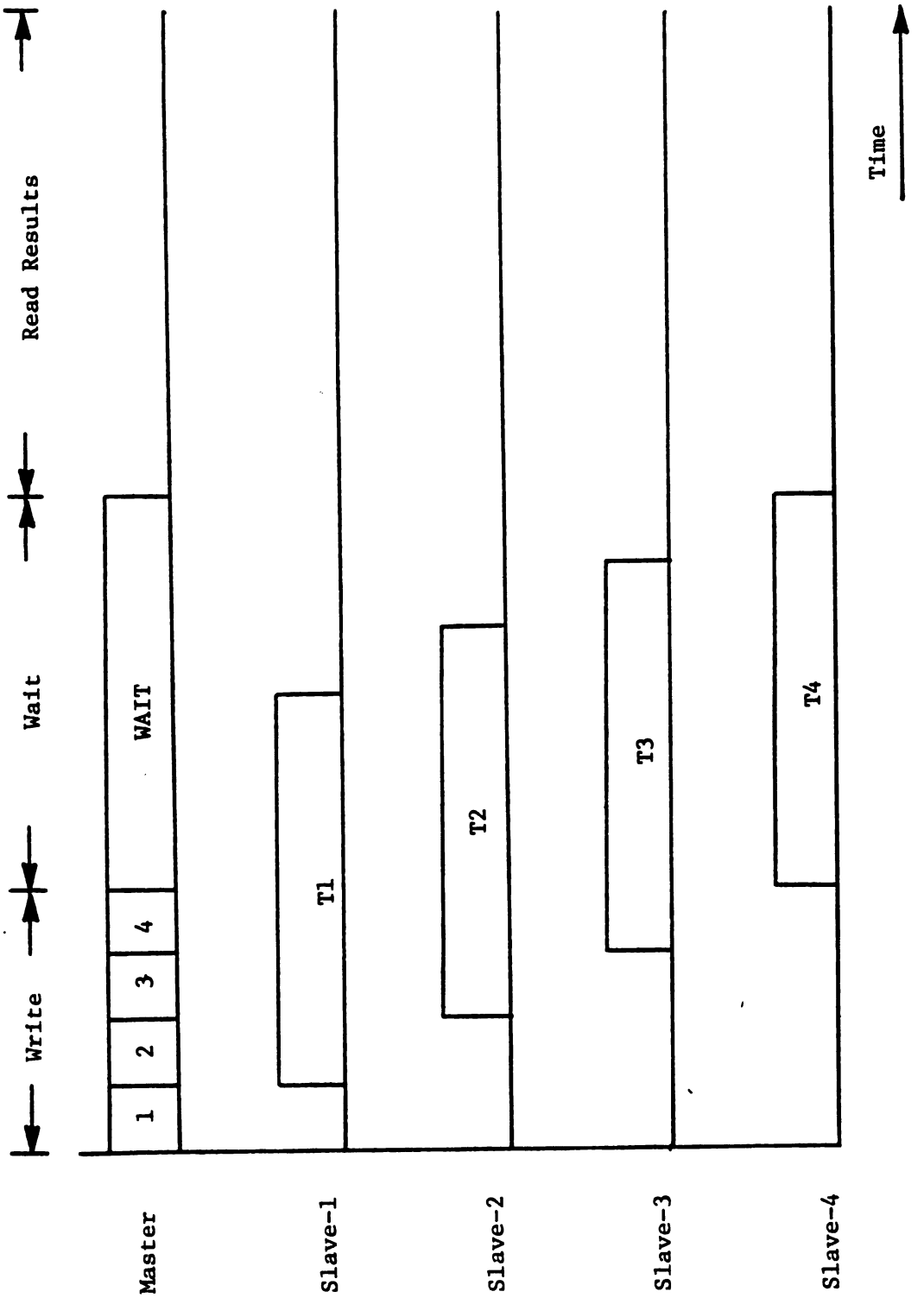


Figure 5.3 Relation between Master and Slaves for Example-1.

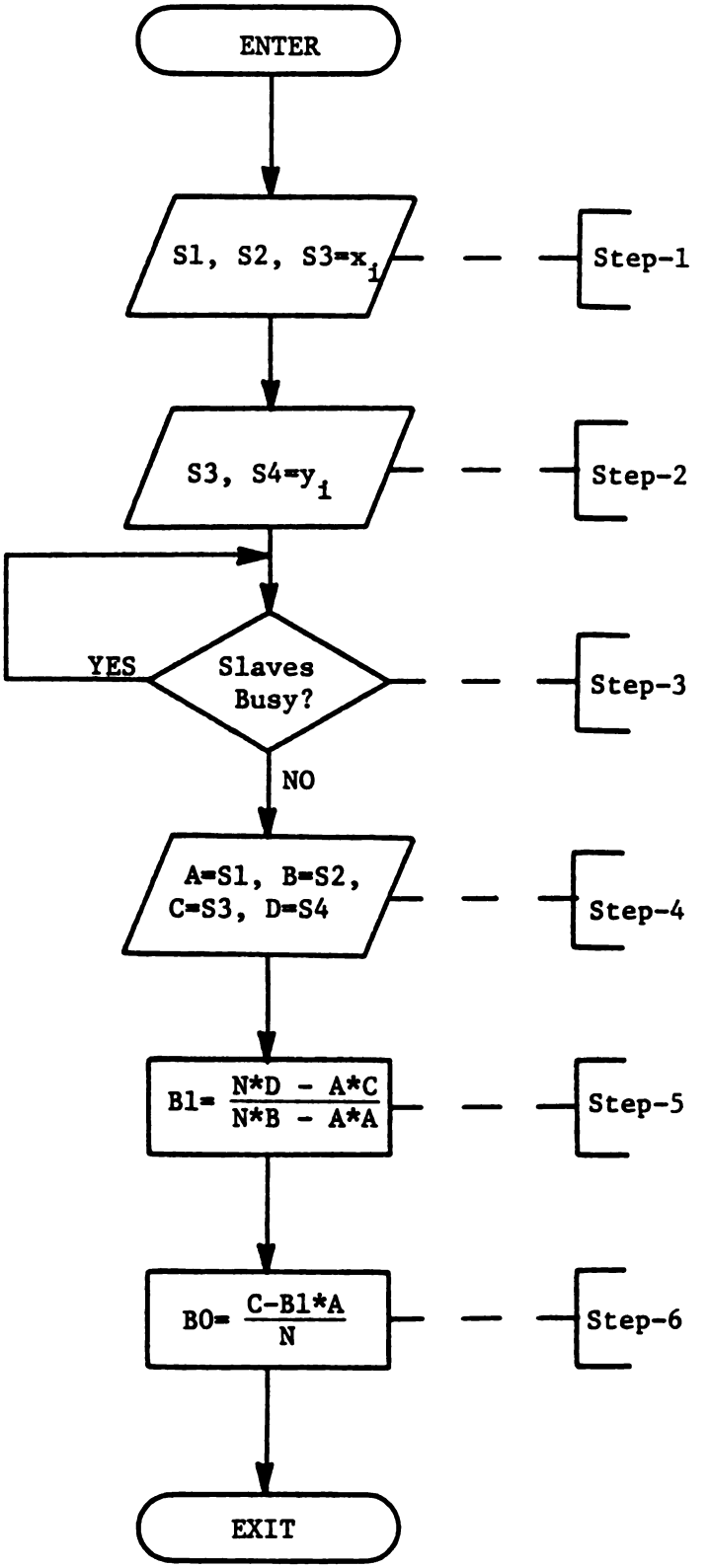


Figure 5.4 Linear regression parallel flowchart: Master.

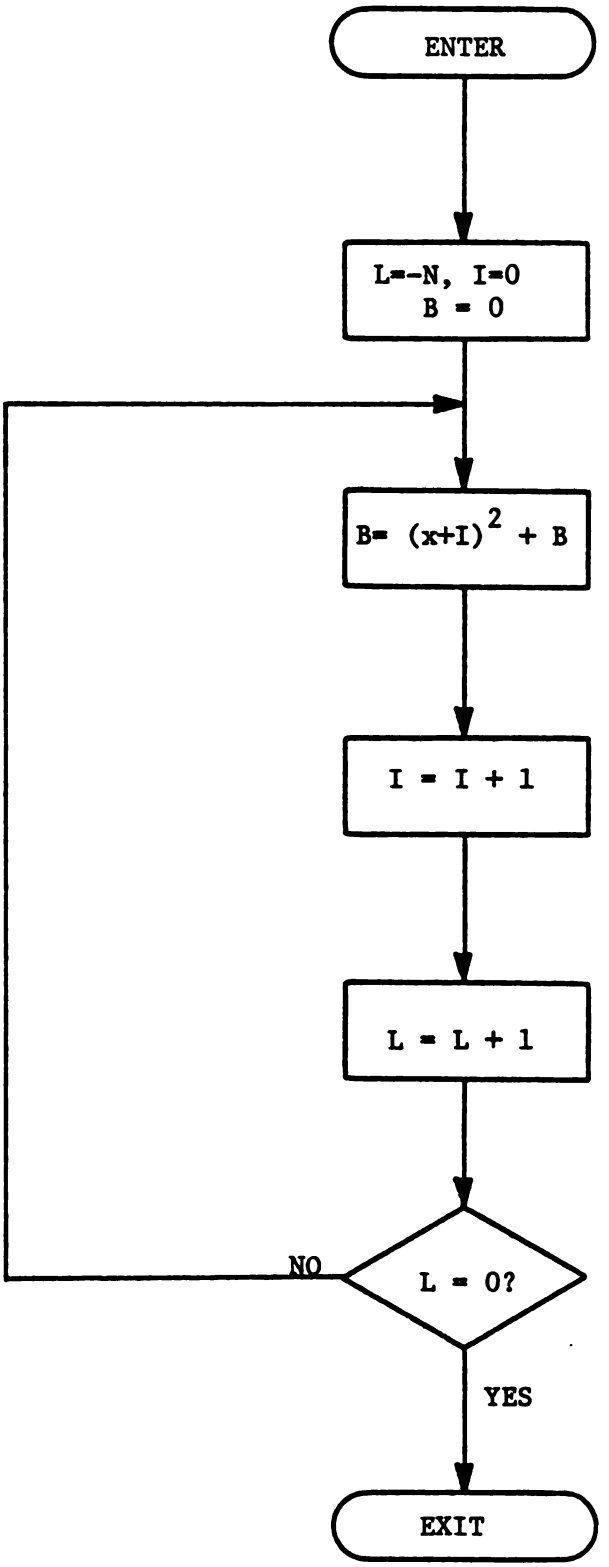


Figure 5.5 Linear regression parallel flowchart: Slave-2.

execute in parallel, the Master's overhead involves vast data routing and, thus, a limit to the reduction in execution-time. If the sampled-data originated with the Slaves, then R would decrease.

When determining the application memory cost, C_{ai}, all subroutine calls (adds/subtracts and multiplies/divides) require four words of program storage and each loop demands an additional four words of program storage. As with execution-time, the series and parallel program flowcharts (Fig. 5.2, 5.4) assist in estimating the application memory cost. Similarly, the contributions of each flowchart step add together to yield the final result. Table 5.3 outlines the steps and their contributions for the series flowchart; these terms combine as follows:

$$C_{ai} = w_o\,[80 + p(2N + 6)];\ i \neq 4. \tag{5.3}$$

But for the mμP CDA (i = 4) the slaves create additional terms of program and data memory storage. Figure 5.5 depicts a worst-case slave flowchart, Slave Two, which illustrates the additional contributions due to the four slaves. Adding these terms to the Master's terms gives (see Table 5.4) the application memory cost C_{a4}; i.e.,

$$C_{a4} = w_o\,[112 + p(7N + 6)]. \tag{5.4}$$

Combined with the results of Chapter III Equations 5.1 through 5.4 produce the set of attribute values for this example (see Fig. 5.6 through 5.13). Similarly, the techniques of Chapter IV convert these values to marginal utilities and, finally, to utilities as shown in Tables 5.5 and 5.6.

From inspection of the figures and tables, several conclusions become clear. First, the AU CDA always possesses the fastest execution-time due to its hardware multiply and divide. For all cases, the mμP

Table 5.3 Estimates of P_i, D_i ($i \neq 4$) for Example-1.

Step	Program	Data
1	8	N + 1
2	12	1
3	8	N + 1
4	12	1
5	28	1
6	12	1
Totals	P_i = (80)	D_i = (2N + 6)

Table 5.4 Estimates of P_4, D_4 for Example-1.

Step	Program	Data
1	16	N
2	12	N
3	--	--
4	4	4
5	28	1
6	12	1
Slaves (4)	40	5N
Totals	P_4 = (112)	D_4 = (7N + 6)

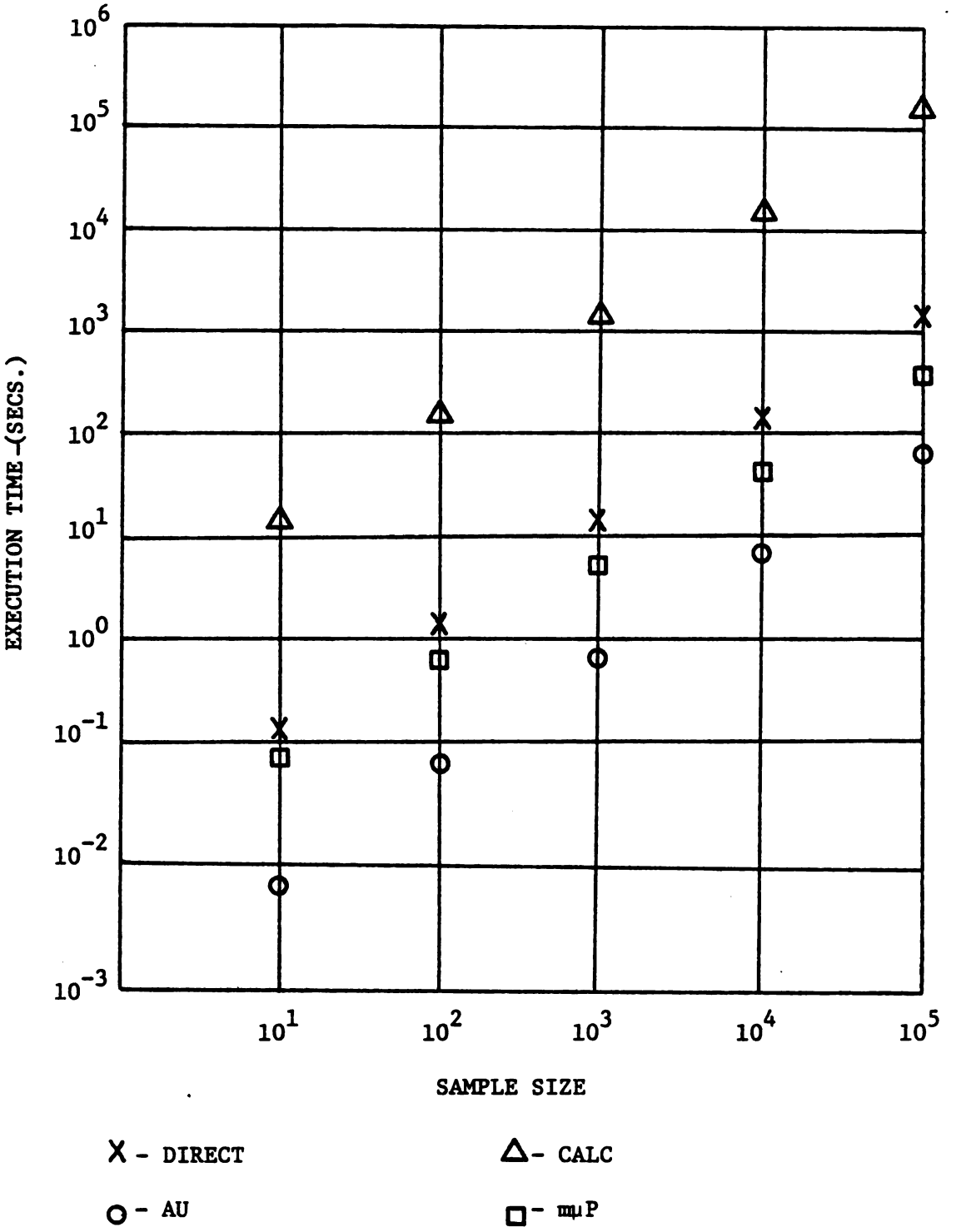


Figure 5.6 Single-precision execution-time versus sample size for Example-1.

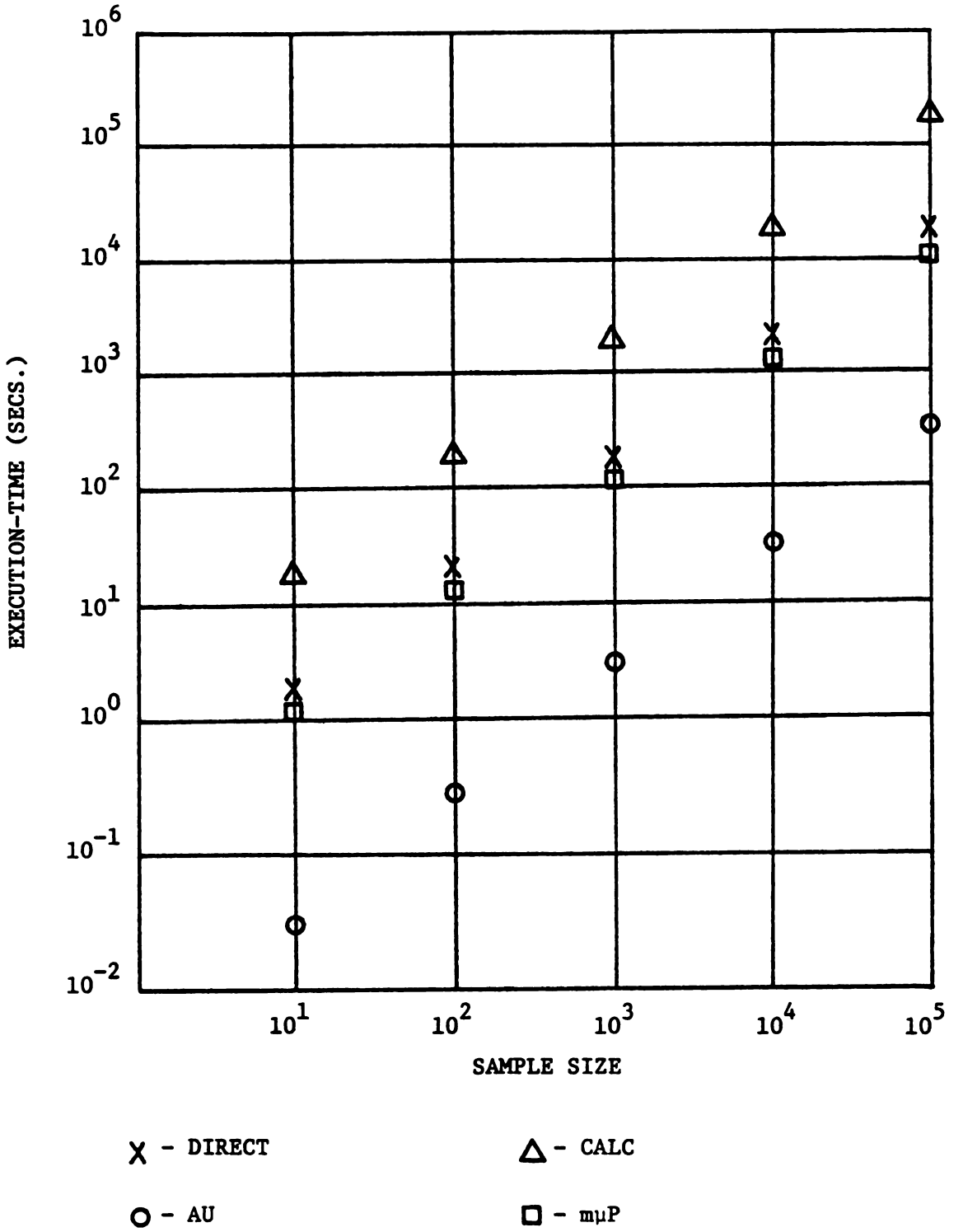


Figure 5.7 Double-precision execution-time versus sample size for Example-1.

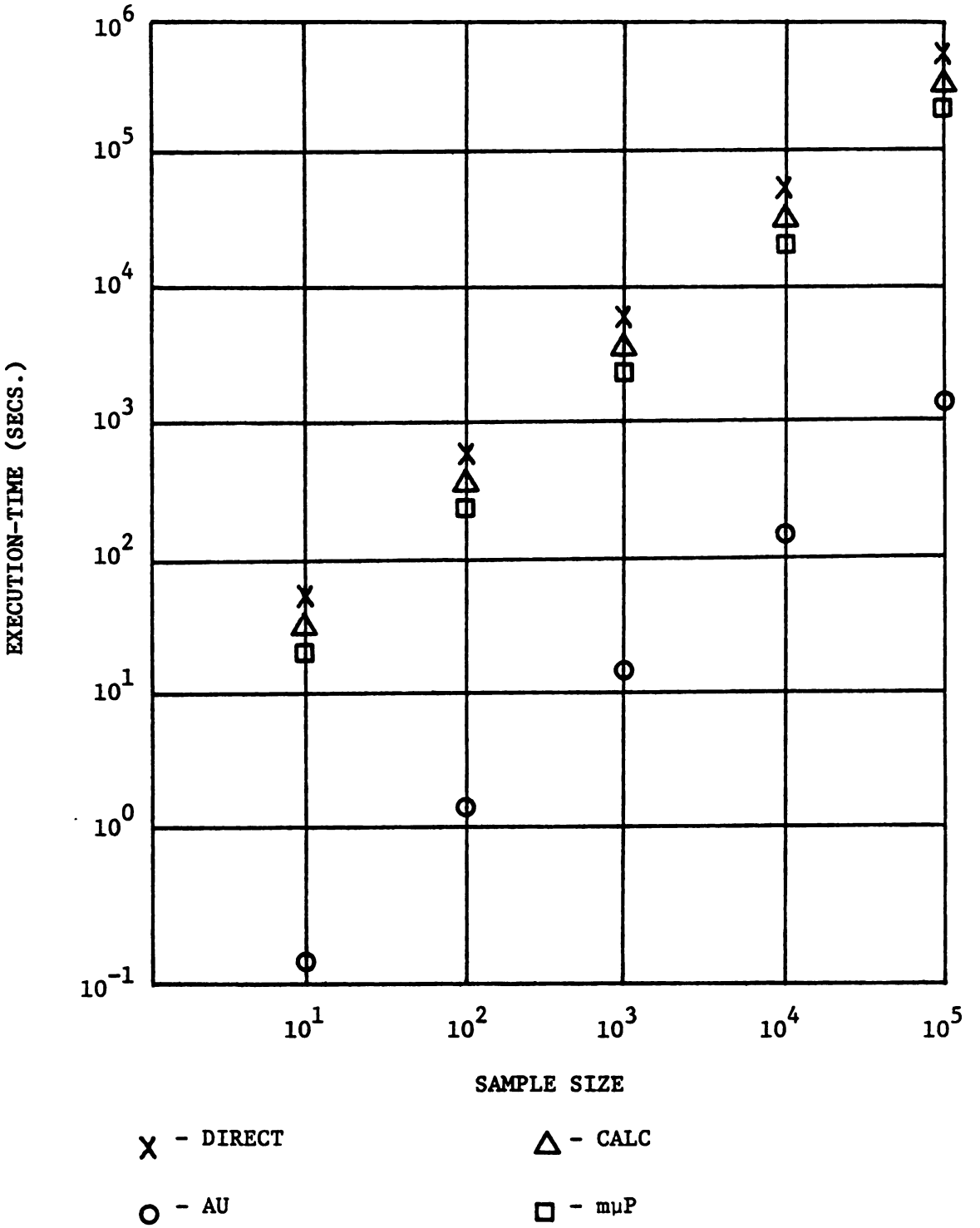


Figure 5.8 Triple-precision execution-time versus sample size for Example-1.

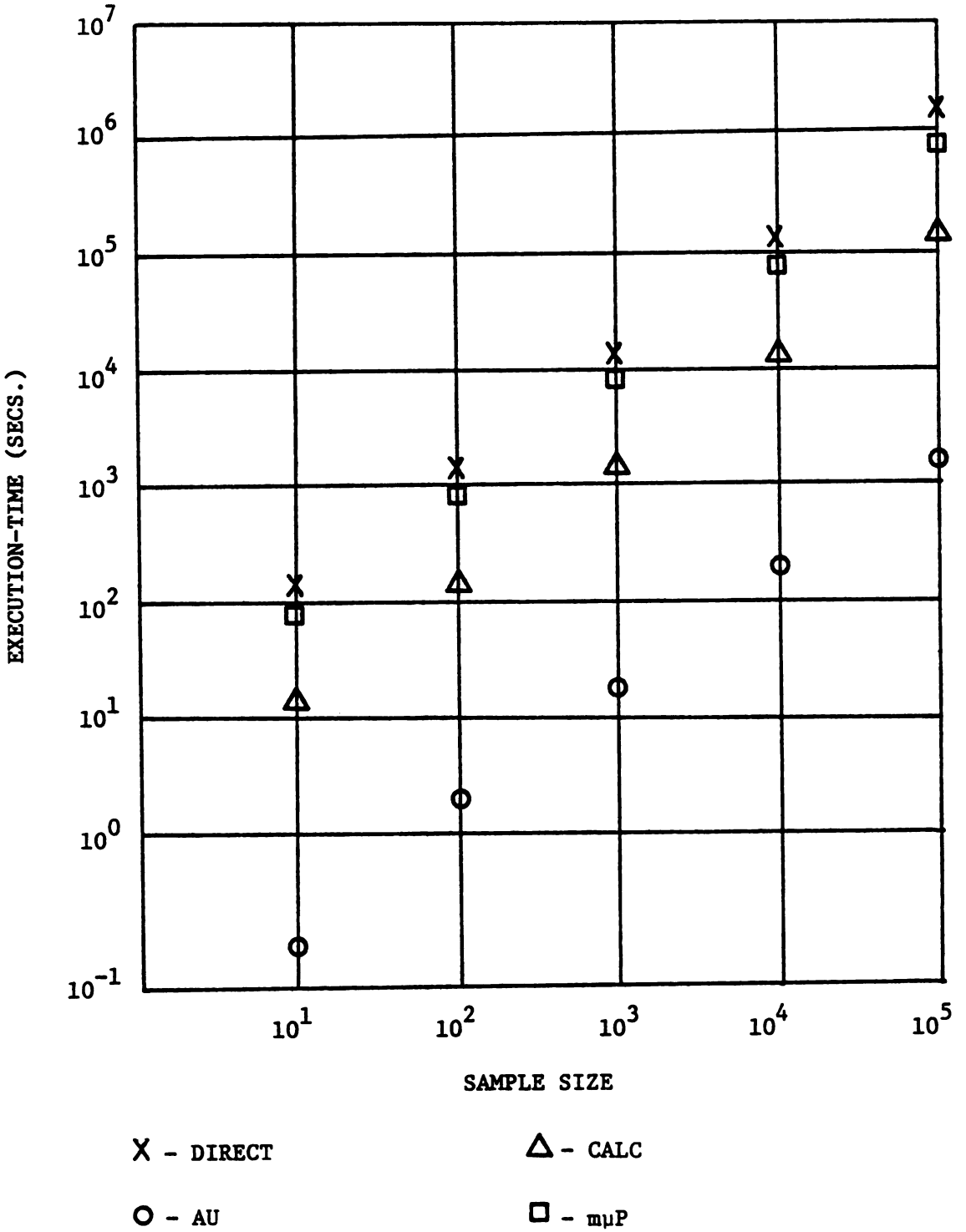


Figure 5.9 Quadruple-precision execution-time versus sample size for Example-1.

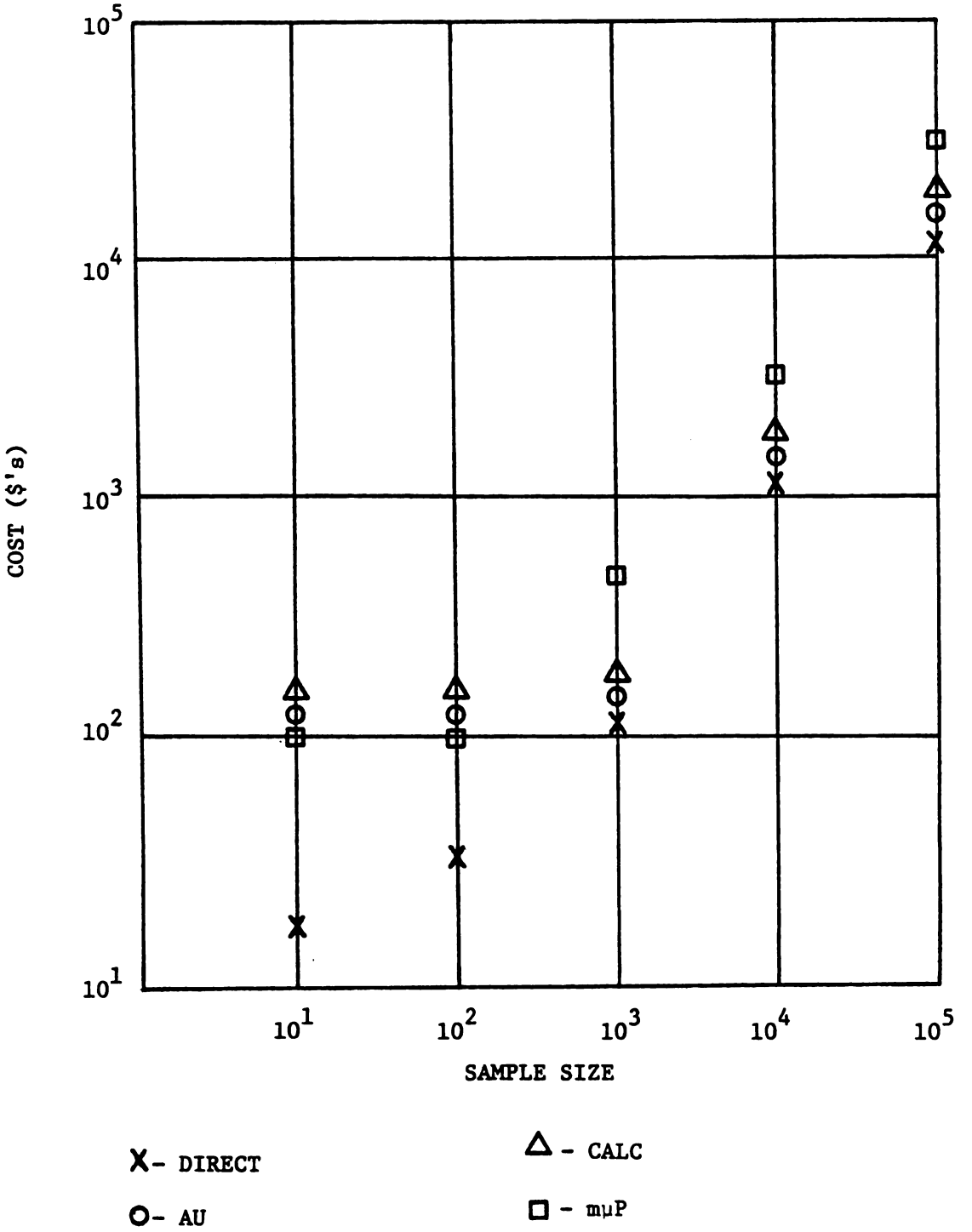


Figure 5.10 Single-precision costs versus sample size for Example-1.

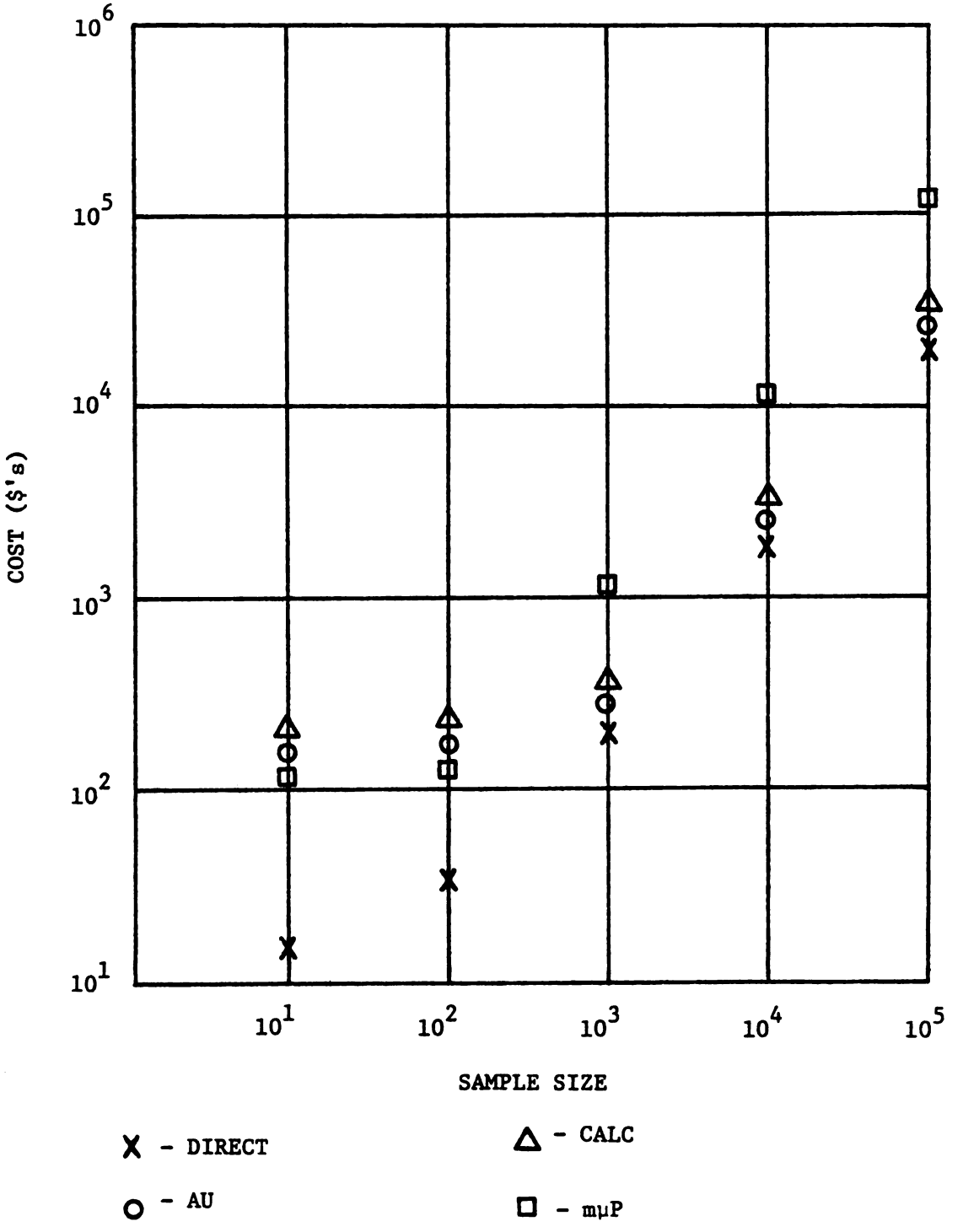


Figure 5.11 Double-precision costs versus sample size for Example-1.

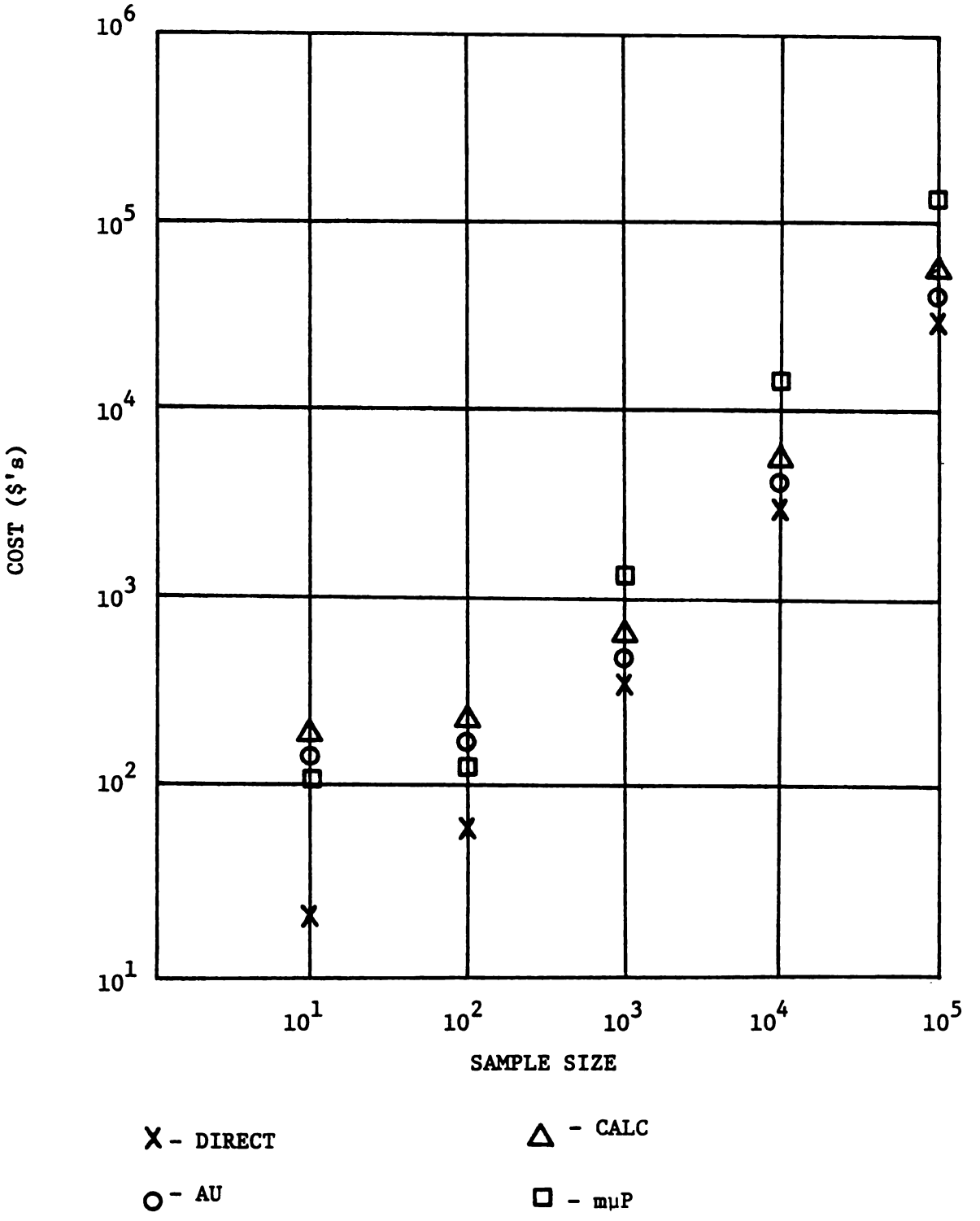


Figure 5.12 Triple-precision costs versus sample size for Example-1.

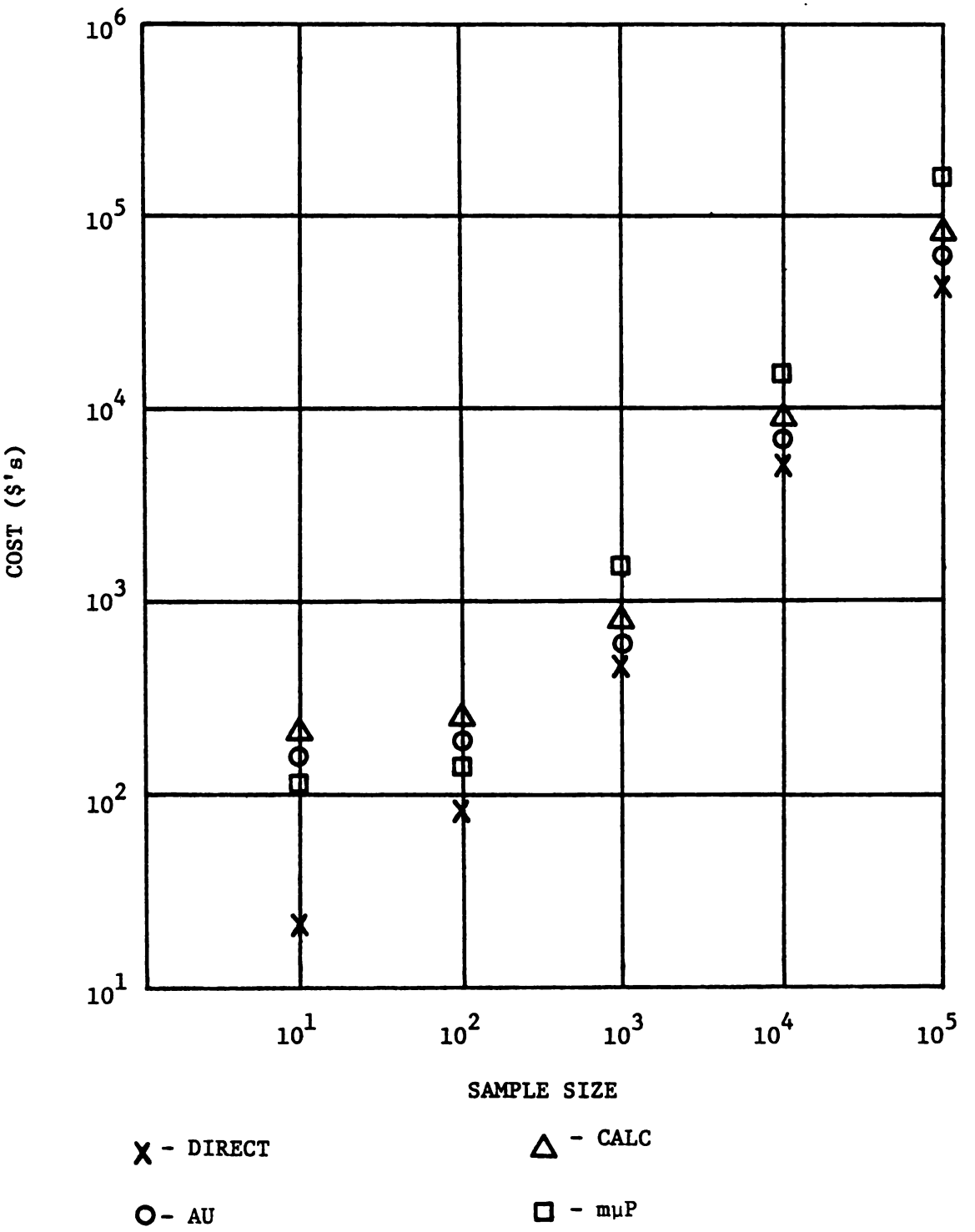


Figure 5.13 Quadruple-precision costs versus sample size for Example-1.

Table 5.5 Typical consequence set and utilities for example-1 (10^1 samples).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	7.08E1	0	55	63	17	87
	3.06E1	100				
	1	0				
2	8.84E-3	100	51	15	85	52
	1.30E2	53				
	4	100				
3	5.74E1	19	40	82	25	12
	2.41E2	0				
	2	33				
4	2.1E0	97	64	42	87	62
	1.11E2	62				

Note: $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), $\bar{k}_3$ = (0.1, 0.1, 0.8)

Table 5.6 Typical consequence set and utilities for example-1(10^5 samples).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	5.24E5	00	44	60	13	60
	4.80E4	67				
	1	00				
2	7.20E1	100	67	20	90	90
	1.61E4	100				
	4	100				
3	4.91E5	6	52	86	20	51
	6.42E4	50				
	2	33				
4	1.24E4	98	44	36	81	13
	1.12E5	0				

Note: $\bar{k}_a = (0.3, 0.3, 0.3)$, $\bar{k}_1 = (0.8, 0.1, 0.1)$, $\bar{k}_2 = (0.1, 0.8, 0.1)$, $\bar{k}_3 = (0.1, 0.1, 0.8)$

CDA shows reduced execution-time when compared to the DIRECT CDA even though they perform multiplies and divides in software. Here, R = 1.5 which agrees well with the calculated value and, thus, confirms the validity of the $m_i = 8a_i$ assumption. Also, the DIRECT/mμP CDA's for p = 1, 2 execute faster than the CALC, yet for p = 4 this reverses because of the repetitive adds associated with the software multiplies and divides. All costs rise with larger N due to additional data-storage memory, especially after 10^3 samples, but the mμP costs go up higher (more memory needed). From Table 5.5, for weight vector $\bar{k}_a$, the mμP CDA represents the "best" CDA for the typical consequence set and small problem dimensions (i.e., 10^1 samples). Similarly, Table 5.6 reveals that the AU CDA depicts the "best" CDA for large problem dimensions (i.e., 10^5 samples). As the k_i's vary, the CDA's with strong marginal utilities in the dimension of the emphasized k_i show increased utility, and the rank ordering of the CDA's changes accordingly. For example, in Table 5.5 the CALC CDA exhibits the greatest "sensitivity" to the precision dimension as the weight vector changes from $\bar{k}_a$ to $\bar{k}_i$; i.e., its utility varies from 40 to 82, respectively. Also, the DIRECT and mμP CDA's (for $\bar{k}_a$) in Table 5.6 correspond to "break-even" alternatives.

5.3 Matrix Inversion Example

For the second application a square, nonsingular matrix A of order N is inverted. A standard approach, that of Guassian elimination[54] is used; this technique performs a sequence of elementary row operations on the partitioned matrix [A, I] to yield the matrix [I, A^{-1}]: where I is the identity matrix.

The series flowchart for this technique includes two principal steps imbedded within two major loops as shown in Figure 5.14. Here,

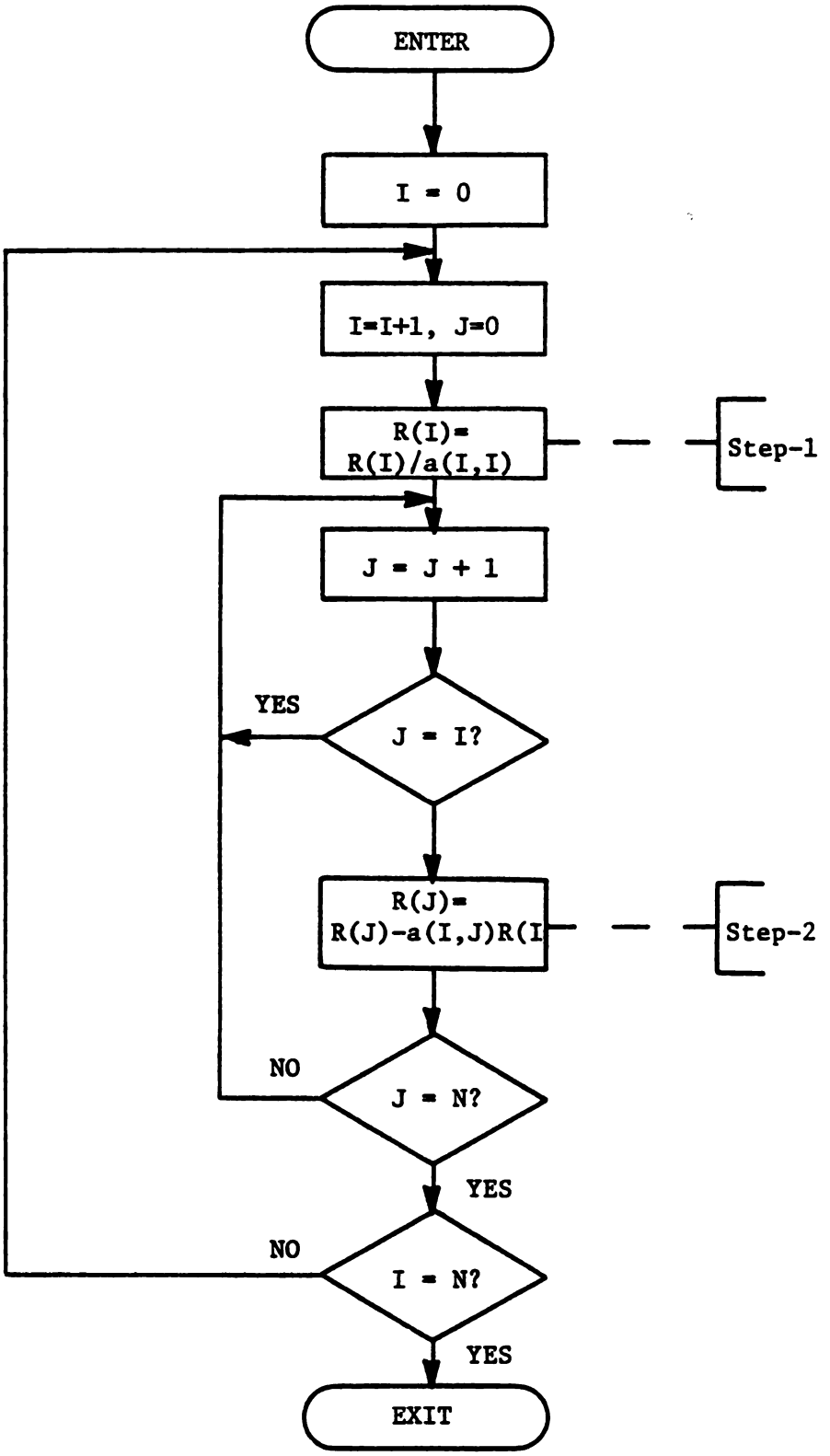


Figure 5.14 Matrix inversion series flowchart (Gaussian Elimination technique).[55]

Step-1 replaces row I with itself divided by the pivot entry a(I, I): hence, the pivot entry becomes unity. Once completed, Step-2 uses row I to reduce the remainder of column I to zeros. Together with the two loops, Steps 1 and 2 transform the left submatrix to the identity matrix and the right submatrix to the inverse of A; i.e., $[I, A^{-1}]$. To estimate the total number of adds and multiplies, A_i and M_i, sum up the contribution of each step and multiply this by the number of times through each loop (see Table 5.7). These terms, when multiplied by the multiple-precision add and multiply times, combine to give the series execution-time

$$T_s = T_i = 2N^2(N - 1)a_i + 2N^3m_i;\ i \neq 4. \tag{5.5}$$

Equation 5.5 agrees with published results.[56]

By examining Figure 5.14 it is possible to identify those sections of the series flowchart that can be executed simultaneously, or in parallel. Since Step-2 only requires the results of Step-1, N slaves can perform Step-2 simultaneously and the Master flowchart now involves just one loop (see Fig. 5.15 and 5.16). Again, to find the total number of adds and multiplies sum the contribution of each step in the parallel flowchart as detailed in Table 5.8. (Note: for an estimate of the "SLAVE BUSY" term use Figure 5.17) Multiplying these results by the multiple-precision add and multiply times yields the parallel execution-time

$$T_p = T_4 = 2N^2(N + 1)a_4 + 2N^2m_4. \tag{5.6}$$

Again, the ratio of series to parallel execution-times produces an indication of the decrease in execution-time due to simultaneous, or parallel, execution. Using Equations 5.5 and 5.6 with the assumption

$m_i = 8a_i$

Table 5.7 Estimates of A_i, M_i (i = 4) for Example-2.

Step	Adds	Multiplies
1	--	$2N^2$
2	$2N^2(N - 1)$	$2N^2(N - 1)$
Totals	$A_i = 2N^2(N - 1)$	$M_i = 2N^3$

Table 5.8 Estimates of A_4, M_4 for Example-2.

Step	Adds	Multiplies
1	$2N^2$	--
2	$2N^2(N - 1)$	--
3	$2N^2$	$2N^2$
Totals	$A_4 = 2N^2(N + 1)$	$M_4 = 2N^2$

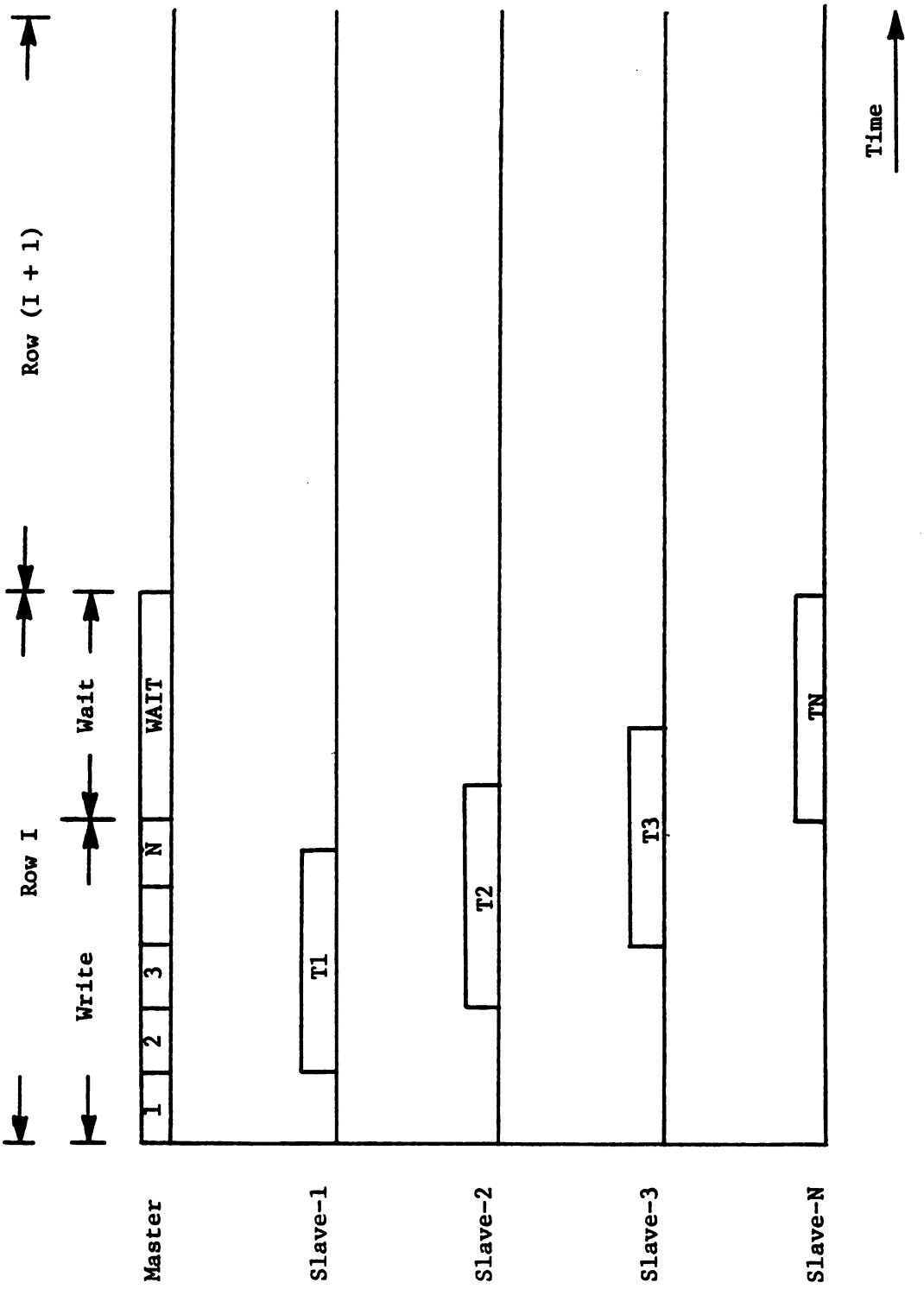


Figure 5.15 Relation between Master and Slaves for Example-2.

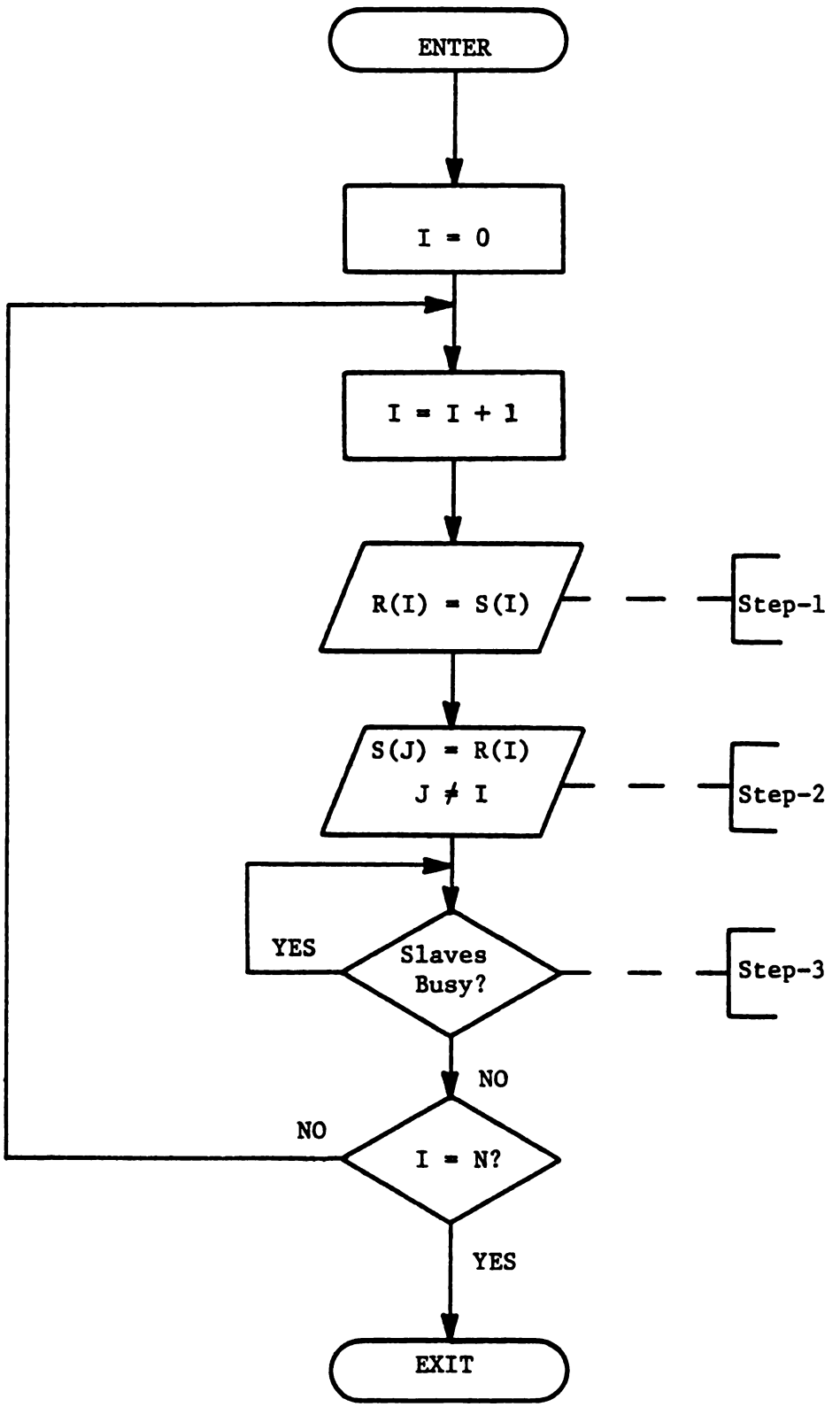


Figure 5.16 Matrix inversion parallel flowchart: Master

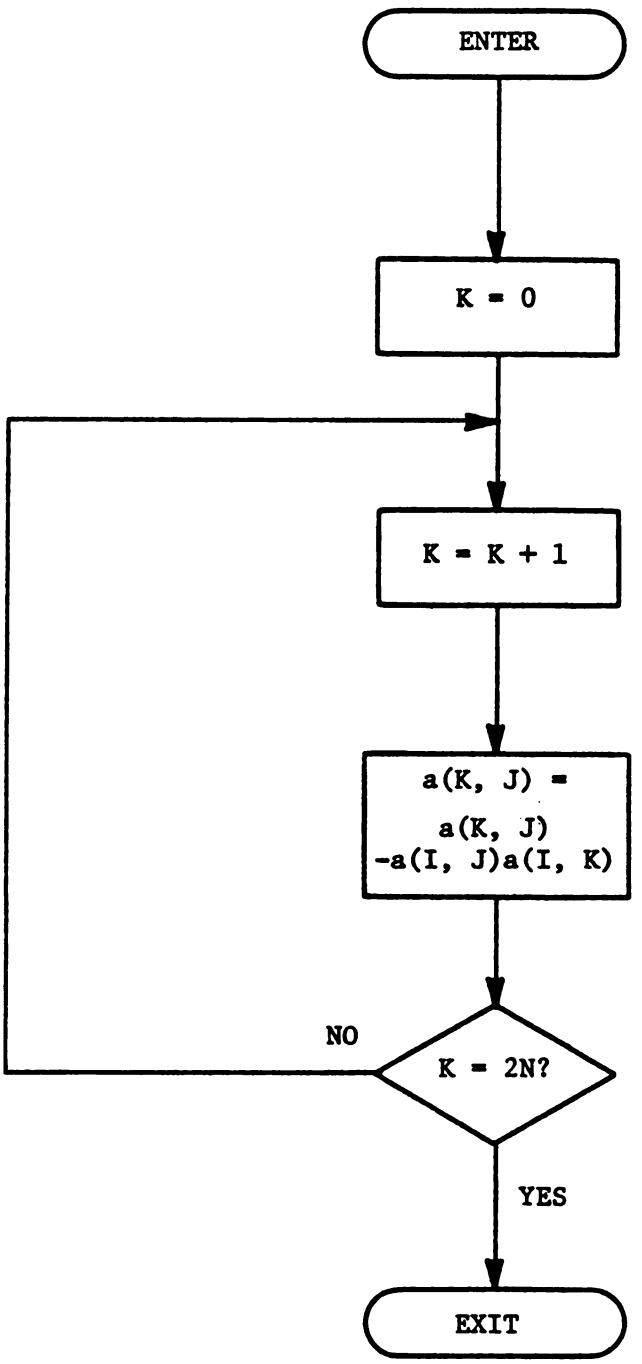


Figure 5.17 Matrix inversion parallel flowchart: Slave-J.

$$R = \frac{T_s}{T_p} = \frac{2N^2(N-1)a_i + 2N^3(8a_i)}{2N^2(N+1)a_i + 2N^2(8a_i)},$$

$$= \frac{18N - 2}{2N + 20}.$$

And as N grows large,

$$\lim_{N \to \infty} R = \frac{18}{2} = 9,$$

the execution-time decreases, at best, by a factor of 9. Since this problem yields a greater R than Example-1, it exhibits greater parallel traits and, thus, better fits simultaneous execution concepts. Again, the Master's overhead must pass a new row to (N - 1) Slaves for each column and, eventually, this action becomes more time consuming than the parallel execution of the Slaves.

In determining the application memory cost, C_{ai}, all subroutine calls (add/subtract and multiply/divide) require four words of program storage and each loop demands an additional four words of program storage. By using the flowcharts of Figures 5.14 through 5.17 the estimates for the program and data storage can be determined by summing the contributions of each step. Table 5.9 delineates the steps and their contribution for the series flowchart; combining terms

$$C_{ai} = w_o(28 + p2N^2); \; i \neq 4. \tag{5.7}$$

But for the mμP CDA (i = 4) the N slaves create additional program and data storage requirements. When added to the Master's terms (see Table 5.10) they sum to give the mμP application memory

$$C_{a4} = w_o \; [4(3N + 5) + p2N(N + 1)]. \tag{5.8}$$

Finally, the results of Chapter III and Equations 5.5 through

Table 5.9 Estimates of P_i, D_i ($i \neq 4$) for Example-2.

Step	Program	Data
1	12	$2N^2$
2	16	--
Totals	$P_i = 28$	$D_i = 2N^2$

Table 5.10 Estimates of P_4, D_4 for Example-2.

Step	Program	Data
1	8	2N
2	12	--
3	--	--
Slaves (N)	12N	$2N^2$
Totals	$P_4 = 4(3N + 5)$	$D_4 = 2N(N + 1)$

5.8 combined to produce the set of attribute values for this example (see Fig. 5.18 through 5.25). Additionally, the techniques of Chapter IV convert these values to marginal utilities and, lastly, to utilities as shown in Tables 5.11 and 5.12.

Again, these figures and tables expose several important CDA features. The AU CDA continues to perform the task fastest while the mμP CDA always beats the DIRECT CDA. But as N increases the last two execution-times separate and for even small N, R = 5 already. Similarly, for p = 1, 2 the CALC CDA shows the slowest speed and for p = 4 the order changes; i.e., the CALC CDA finishes before either the DIRECT or mμP CDA's. With respect to costs, they do not change as dramatically as in Example-1 because the program and data-storage memory requirements do not rise as sharply. Still the mμP costs grow exceedingly fast as N changes due to the increased number of Slaves as reflected in the elemental cost term C_{e4}. For both typical consequence sets and weight vector $\bar{k}_a$, the tables reveal that the mμP CDA represents the overall, "best" CDA. In Table 5.11 for $\bar{k}_3$, the DIRECT and mμP CDA's illustrate "break-even" utilities; i.e., 87 and 86, respectively. Thus, these CDA's correspond to equally useful alternatives. And finally, Table 5.12 for weight vector $\bar{k}_2$, shows the "sensitivity" of the AU CDA to emphasis in execution-time; it increases from 52 (with $\bar{k}_a$) to 86, or by 34.

5.4 Fast Fourier Transform Example

In this final example an input sequence x(n) is transformed from the discrete time domain to the discrete frequency domain X(m) using the fast Fourier transform (FFT) algorithm[57] of the discrete

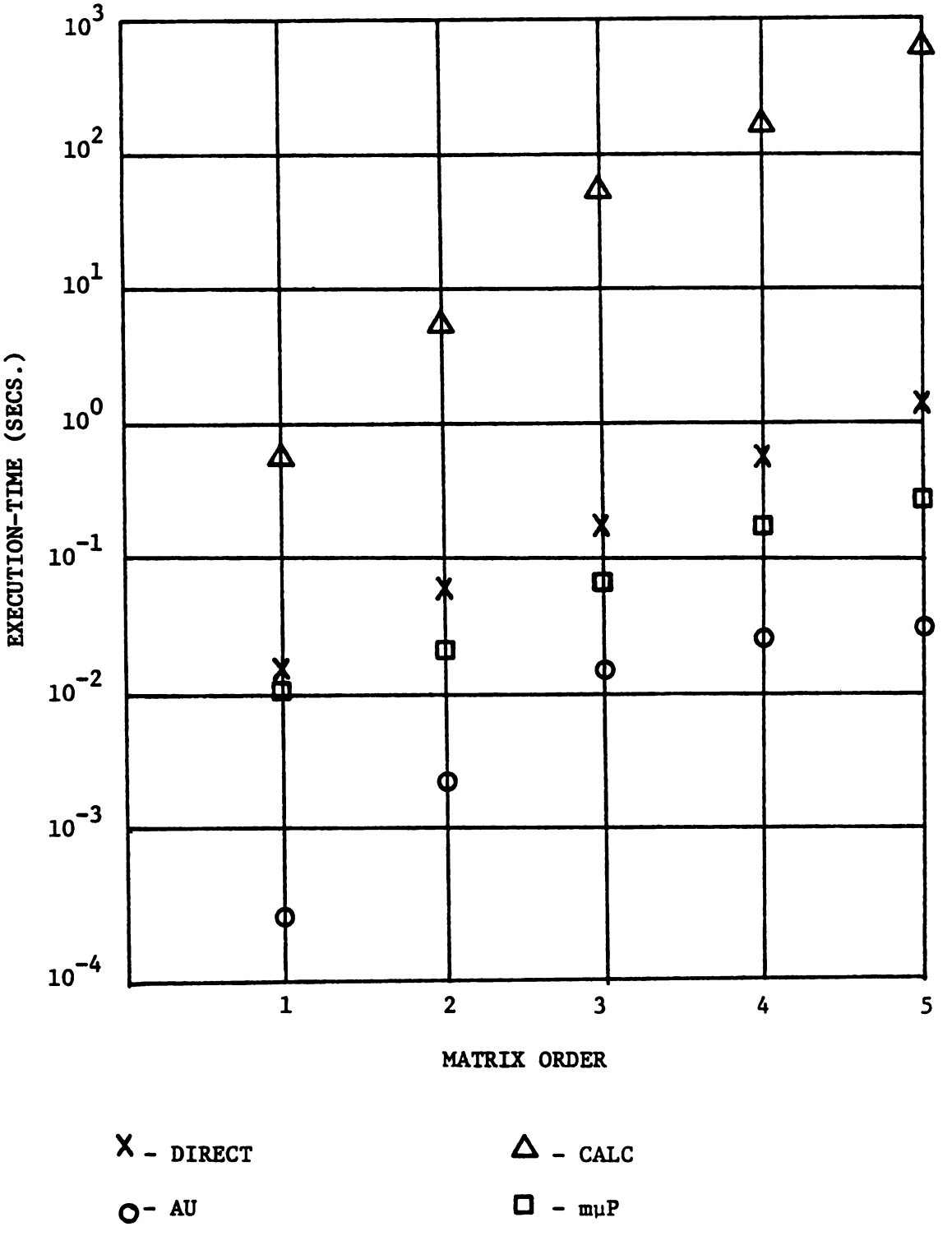


Figure 5.18 Single-precision execution-time versus matrix order for Example-2.

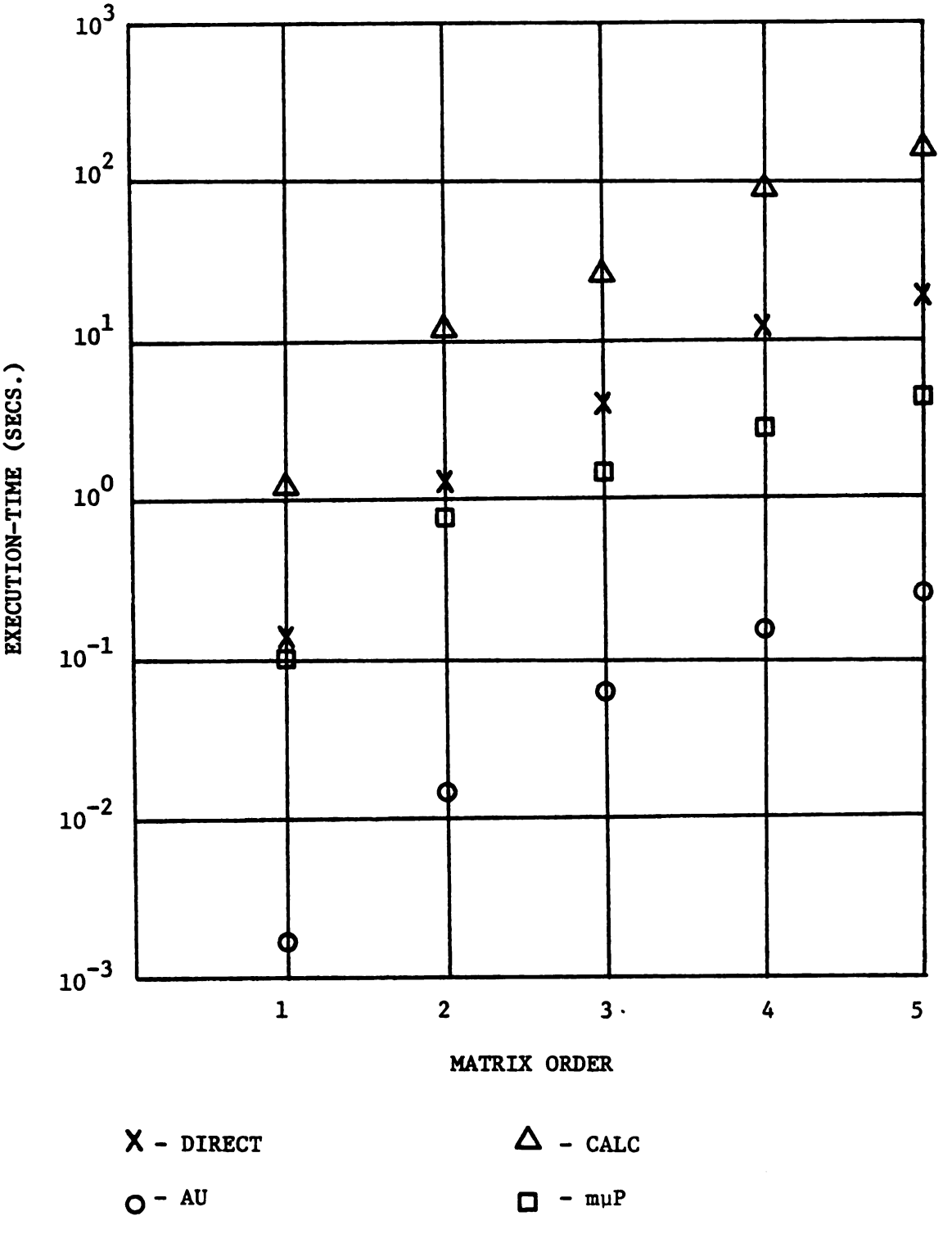


Figure 5.19 Double-precision execution-time versus matrix order for Example-2.

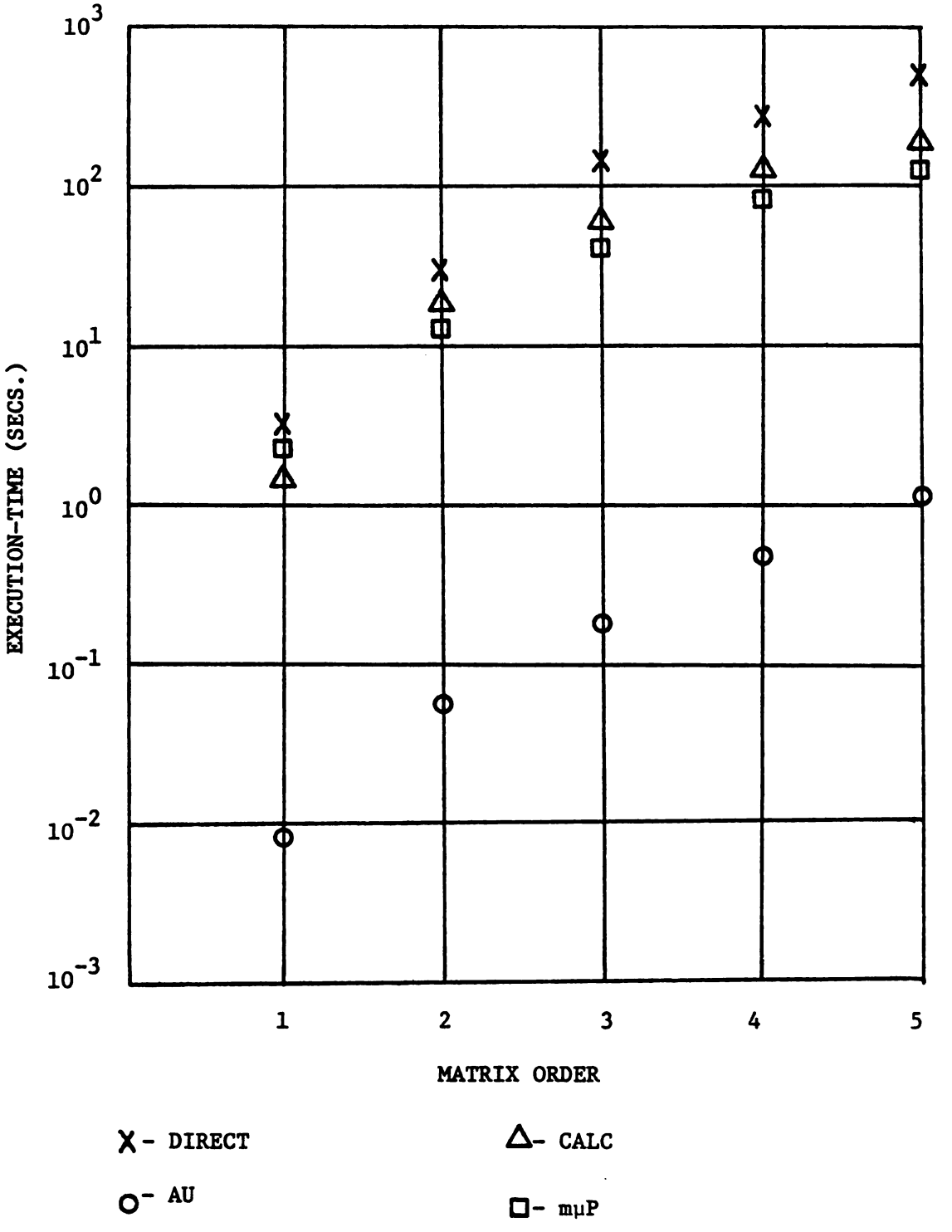


Figure 5.20 Triple-precision execution-time versus matrix order for Example-2.

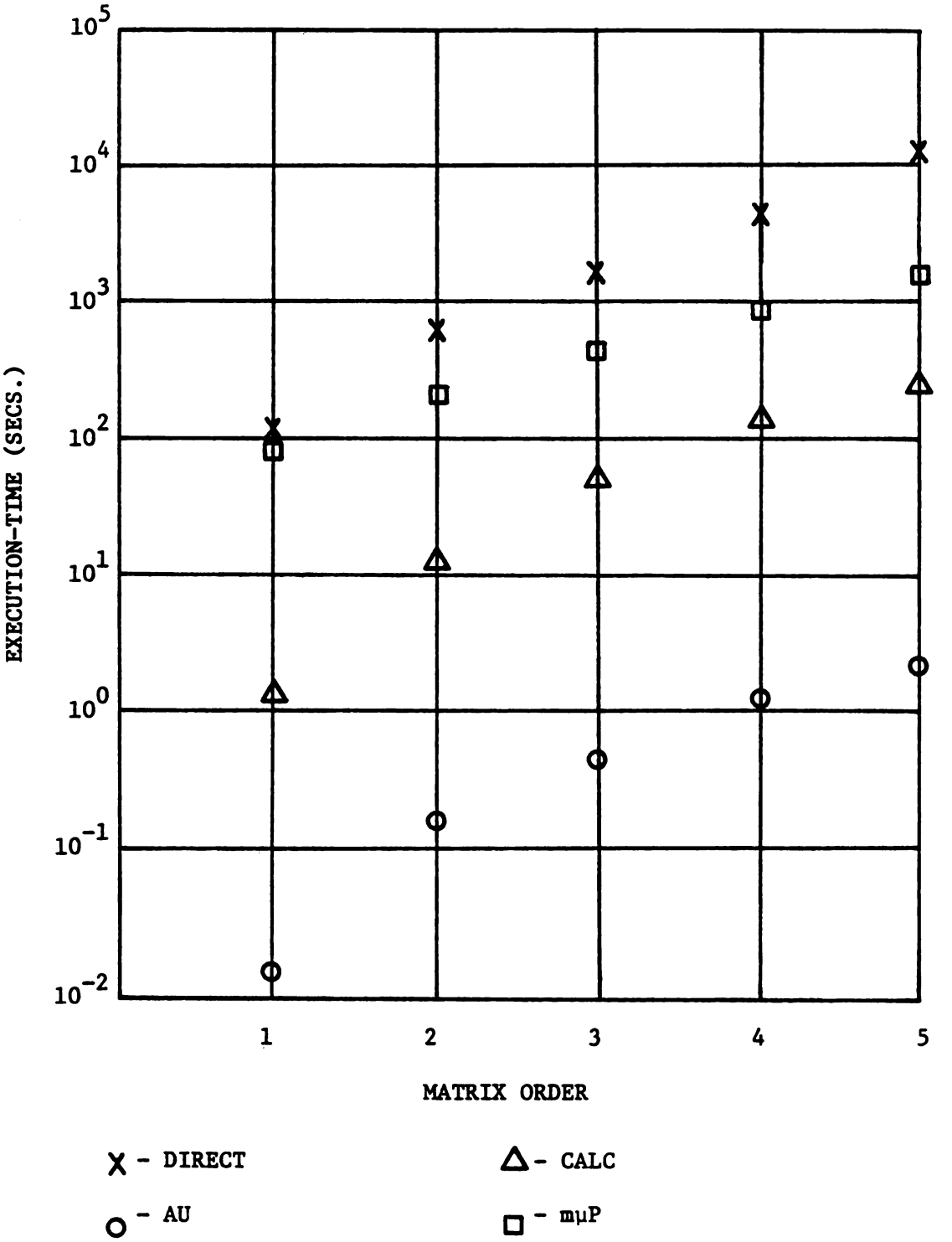


Figure 5.21 Quadruple-precision execution-time versus matrix order for Example-2.

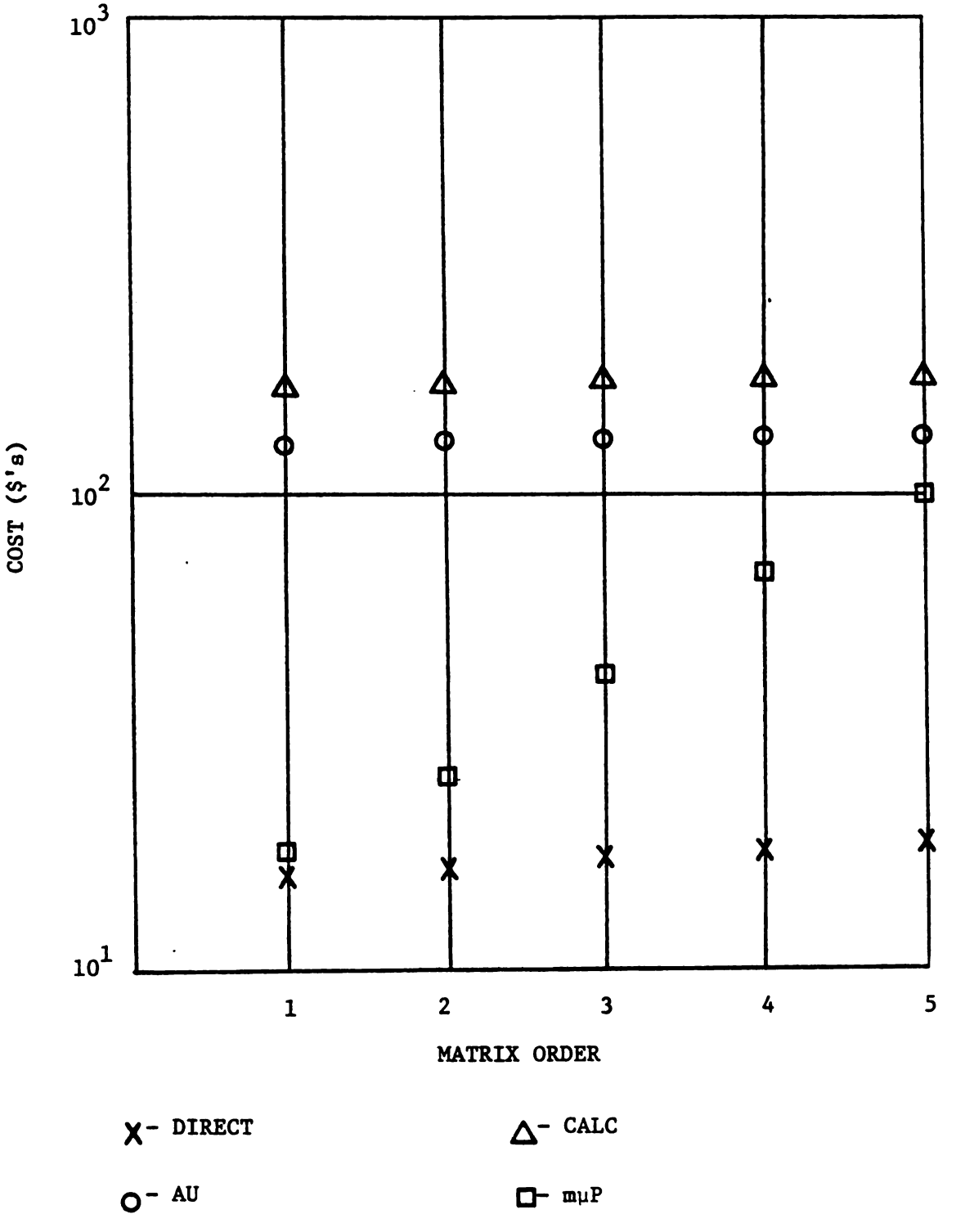


Figure 5.22 Single-precision costs versus matrix order for Example-2.

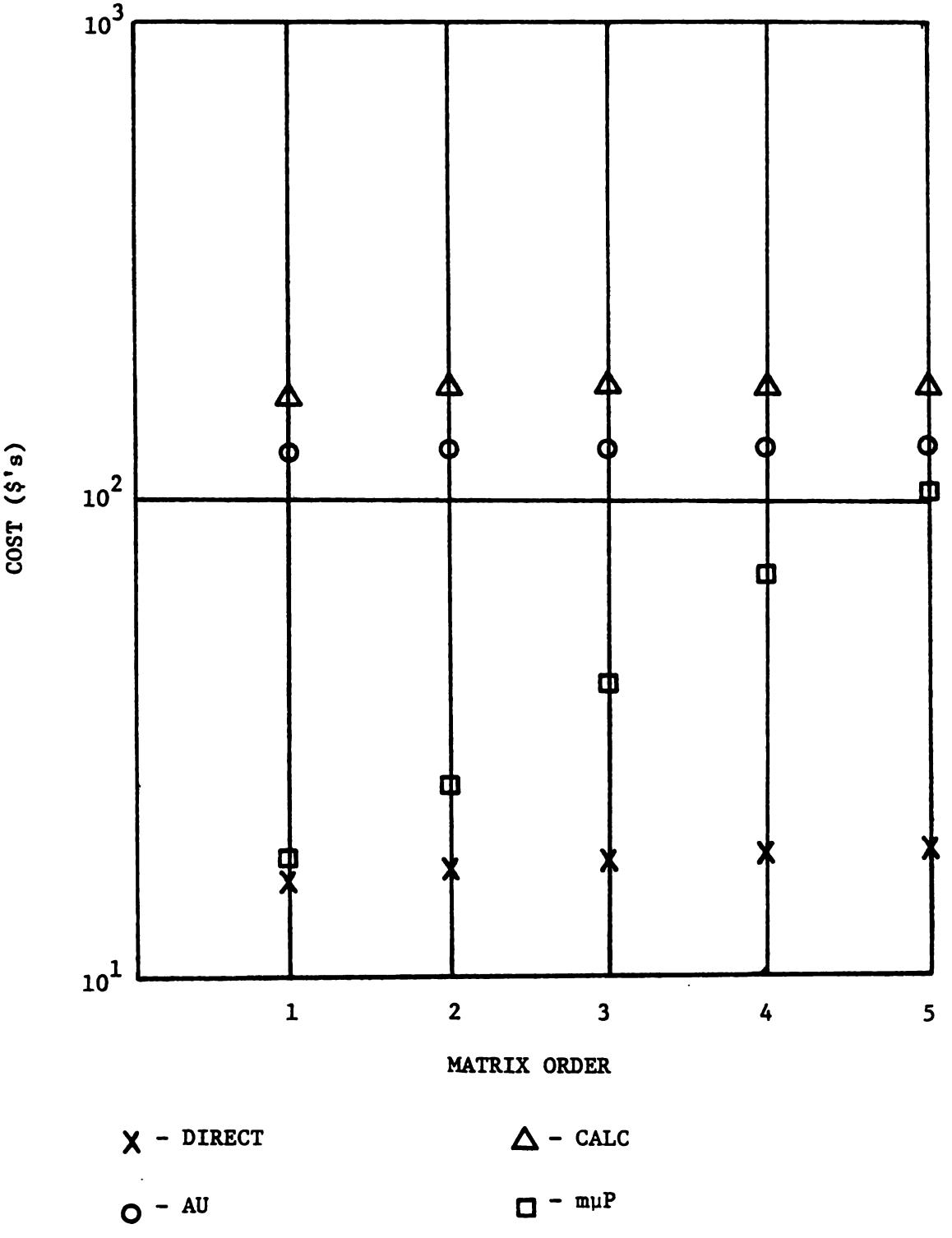


Figure 5.23 Double-precision cost versus matrix order for Example-2.

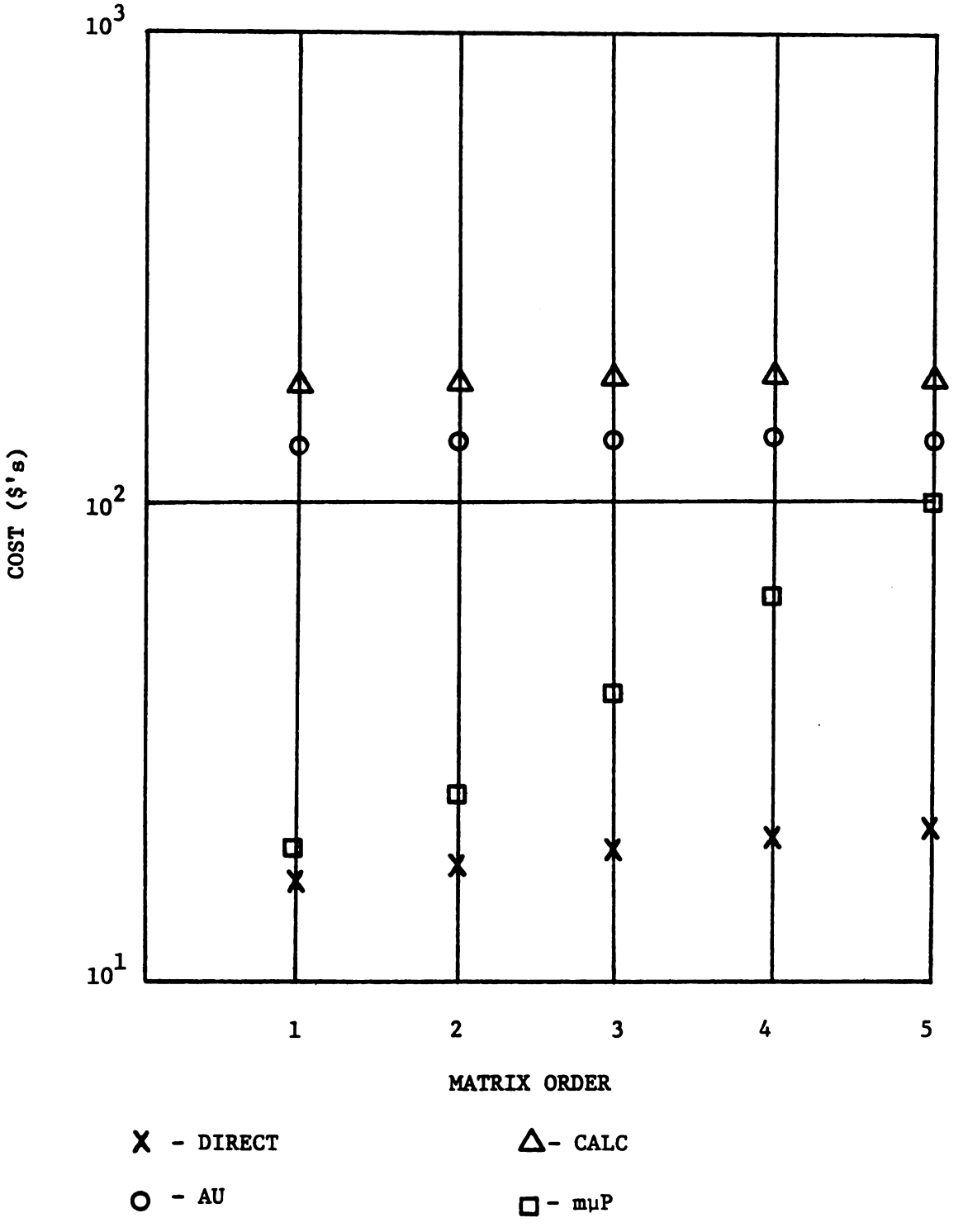


Figure 5.24 Triple-precision costs versus matrix order for Example-2.

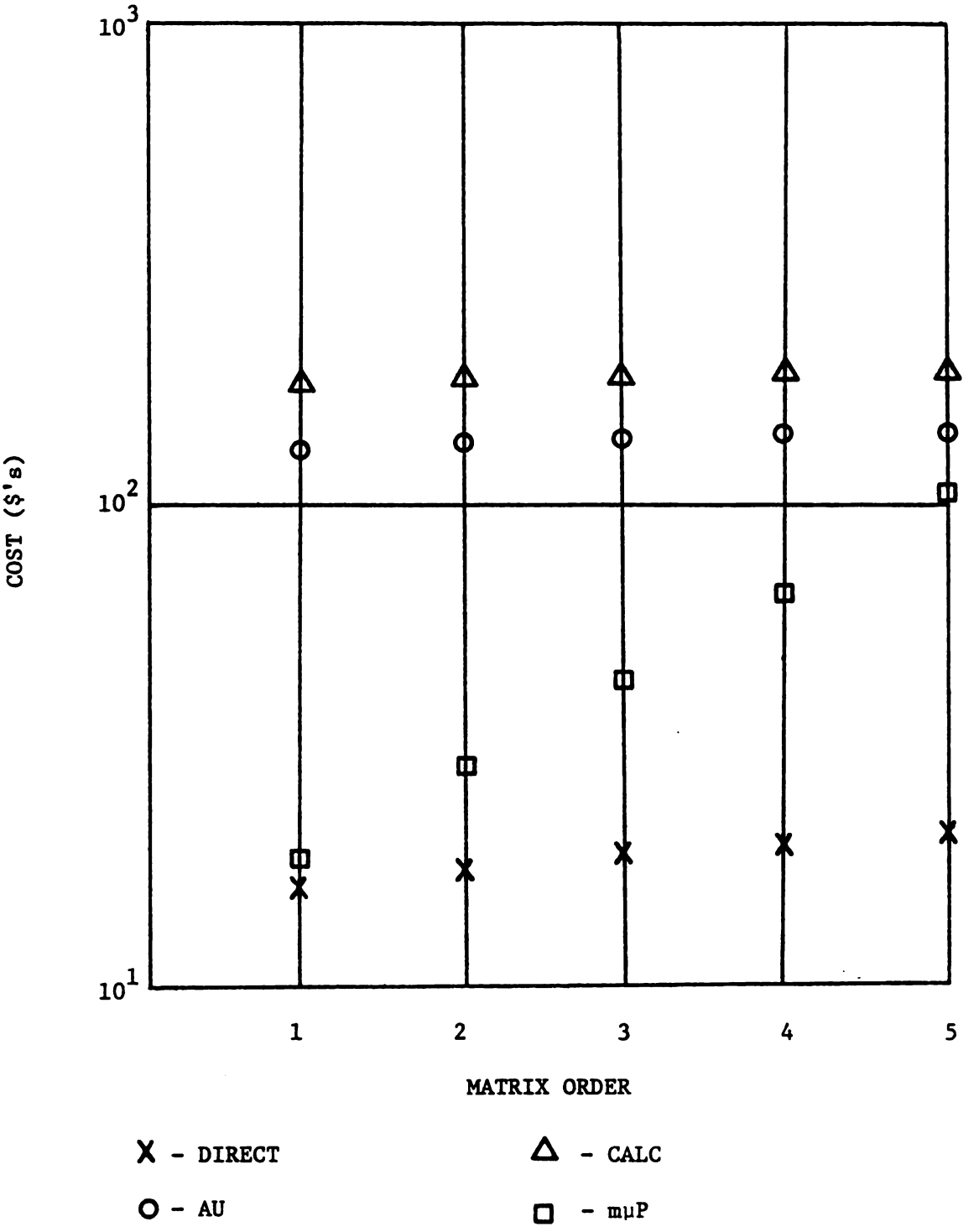


Figure 5.25 Quadruple-precision costs versus matrix order for Example-2.

Table 5.11 Typical consequence set and utilities for example-2(2nd order).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	4.19E1	00	55	63	16	87
	2.22E1	100				
	1	00				
2	3.84E-3	100	50	15	85	51
	1.25E2	51				
	4	100				
3	199E1	53	51	85	52	15
	2.31E2	00				
	2	33				
4	9.87E-1	98	74	45	91	86
	4.05E1	91				

Note: $\bar{k}_a = (0.3, 0.3, 0.3)$, $\bar{k}_1 = (0.8, 0.1, 0.1)$, $\bar{k}_2 = (0.1, 0.8, 0.1)$, $\bar{k}_3 = (0.1, 0.1, 0.8)$

Table 5.12 Typical consequence set and utilities for example-2(5th order).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	6.55E2	00	55	63	17	87
	3.22E1	100				
	1	00				
2	6.60E-2	100	52	16	86	54
	1.28E2	55				
	4	100				
3	3.71E2	43	48	84	45	14
	2.44E2	00				
	2	33				
4	6.19E0	99	67	43	89	67
	1.01E2	68				

Note: $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), $\bar{k}_3$ = (0.1, 0.1, 0.8)

Fourier transform (DFT). If it is assumed that the input sequence is real and even, then the DFT yields

$$X(m) = \sum_{n=0}^{N-1} x(n) \cos\{\frac{2\pi mn}{N}\}, \tag{5.9}$$

$$= \sum_{n=0}^{N-1} x(n)W^{mn}, \text{ where} \tag{5.10}$$

N = number of samples,

x(n) = input sequence, and

X(m) = transform sequence.

Using matrix notation Equation 5.10 becomes

$$\bar{X} = \bar{W}\,\bar{x};$$

but if $N = 2^L$, then $\bar{W}$ can be factored into L terms,

$$\bar{X} = (\prod_{i=1}^{L} \bar{W}_i)\,\bar{x}. \tag{5.11}$$

This feature greatly reduces the execution-time of the DFT by minimizing the number of multiplies. Figure 5.26 illustrates the series program flowchart.

The series flowchart contains three principal steps within three basic loops. Much of the algorithm deals with altering loop control variables which do not demand a large part of the total execution-time. Additionally, the process defined in Step-1 must find the weight coefficient W^{IE} which involves a cosine function as in Equation 5.9. A Maclaurin series expansion of the cosine yields

$$\cos X = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + Rn, \tag{5.12}$$

Rn = remainder.

Thus, Step-1 can use this expansion to determine the weight coefficient

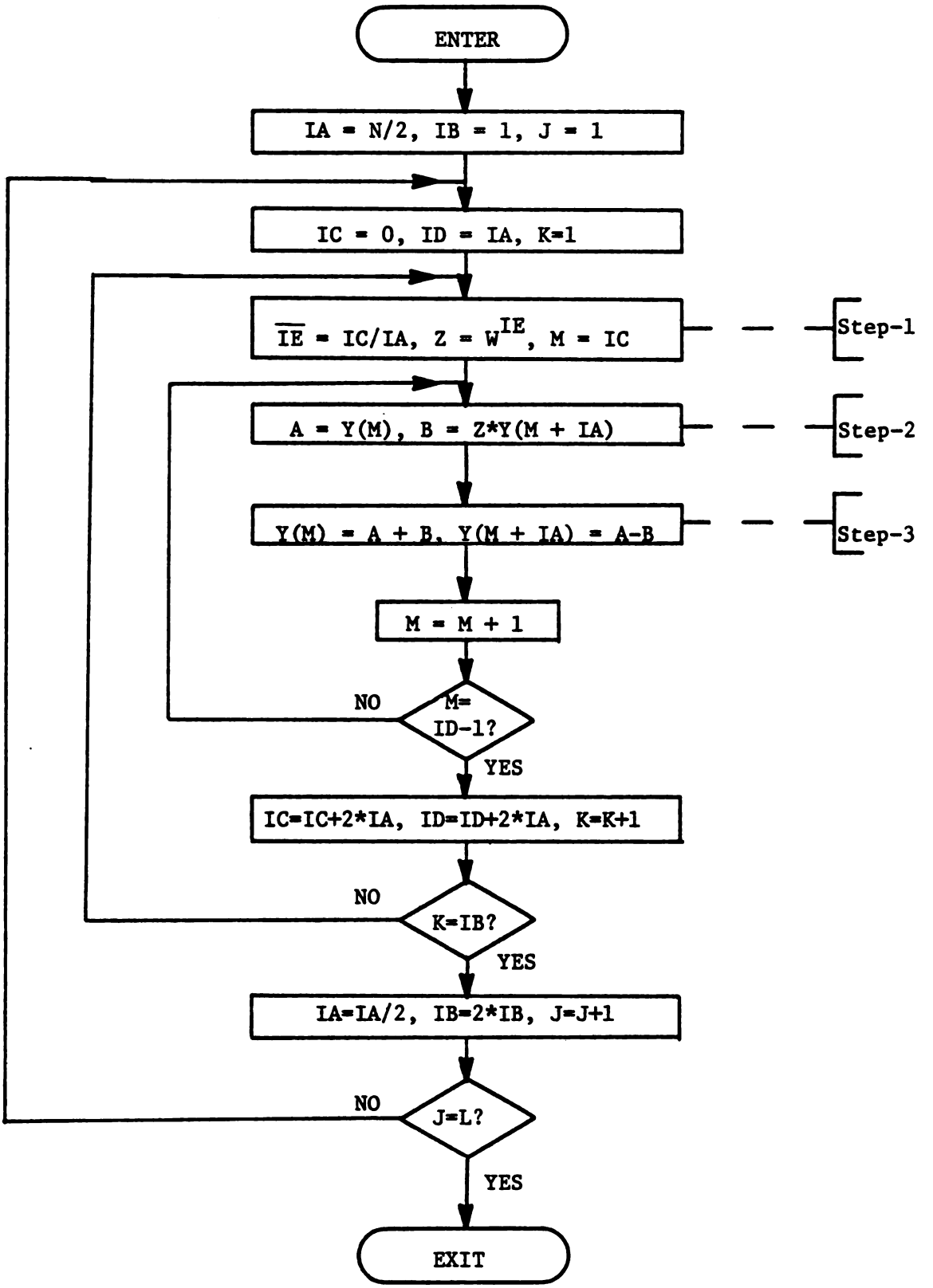


Figure 5.26 FFT series flowchart (Cooley, Tukey technique).[58]

W^{IE}. To complete the matrix multiplication of Equation 5.11, Steps 2 and 3 perform the required multiplies and adds. As with the other examples, the contributions of each step sum to give the total number of adds and multiplies (terms A_i and M_i). But since the CALC CDA contains the cosine function in hardware, the contributions of Step-1 (or Eq. 5.12) reduces to a simpler form. Table 5.13 outlines the steps and their contribution for A_i, M_i; i = 1, 2. When multiplied by the multiple-precision add and multiply times these terms produce

$$T_s = T_i = [2N\log_2 N + 6(N - 1)]a_i + [N\log_2 N + 18(N - 1)]m_i;\ i = 1, 2. \quad (5.14)$$

Equation 5.14 agrees with published results.[59] Similarly, Table 5.14 depicts the estimates of A_3 and M_3 which give

$$T_s' = T_3 = [2N\log_2 N + 27.2(N - 1)]a_3 + [N\log_2 N]m_3. \quad (5.15)$$

By examining Figure 5.26 it is possible to determine those sections of the series flowchart that can be executed simultaneously, or in parallel. Since Steps 2 and 3 only require the weight W^{IE}, N slaves can perform these tasks and the Master flowchart now contains far less multiplies (see Fig. 5.27 and 5.28). Again, to find the total number of adds and multiplies sum the contribution of each step in the parallel flowchart as detailed in Table 5.15. (Note: for an estimate of the "SLAVES BUSY" term use Fig. 5.29) Multiplying these results by the multiple-precision add and multiply times yields the parallel execution-time

$$T_p = T_4 = [(6N + 1)\log_2 N + 6(N - 1)]a_4 + [\log_2 N + 18(N - 1)]m_4 \quad (5.16)$$

Table 5.13 Estimates of A_i, M_i (i = 1, 2) for Example-3.

Step	Adds	Multiplies
1	$6(N - 1)$	$18(N - 1)$
2	--	$N\log_2 N$
3	$2N\log_2 N$	--
Totals	$A_i = 2N\log_2 N + 6(N - 1)$	$M_i = N\log_2 N + 18(N - 1)$

Table 5.14 Estimates of A_3, M_3 for Example-3.

Step	Adds	Multiplies
1	$27.2(N - 1)$	--
2	--	$N\log_2 N$
3	$2N\log_2 N$	--
Totals	$A_3 = 2N\log_2 N + 27.2(N - 1)$	$M_3 = N\log_2 N$

Table 5.15 Estimates of A_4, M_4 for Example-3.

Step	Adds	Multiplies
1	$6(N - 1)$	$18(N - 1)$
2	$2N\log_2 N$	--
3	$4N\log_2 N$	--
4	$\log_2 N$	$\log_2 N$
Totals	$A_4 = (6N + 1)\log_2 N + 6(N - 1)$	$M_4 = \log_2 N + 18(N - 1)$

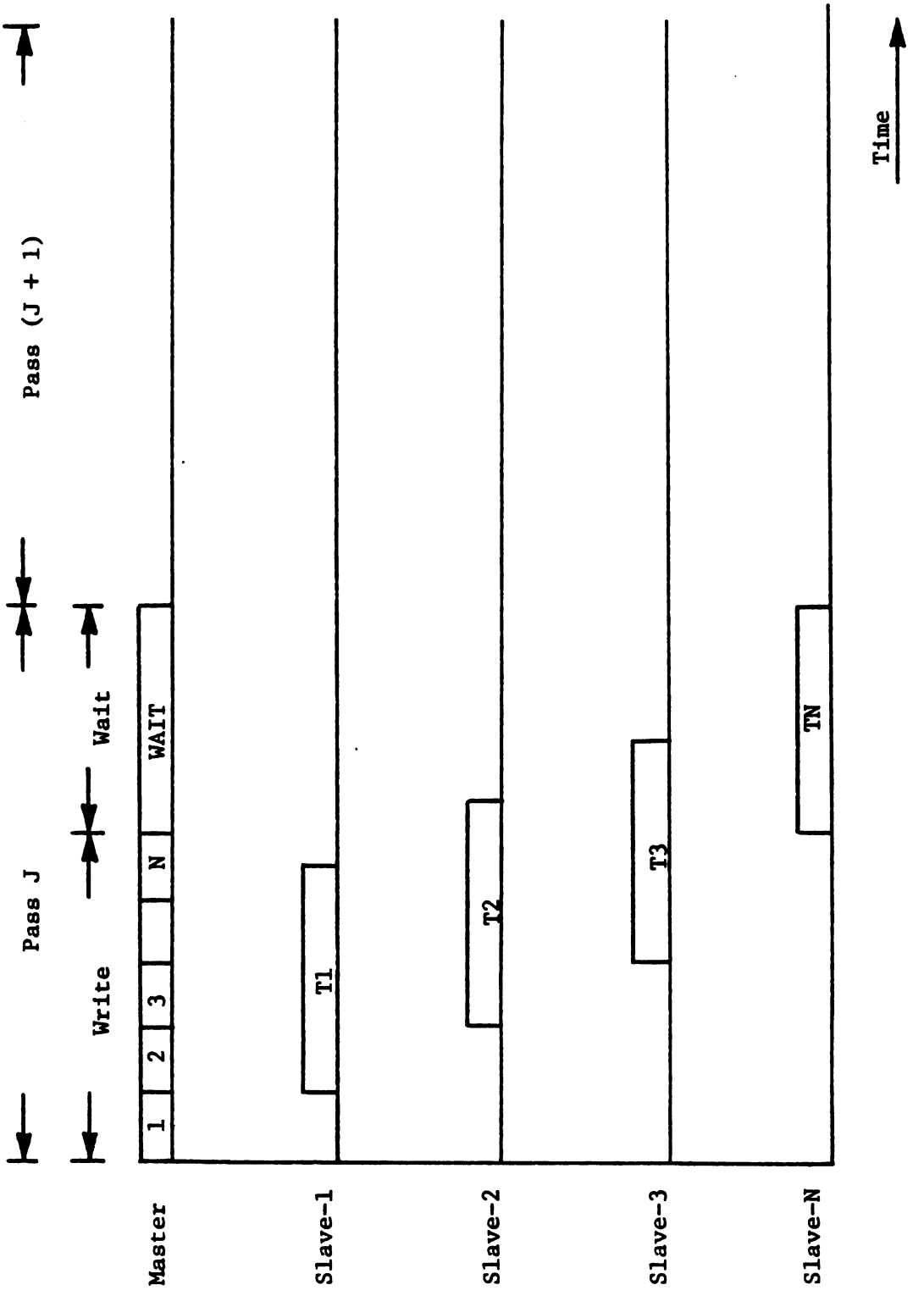


Figure 5.27 Relation between Master and Slaves for example-3

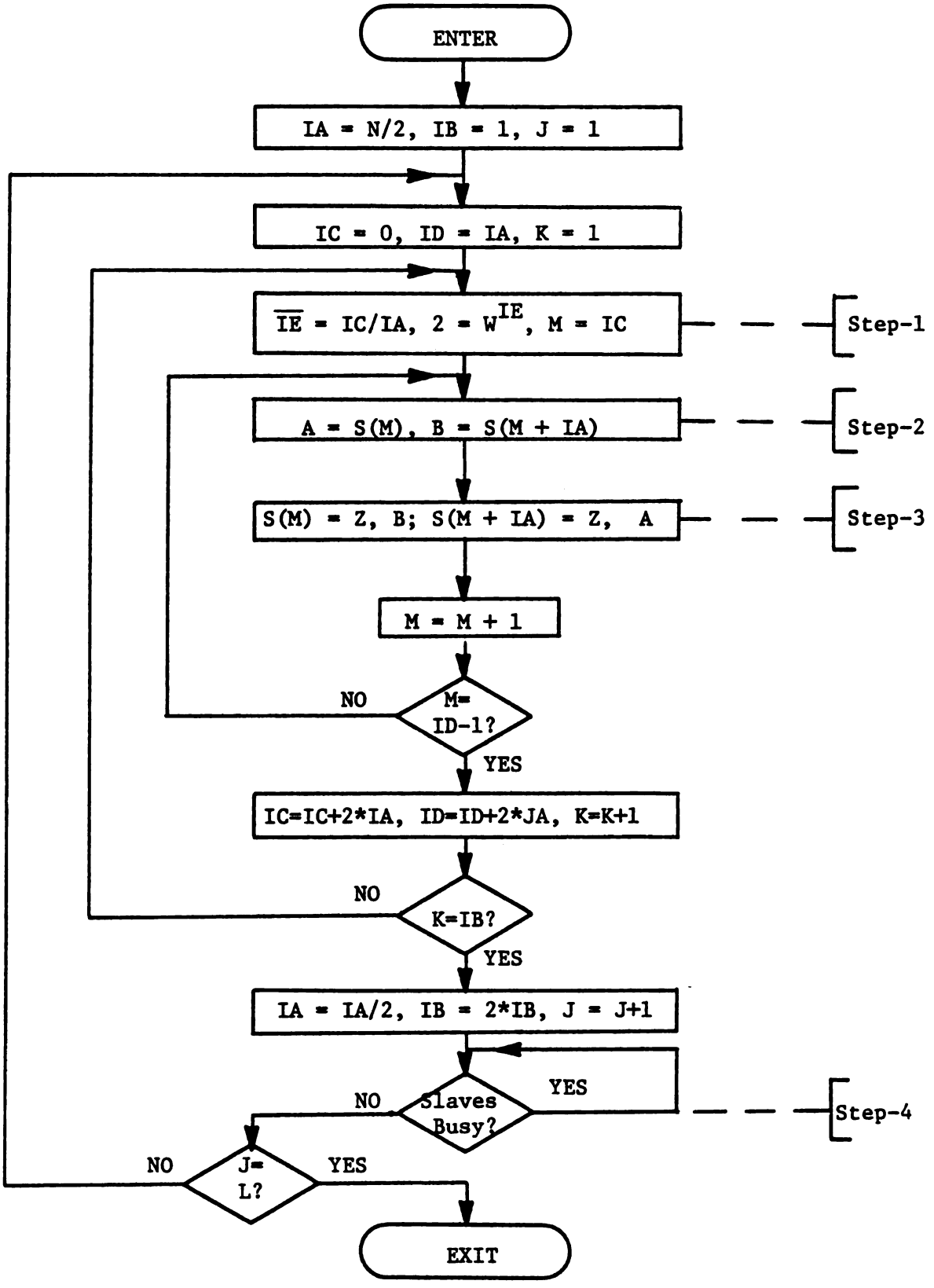


Figure 5.28 FFT parallel flowchart: Master.

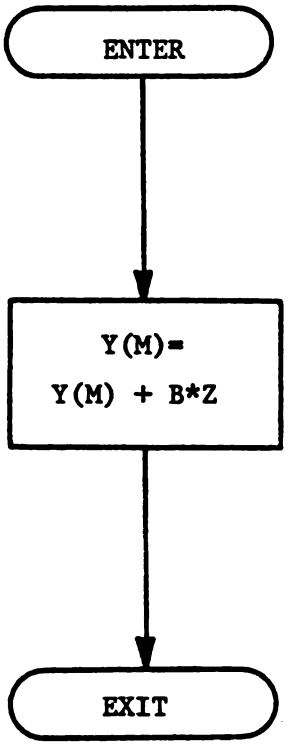


Figure 5.29 FFT parallel flowchart: Slave M.

When the ratio of series to parallel execution-times is determined, it gives an indication of the decrease in execution-time due to simultaneous execution. Using Equations 5.14 and 5.16 with the assumption $m_i = 8a_i$

$$R = \frac{T_s}{T_p} = \frac{[2N\log_2 N + 6(N-1)]a_i + [N\log_2 N + 18(N-1)](8a_i)}{[(6N+1)\log_2 N + 6(N-1)]a_i + [(\log_2 N + 18(N-1)](8a_i)},$$

$$= \frac{10N\log_2 N + 150(N-1)}{(6N+9)\log_2 N + 150(N-1)}.$$

For large N,

$$\lim_{N \to \infty} R = \frac{10}{6} = 1.66,$$

the execution-time decreases, at best, by a factor of 1.66. Like Example-1, parallel execution does not reduce the execution-time considerably because the Master's overhead demands much time. But if it did not calculate weight coefficients, i.e., the number of samples is known a priori; then the Master's overhead would lessen for a larger R.

When determining the application memory cost, C_{ai}, all subroutine calls (add/subtract and multiply/divide) require four words of program storage and each loop demands an additional four words of program storage. As with execution-time, the program flowcharts (Fig. 5.26 through 5.29) assist in estimating the application memory cost. Similarly, the contributions of each flowchart step add together to yield the final result. Table 5.16 outlines the steps and their contribution for the series flowchart; these terms combine as follows:

$$C_{ai} = W_o[44 + p(N+8)]; \; i \neq 4. \tag{5.17}$$

But for the mμP CDA (i = 4) the N slaves create additional program and data storage requirements. When added to the Master's terms (see Table 5.17) they sum to give the mμP application memory

Table 5.16 Estimates of P_i, D_i (i = 4) for Example-3.

Step	Program	Data
1	24	N + 8
2	12	--
3	8	--
Totals	$P_i = 44$	$D_i = N + 8$

Table 5.17 Estimates of P_4, D_4 for Example-3.

Step	Program	Data
1	24	8
2	4	--
3	4	--
4	--	--
Slaves (N)	8N	N
Totals	$P_i = 8(N + 4)$	$D_i = N + 8$

$$C_{a4} = W_o[(32 + 8N) + p(N + 8)]. \quad (5.18)$$

Lastly, the results of Chapter III and Equations 5.14 through 5.18 combine to produce the set of attribute values for this example (see Fig. 5.30 through 5.37). Using the techniques of Chapter IV, these values are converted to marginal utilities and, moreover, to utilities as shown in Tables 5.18 and 5.19.

As with the other examples, these figures and tables illustrate several points. Again, the AU CDA displays the fastest speed and the mμP always exhibits quicker execution-time than the DIRECT CDA. And for $N = 2^7$, R = 1.6 which agrees well with the prediction. Similarly, the DIRECT/mμP CDA's lag behind the CALC CDA in speed for p = 1, 2; still it surpasses them for p = 3, 4. With respect to costs, the mμP CDA begins high and for modest N ends up extremely high due to all the additional Slaves (term C_{e4}). Costs for the DIRECT CDA climb faster than the AU and CALC CDA's because the application memory term (C_{a1}) begins to dominate the elemental cost term (C_{e1}). In Table 5.18 for weight vector $\bar{k}_a$, the DIRECT and CALC CDA's correspond to "break-even" utilities; i.e., 55 and 56, respectively. Also, for the typical consequence set in Table 5.19 the CALC CDA depicts the overall, "best" CDA ($\bar{k} = \bar{k}_a$) by quite a large score (by 20 utility values). This results from the large expense of the mμP CDA and large problem dimensions (2^7 samples). Finally, Table 5.18 illustrates the "sensitivity" of the DIRECT CDA to the cost attribute; as $\bar{k}$ changes from $\bar{k}_a$ to $\bar{k}_3$, the DIRECT CDA's utilities vary from 55 to 87, or by 32.

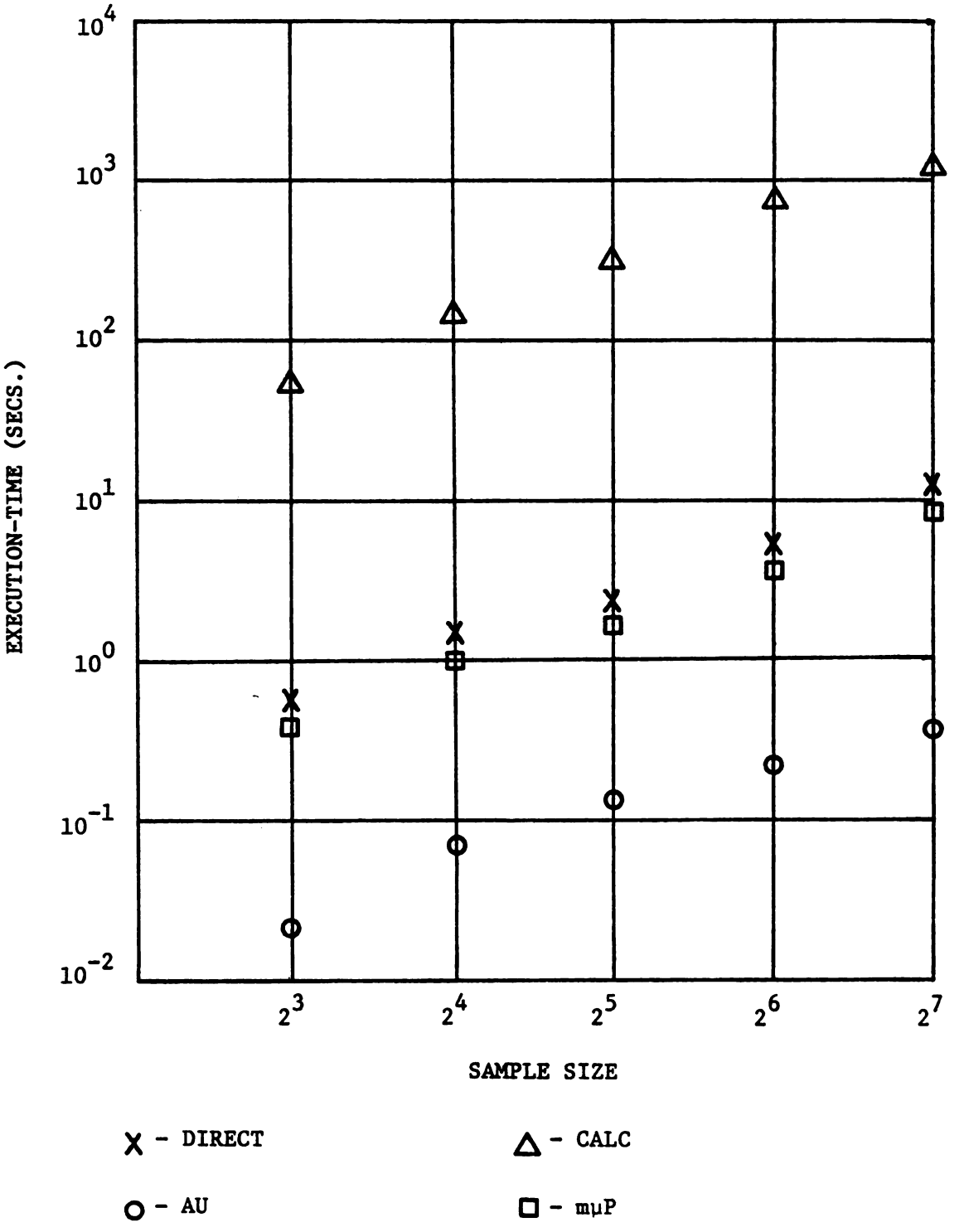


Figure 5.30 Single-precision execution-time versus sample size for Example-3.

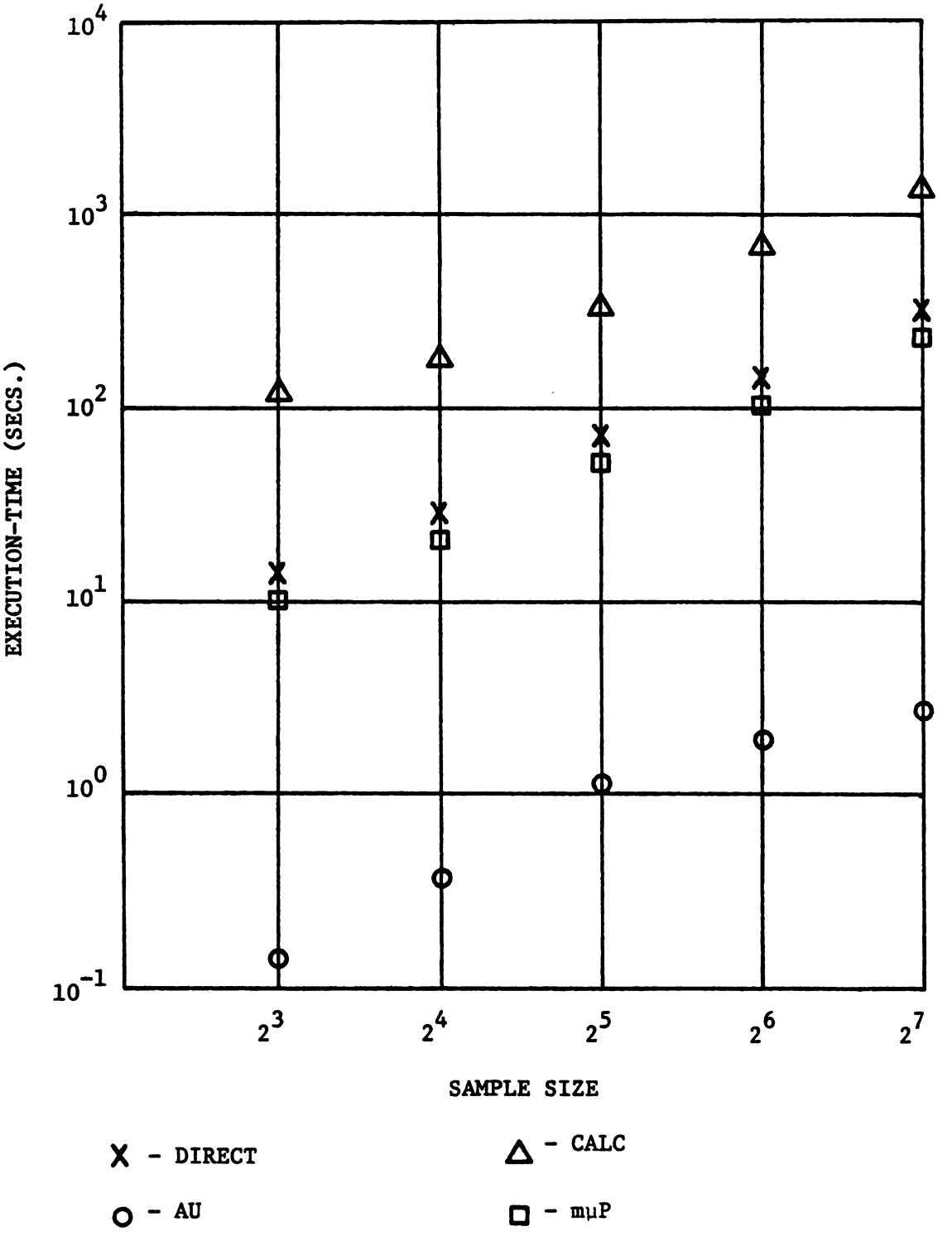


Figure 5.31 Double-precision execution-time versus sample-size for Example-3.

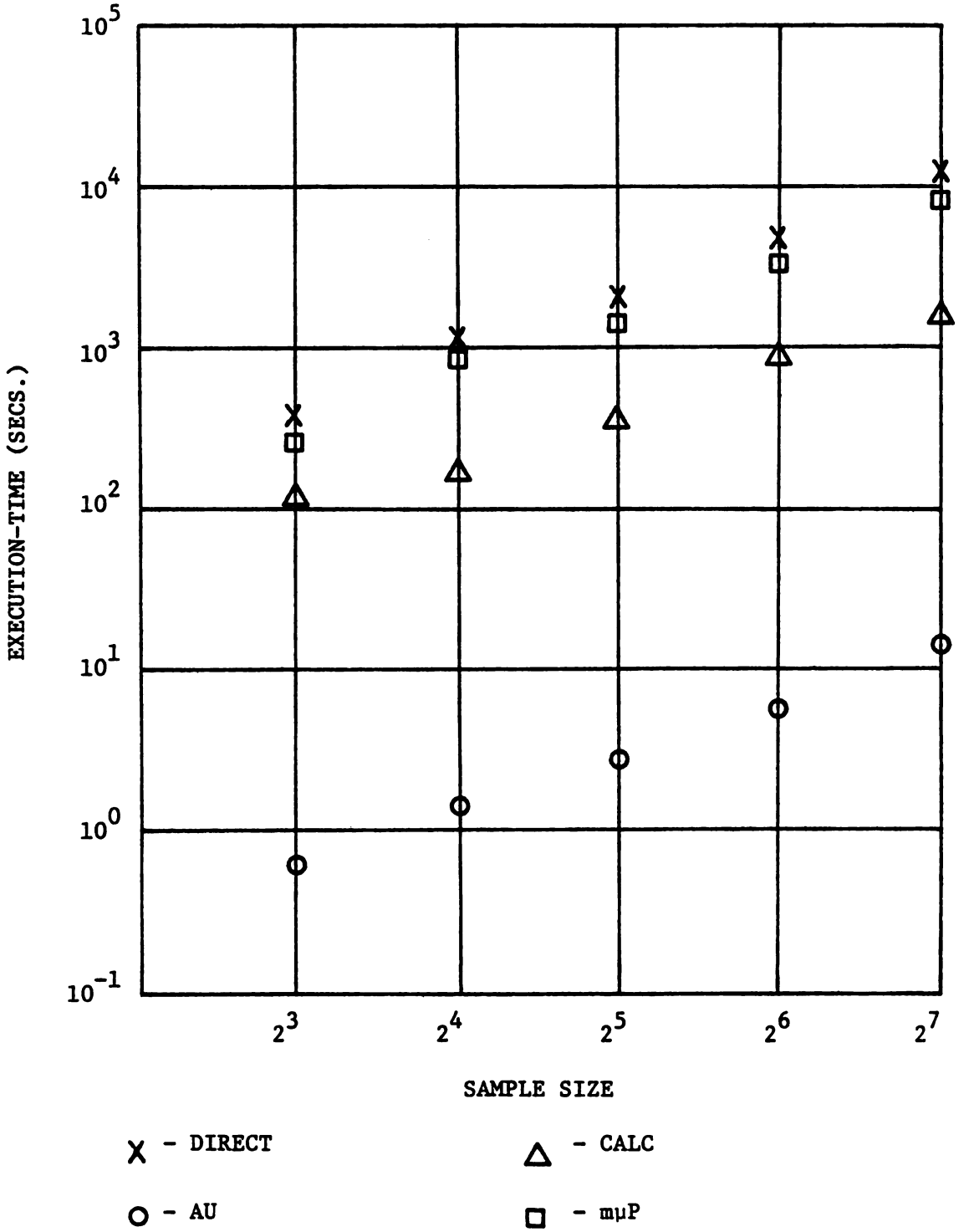


Figure 5.32 Triple-precision execution-time versus sample size for Example-3.

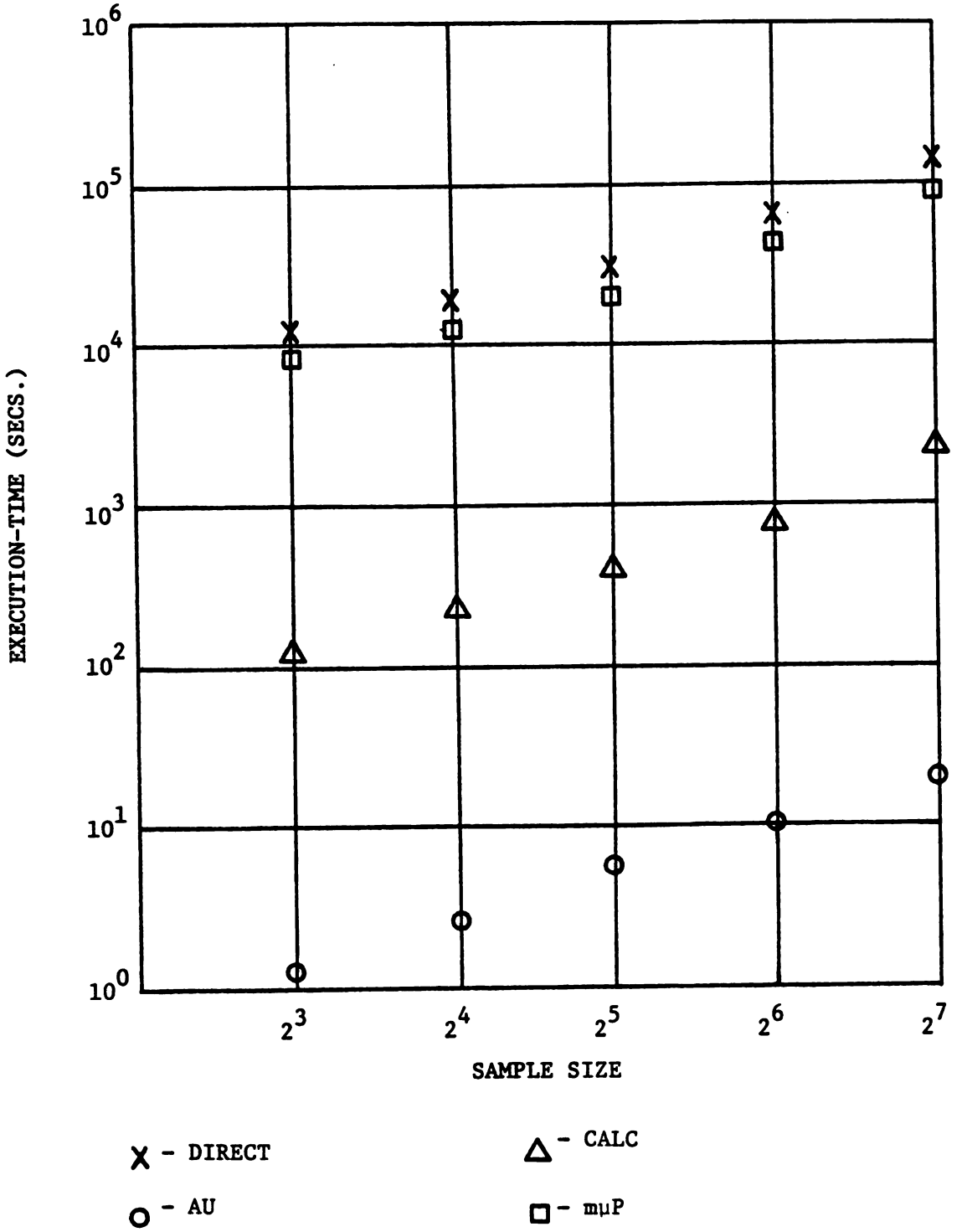


Figure 5.33 Quadruple-precision execution-time versus sample size for Example-3.

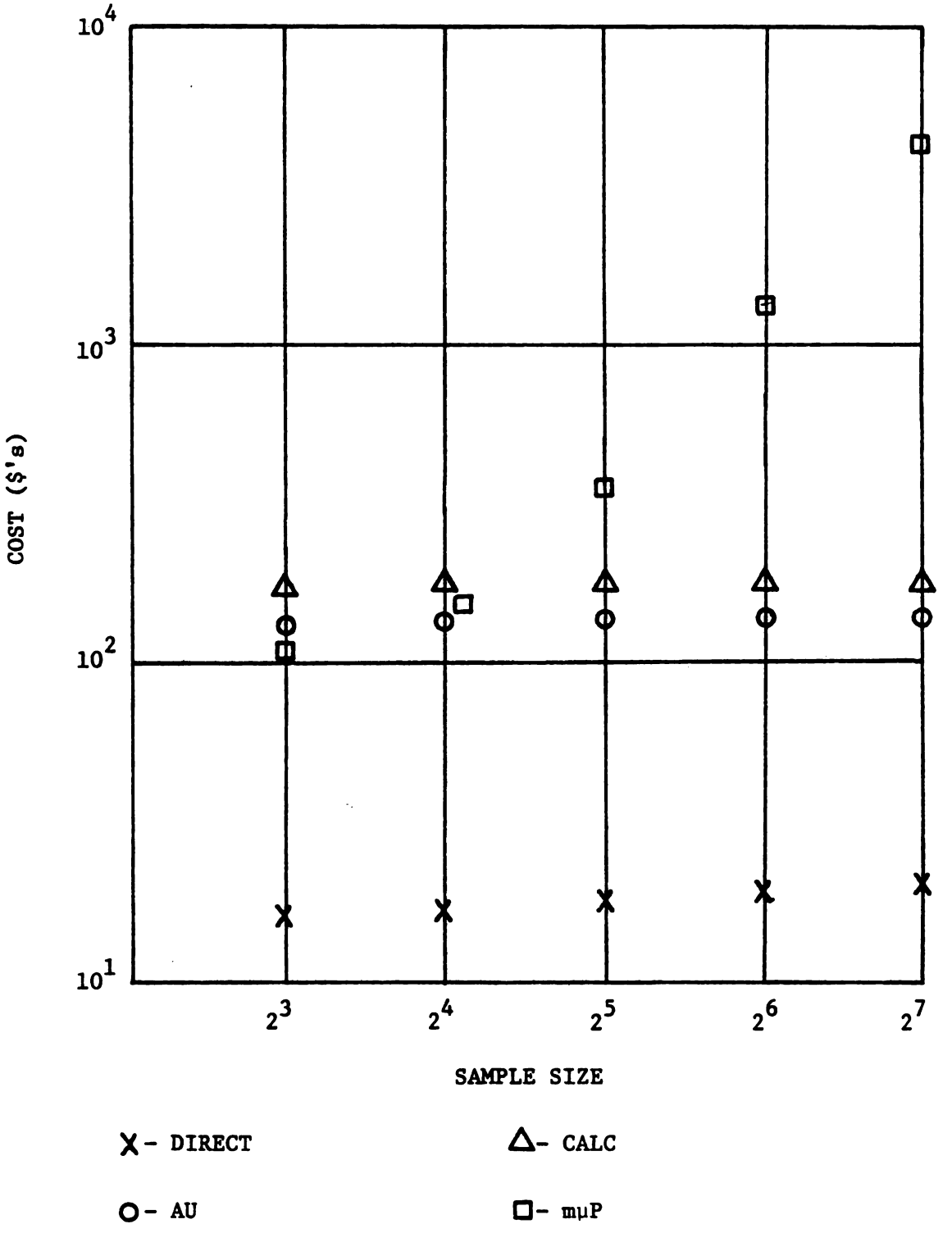


Figure 5.34 Single-precision costs versus sample size for Example-3.

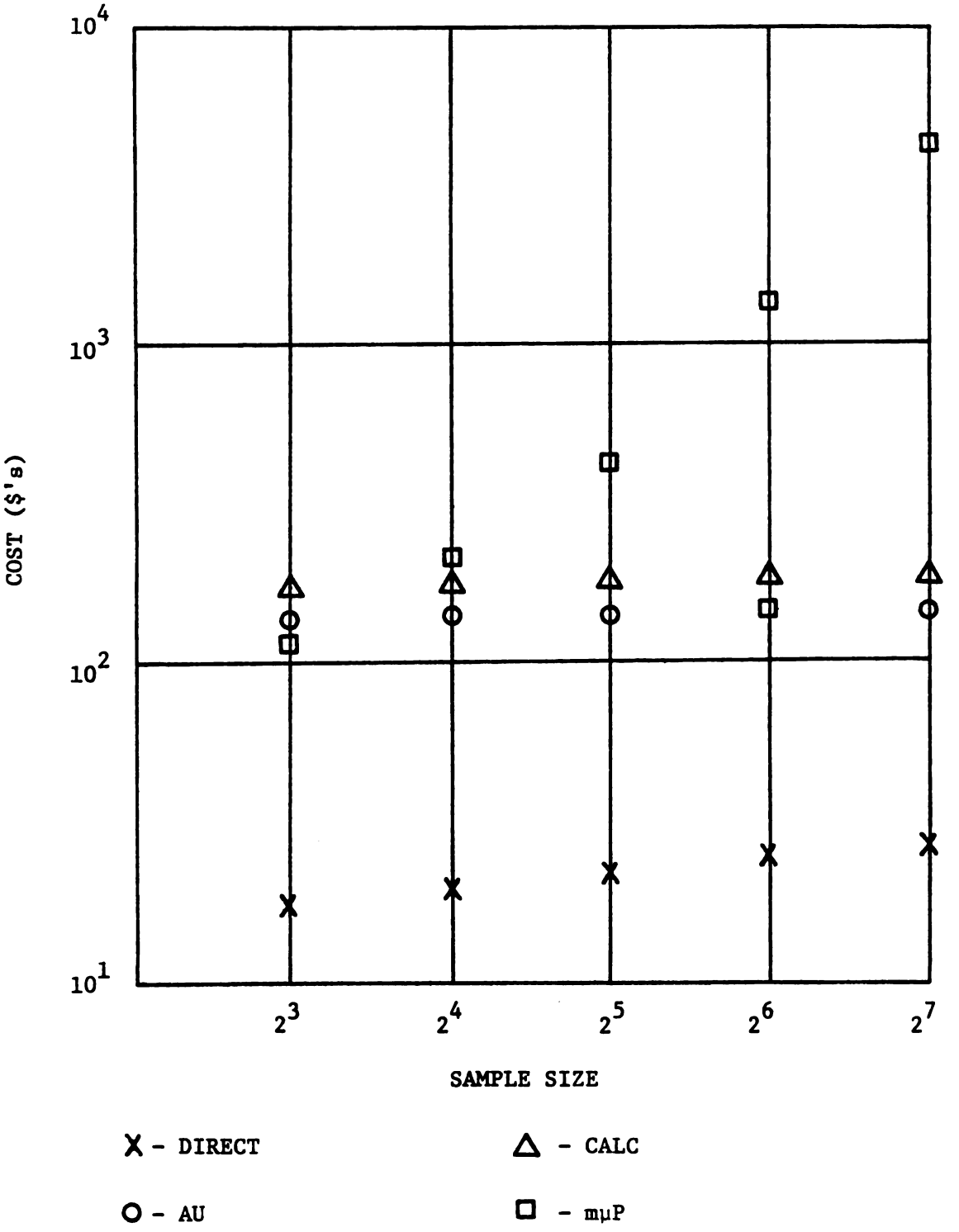


Figure 5.35 Double-precision costs versus sample size for Example-3.

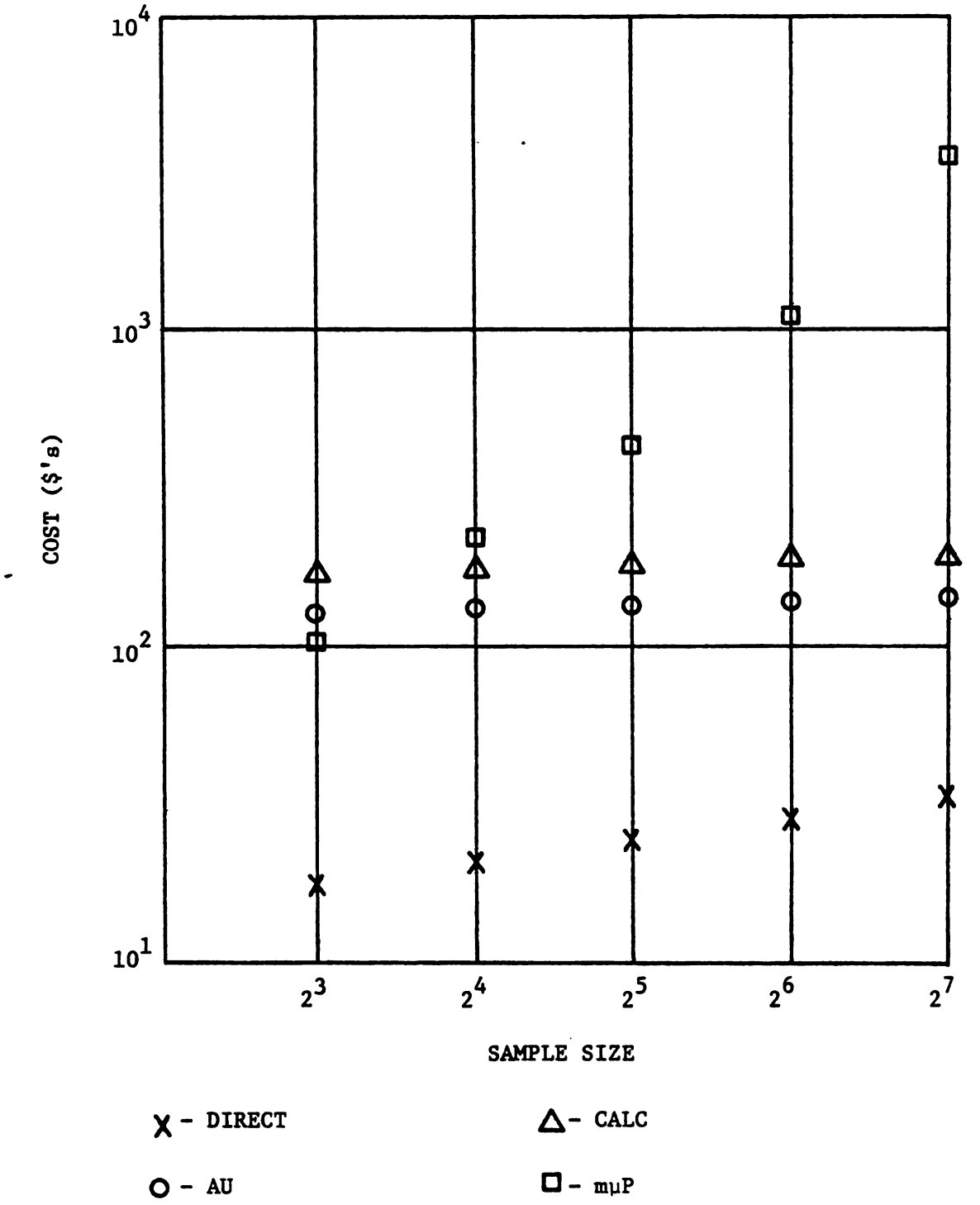


Figure 5.36 Triple-precision costs versus sample size for Example-3.

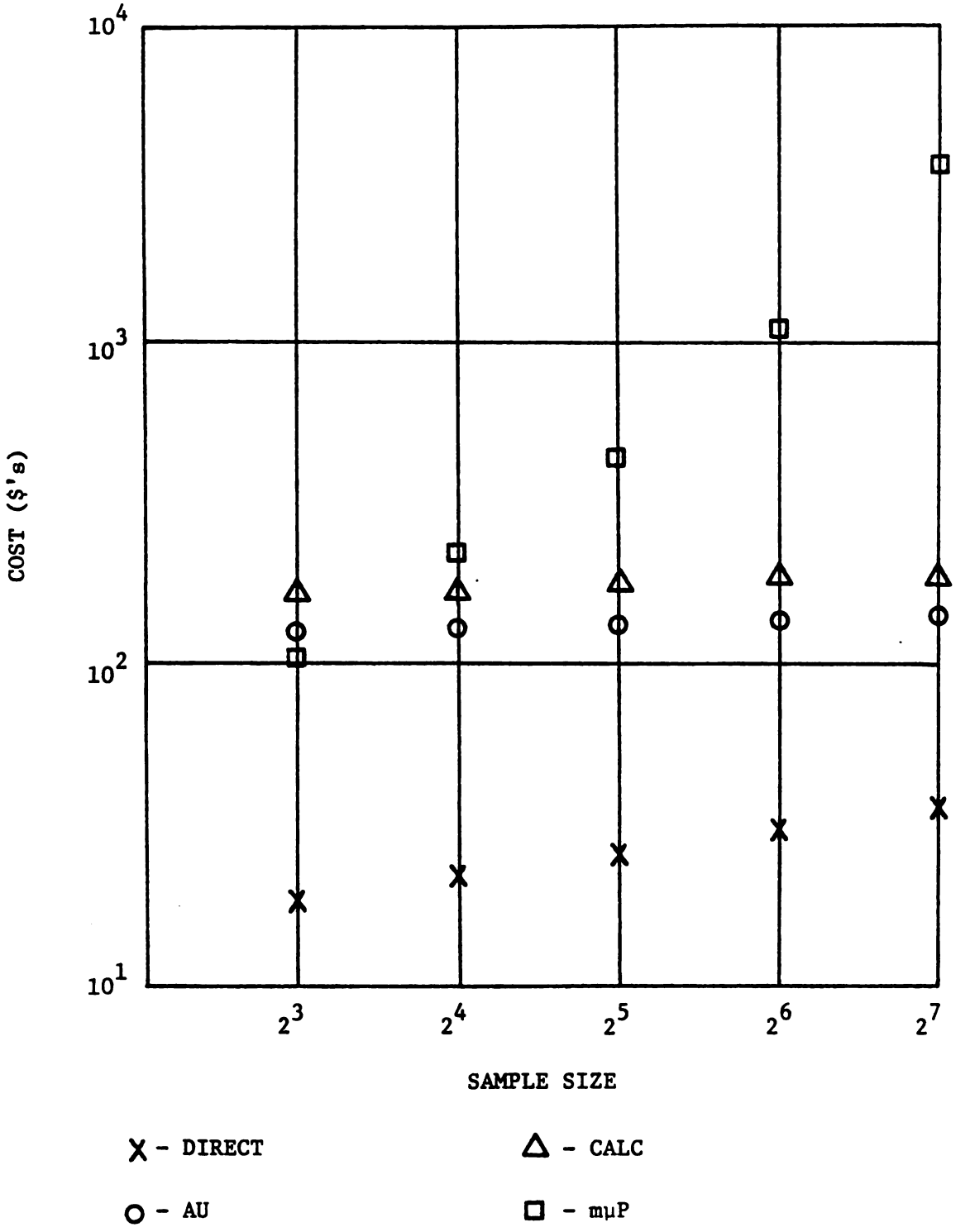


Figure 5.37 Quadruple-precision costs versus sample size for Example-3.

Table 5.18 Typical consequence set and utilities for example-3(2^3 samples).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	5.90E2	00	55	63	17	87
	2.54E1	100				
	1	00				
2	3.72E-2	100	50	15	85	50
	1.27E2	50				
	4	100				
3	1.91E2	68	56	87	64	17
	2.28E2	00				
	2	33				
4	2.12E1	96	52	39	83	35
	1.72E2	28				

Note: $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), $\bar{k}_3$ = (0.1, 0.1, 0.8)

Table 5.19 Typical consequence set and utilities for example-3(2^7 samples).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	1.25E4	00	55	63	17	87
	5.42E1	100				
	1	00				
2	8.41E-1	100	66	20	90	87
	1.36E2	97				
	4	100				
3	4.20E3	66	86	96	72	90
	2.66E2	91				
	2	33				
4	3.77E2	97	43	36	81	13
	2.43E3	00				

Note: $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), $\bar{k}_3$ = (0.1, 0.1, 0.8)

5.5 Additional Design Considerations

For the previous three examples, the overall results depend on both the specific software algorithms used and the hardware devices and architectures selected. Because the research project reported here considers only nominal values for the above, the overall results of these examples may shift dramatically with alternate approaches. But design engineers need only combine their specific approaches with the methodology to discover the changes. In this section, an alternative software algorithm illustrates this procedure.

Often, several algorithms may implement a specific task and each displays widely differing characteristics; e.g., execution-time, memory demand, hardware overhead, etc.[60] Consider the effect of an improved multiply algorithm to reduce execution-time for the DIRECT and mμP CDA's (terms m_1 and m_4). Only a reduction in add times or the number of adds will decrease multiply times, and since a reduced add time only requires a faster μP, the standard "shift-and-add" algorithm improves the number of adds.[61] For a p-precision number this requires approximately 8p shifts and an average of 4p, 2p-precision adds. Using Equation 3.2 as the 2p-precision add time and assuming that the p-precision shifts require p cycle-times,

$$m_1' = 4p(20pt_o) + 8p(pt_o)$$

$$= 88p^2t_o, \text{ and} \tag{5.19}$$

$$m_4' = 88p^2t_o. \tag{5.20}$$

These equations reveal a dramatic improvement in the multiple-precision

multiply times for the DIRECT and mμP CDA's as shown in Figure 5.38. Here, the decrease varies with precision; i.e., the greater the precision, the greater the improvement. And by p = 5 the DIRECT CDA actually executes faster than the AU CDA.

To illustrate the overall improvement in execution-time with an application, the above equations used with Equations 5.5 and 5.6 form the series and parallel execution-times for matrix inversion (see Fig. 5.39). This figure demonstrates a decrease in execution-time by roughly three orders of magnitude for triple-precision. Finally, the techniques of Chapter IV convert these attribute values to marginal utilities and utilities as depicted in Table 5.20. Comparison of this with Table 5.11 (the original table) shows great increase in the DIRECT and mμP CDA's marginal utilities due to improved execution-time. Consequently, the marginal utilities of the CALC CDA drop because it now possesses the slowest execution-time. As before, the "best" CDA varies as the weight vector $\bar{k}$ changes.

5.6 Summary

From careful analysis of the attribute values for the preceding three examples, several trends and conclusions can be deduced. With respect to execution-time, the AU CDA always performs the task the fastest because it contains a hardware multiply and divide. But the results of Chapter III predicted this would occur.

Next, the mμP CDA invariably surpasses the DIRECT CDA in execution-time yet they both accomplished adds and multiplies in software. Although this decrease varies significantly from example to example, it primarily depends on the degree of "parallelism" indigenous to the

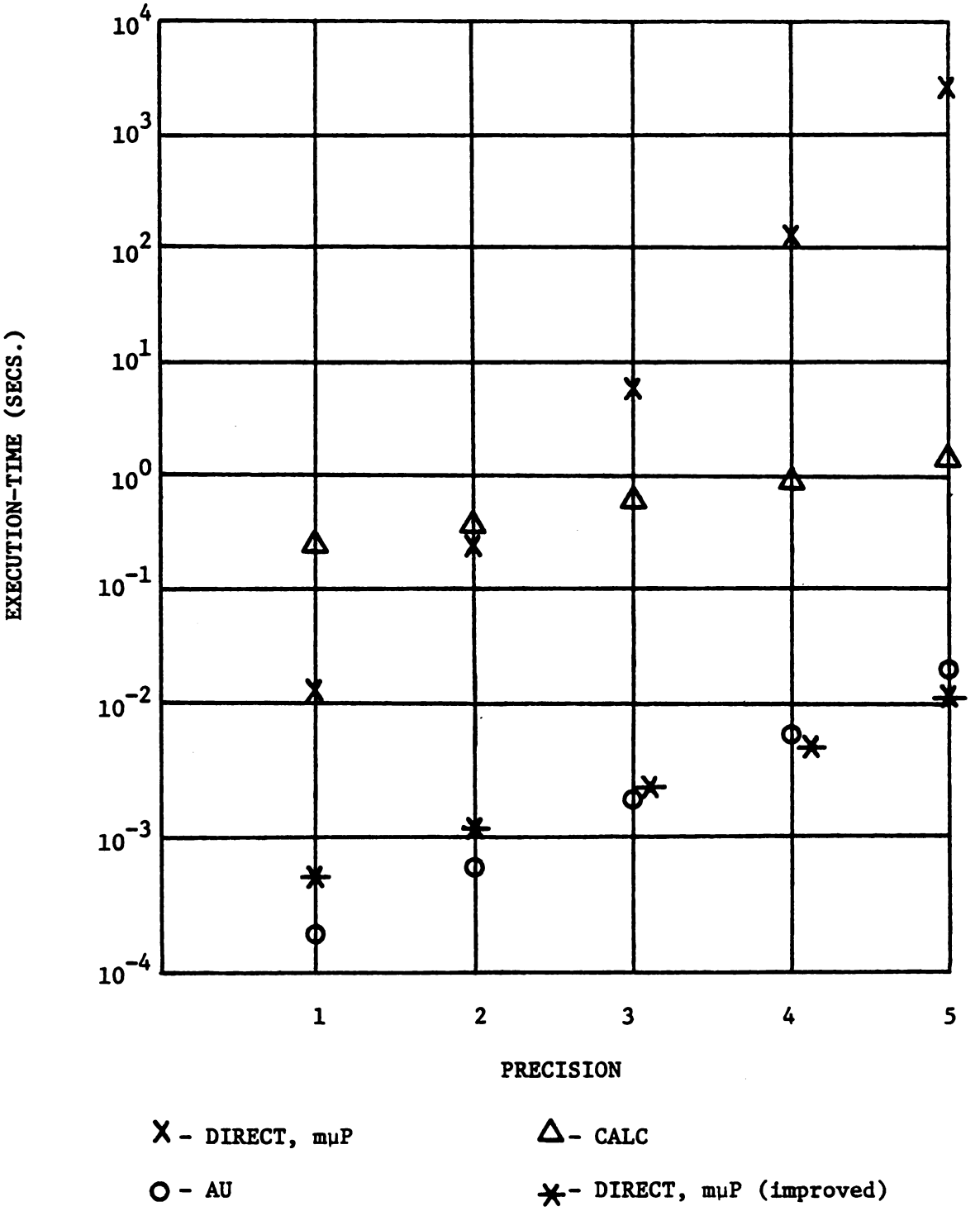


Figure 5.38 Multiple-precision multiply time for various CDA's.

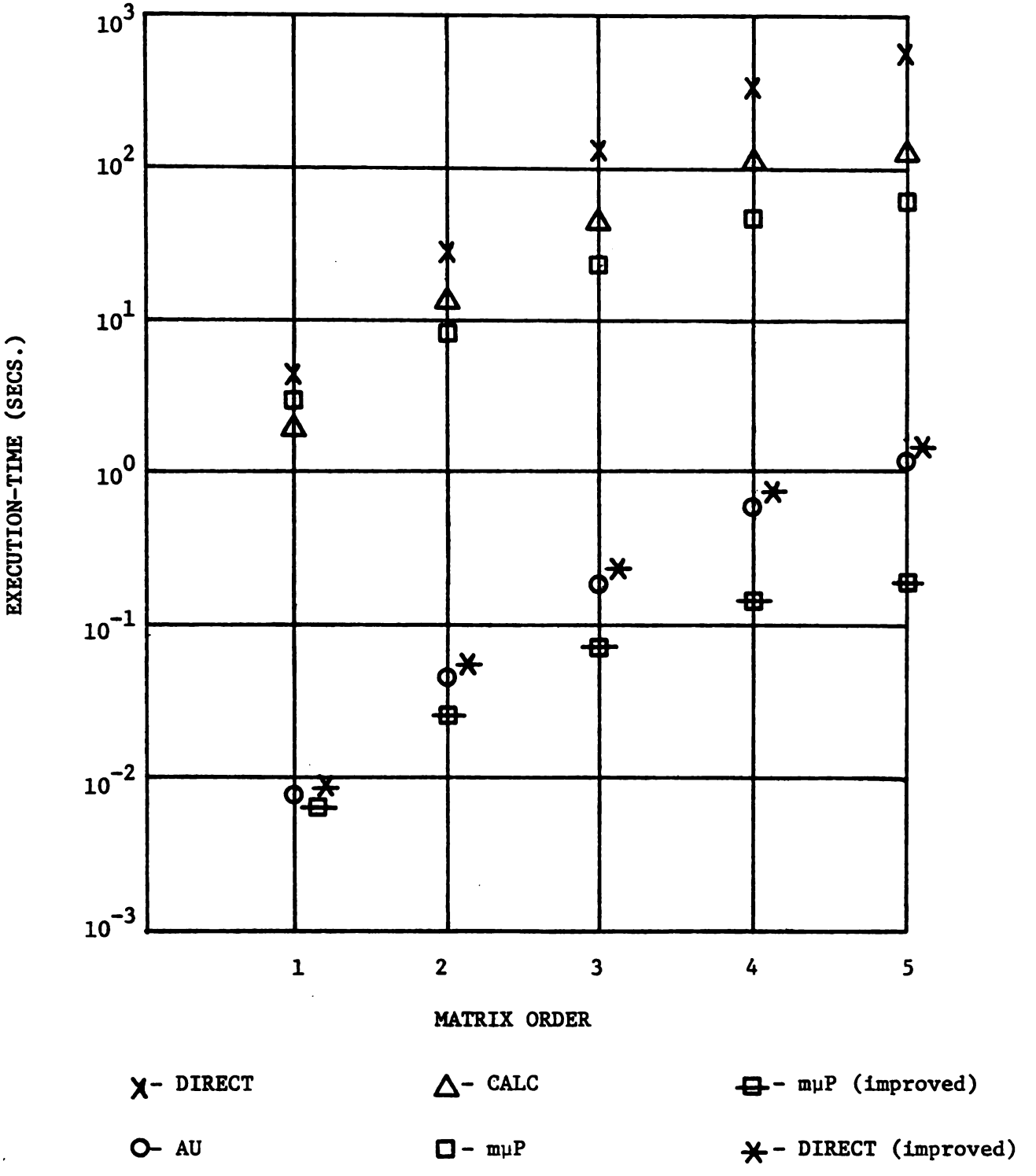


Figure 5.39 Triple-precision execution-time versus matrix order for Example-2.

Table 5.20 Typical consequence set and utilities for example-2(2nd order, improved multiply algorithm).

M	$\bar{x}_M$	$u_i(x_i)$	$u(\bar{x}_M, \bar{k}_a)$	$u(\bar{x}_M, \bar{k}_1)$	$u(\bar{x}_M, \bar{k}_2)$	$u(\bar{x}_M, \bar{k}_3)$
	3	66				
1	9.72E-2	99	80	73	27	97
	2.22E1	100				
	1	00				
2	3.84E-3	100	45	15	85	51
	1.25E2	51				
	4	100				
3	1.99E1	00	30	80	10	10
	2.31E2	00				
	2	33				
4	2.53E-2	99	67	46	92	86
	4.05E1	91				

Note: $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), $\bar{k}_3$ = (0.1, 0.1, 0.8)

specific computation and the severity of the Master's overhead. By extending the idea of parallel (or simultaneous) operation, even the AU CDA can realize a decrease in execution-time.

For single-precision problems the CALC CDA computes its tasks the slowest. Now, as precision increases the CALC CDA execution-times fail to grow as fast as the other CDA's (specifically, the CDA's with software multiplies); thus, it becomes faster than the DIRECT/mμP CDA's. Since the CALC CDA always finds the answer to a fixed number of digits, greater precision only requires additional digit entries, and this contributes little to the execution-time.

As the problem dimensions increase the total costs grow for all CDA's due to the extra data memory (not program memory) demands. And for the mμP CDA costs enlarge dramatically faster than the other CDA's because of repeated hardware and software. Thus, parallel execution can potentially create enormous costs.

This chapter provides typical consequence sets, marginal utilities, and utilities for various weight vector $\bar{k}$; i.e., $\bar{k}_a$ = (0.3, 0.3, 0.3), $\bar{k}_1$ = (0.8, 0.1, 0.1), $\bar{k}_2$ = (0.1, 0.8, 0.1), and $\bar{k}_3$ = (0.1, 0.1, 0.8). The first does not emphasize any attribute and, thus, corresponds to the average, while the other three emphasize one attribute at a time. For all examples, the typical consequence sets contain alternatives with equal utilities (for some $\bar{k}$), or "break-even" CDA's. These CDA's represent equivalent alternatives with respect to the others in the consequence set. When the weight vector $\bar{k}$ varies, the utility of CDA's with strong marginal utilities that correspond to emphasized $\bar{k}$ dimensions increase, while CDA's with weak marginal utilities decrease. This follows mathematically since the "sensitivity" of the additive utility function to a change in k_i yields the marginal utility; i.e.,

$$\frac{\partial}{\partial k_i} u(\bar{x}) = \frac{\partial}{\partial k_i} \sum_{i=1}^{N} k_i u_i(x_i) = u_i(x_i).$$

Finally, this chapter explores the overall effect of an alternative multiplication algorithm for the DIRECT and mμP CDA's. As precision increases, the degree of improvement increases; e.g., for p = 1 the instruction execution-time magnitudes differ by roughly two orders of magnitude, but for p = 5 it increases to five orders of magnitude. When applied to the matrix inversion example, this algorithm reduces the computation execution-time by three orders of magnitude for triple-precision. With respect to utilities, the improved multiplication instruction increases the utilities of the DIRECT and mμP CDA's, while it decreases the CALC CDA's utility because of slow speed. So the overall effect involves shifting the DIRECT and mμP CDA's utilities higher.

CHAPTER VI CONCLUSIONS

Continued advances in semiconductor device processing technology led to LSI devices with enhanced performance at reduced cost. Consequently, microprocessor-based (μP-based) systems now function in a myriad of application areas with widely differing computational complexities. Since numerous LSI devices may accomplish these advanced calculations, many designers follow ad hoc approaches to synthesizing computational sections. So the purpose of the research reported here was to investigate and characterize these computational design alternatives (CDA's) and to develop a more rigorous and structured design approach which incorporates a firm, theoretic foundation.

Using the results of this study, designers of μP-based systems will be able to create advanced systems with increased performance. For example, careful use of these CDA characteristics may lead to increased execution-time at lower costs. Second, these results will benefit LSI chip designers by influencing what properties the next generation of devices will possess. Because some properties aid computations and others hinder, the new devices will contain the desirable traits and minimize the action of undesirable ones. Third, the methodology developed in this study assists developers of μP-based systems by providing them the rigorous theoretic foundation to investigate and characterize future systems as technology advances. Since this development begins at the axiom level, they can easily modify the methodology to suit their specific orientation.

Within the research reported here, the investigation attained several objectives. First, it surveyed the present semiconductor market for LSI devices which facilitate computations and it generalized their key properties. With these devices, it defined a basic set of elemental CDA's which may be combined to realize the computation section of a generalized μP-based system. To provide for comparison of these members, it determined common attributes and techniques to evaluate them. Next, the investigation developed a decision mechanism for selecting the "best" CDA using these attributes. Finally, through three representative and contemporary engineering examples, it illustrated the use of this decision mechanism; in the process, it characterized the properties of each CDA. By satisfying these objectives, the investigation achieved the overall purpose of the research project.

During this investigation the research project reached several principal results. First, it defined a basic set of four elemental CDA's: DIRECT, AU, CALC, and mμP. The DIRECT CDA consisted of a μP and memory connected via the system bus. This CDA described the simplest technique to execute a calculation because the memory contained all arithmetic subroutines. Next, the second CDA employed a μP, memory, and an arithmetic unit (AU) all joined by the system bus. It differed from the DIRECT CDA in that it performed multiplies and divides with the AU. The third CDA incorporated a μP, memory, and calculator chip to accomplish arithmetic operations. With this memory, the program must simulate depressing the keys as with a hand held calculator, and it must read back the answers and decode them when the requested function finishes. Finally, the mμP CDA applied the concept of simultaneous, or parallel, execution: here, a Master μP directs several Slave μP's to execute sections of the

problem together, then it adds the partial results for the completed answer. Thus, each Slave memory contained a replica of the DIRECT CDA memory, yet the Master's memory held an "overhead" program.

To facilitate comparison of these CDA's, the investigation identified and used three common attributes: precision, execution-time, and cost. Precision involves the quantity of 8-bit words used to represent a number; hence, all numbers corresponded to a multiple-precision, fixed-point, two's complement format. For execution-time, this value gave the number of seconds needed to complete the requested computation. If the number of add and multiply instructions indicates the execution-time, then these values, multiplied by the time to perform each instruction, add to give the execution-time. With costs, this quantity represented the dollar value of LSI devices used in a CDA. Two terms combined to produce the total cost; one resulted from the basic expense of procuring a CDA, and the other pertained to the additional expense of the application memory. For both execution-time and cost, these values occurred from terms indigenous to each CDA and from application derived terms. Multiple-precision flowcharts for arithmetic operations assisted in estimating these indigenous terms. The CALC CDA add time exceeded the other CDA's (all the same) by three orders of magnitude; i.e., msecs. compared to μsecs. For multiplies the CALC CDA exhibited roughly 1 sec. operation, while the AU CDA functioned in about 1 msec. But the DIRECT and mμP CDA's varied considerably with precision since repetitive adds implemented the multiply; i.e., they ranged from msecs. to 3.5 mins.

With respect to indigenous costs, the DIRECT CDA differed from the AU and CALC CDA's by roughly one order of magnitude less ($100's compared to $10's), and the mμP CDA varied dramatically with the number of

Slaves. For more than 12 Slaves, the mμP CDA represented the largest indigenous cost.

Next, the investigation considered a viable decision mechanism to select the "best" CDA which used the above attributes: Multiattribute Utility Theory (MUT). Here, MUT assigned a numeric quantity, the utility, to each CDA which indicated its usefulness with respect to the other alternatives. Because the attributes satisfied eight axioms, and utility and preferential independence definitions, the decision mechanism reduced to an additive utility function; i.e.,

$$u(\bar{x}) = \sum_{i=1}^{3} k_i u_i(x_i) = \text{utility} \qquad (6.1)$$

$$\bar{x} = (x_1, x_2, x_3) = \text{consequence}$$

x_1 = precision, x_2 = execution-time, x_3 = cost

$u_i(x_i)$ = marginal utility function

k_i = scaling constant, where $\sum_{i=1}^{3} k_i = 1.$

Here, marginal utility functions converted specific attributes of a dimension to a numeric quantity which represented its usefulness. So, the utility of a CDA involved a weighted sum of their marginal utilities, and the "best" CDA consisted of the alternative with the greatest numeric utility.

By analyzing three examples, the investigation clarified the techniques used to evaluate attributes, and it elucidated use of the additive utility function. These applications--linear regression, matrix inversion, and fast Fourier transform computations--exemplified the overall procedure of this research project and, moreover, led to conclusions that characterized each CDA.

For the linear regression example, a typical consequence set included the following marginal utilities: DIRECT = (60, 00, 100), AU = (00, 100, 53), CALC = (100, 19, 00), and mμP = (33, 97, 62). With the weight vector $\bar{k}$ = (0.3, 0.3, 0.3), Equation 6.1 formed the utility of each CDA as the average of its marginal utilities; i.e., DIRECT = 55, AU = 51, CALC = 40, and mμP = 64. Since the mμP CDA possessed the largest utility, it represented the "best" CDA in the above consequence set. As the values in the weight vector changed, the utility of CDA's with strong marginal utilities that correspond to emphasized $\bar{k}$ dimensions increased, while the others decreased, and the "best" CDA varied accordingly. So determining the weight vector $\bar{k}$ involves important evaluation. Similarly, the other examples strengthened these results.

In characterizing the CDA's, the three examples revealed that the AU CDA always exhibited the fastest execution-time because of its hardware multiply and divide instructions. For the DIRECT and mμP CDA's, which both perform adds and multiplies in software, the mμP's speed invariably surpassed the DIRECT CDA due to its simultaneous, or parallel, execution. But this increase varied significantly from example to example, and it primarily depended on the degree of "parallelism" found in the specific example and the severity of the Master's overhead. With respect to precision, the CALC CDA represented the slowest CDA for single-precision, yet as precision increased it becomes faster than the DIRECT and mμP CDA's and still slower than the AU CDA. Since the CALC CDA always finds the answer to a fixed number of digits, greater precision only requires extra digit entries and this contributes little to the execution-time. As the problem dimensions grew, so did the total cost of all CDA's because of additional data (not program) memory demands.

For the mμP CDA, costs enlarged dramatically because of its repeated hardware and software. Finally, precision changes did not alter the costs significantly since the memory requirements of the problem dimensions dominate those of added precision.

As with any research project, the investigation reported here points toward several areas of additional study. For example, rather than characterize CDA's through three specific applications, perhaps a generalized algorithm would induce broader conclusions. If integer N measures the size, or dimension, of a generalized algorithm, then the time complexity, T(N), expresses the number of seconds required to complete the task. Similarly, the space complexity of the algorithm, S(N), denotes the number of memory words needed to execute the task (this quantity relates closely to the attribute cost). So for fixed execution-interval and memory size, such expressions could give limits to the problem dimensions for an entire class of problems.

Besides extending this research project to match technological growth, the fundamental concepts of MUT could be used to investigate and characterize various mμP systems in detail. With enhanced throughput, improved reliability, increased system response, and modular expansion capabilities, the mμP arrangement deserves much attention. Still a flock of problems exist which need careful research before implementation progress can occur. But MUT paves the path since it can facilitate analysis of both hardware and software.

Finally, since the methodology developed in this research project does not assume any specific hardware devices or software algorithms, it could aid a parametric study of either these two topics. Such

investigations may lead to optimal design strategies for μP-based systems which consider both hardware and software tradeoffs.

REFERENCES

REFERENCES

1) R. K. Jurgen, "Electronics in medicine," *IEEE Spectrum*, vol. 15, pp. 68-72, Jan. 1978.

2) H. L. Van Trees, E. V. Hoversten, "Communications satellites: Looking to the 1980s," *IEEE Spectrum*, vol. 14, pp. 42-51, Dec. 1977.

3) S. L. Lillevik, P. D. Fisher, and A. L. Jones, "A predictive CMOS-based instrument for agriculture," *Microcomputer Design and Applications* (ed. S.C. Lee). New York: Academic Press, 1977, pp. 275-286.

4) P. D. Fisher and S. L. Lillevik, "Monitoring system optimizes apple-tree spray cycle," *Electronics*, vol. 50, pp. 125-127, Nov. 24, 1977.

5) S. L. Lillevik, P. D. Fisher, and A. L. Jones, "A predictive field instrument for agricultural production," *Proc. IEEE Microcomputer '77 Conf.*, pp. 137-142, Apr. 1977.

6) R. K. Jurgen, "Electronic funds transfer: Too much, too soon?," *IEEE Spectrum*, vol. 14, pp. 51-54, May 1977.

7) C. C. Foster, "Something new--The Intel MCS-4 microcomputer set," *Comp. Arch. News*, vol. 1, pp. 16-17, Apr. 1972.

8) J. L. Ogdin, "Survey of 8-bit microprocessors reveals wide choice for users," *EDN*, vol. 19, pp. 44-50, June 20, 1974.

9) R. L. Petritz, "The pervasive microprocessor: Trends and prospects," *IEEE Spectrum*, vol. 14, pp. 18-24, July 1977.

10) P.W.J. Verhoftstad, "Technology for microprocessor hardware," *Proc. IEEE Spring COMPCON*, pp. 277-282, Apr. 1976.

11) S. Y. Lau, "Design high-performance processors," *Electronic Design*, vol. 25, pp. 86-95, Mar. 29, 1977.

12) S. Y. Lau, "Bit-slice microprogramming saves software compatibility," *EDN*, vol. 23, pp. 42-46, Mar. 5, 1978.

13) J. Nemec, G. Sim, and B. Willis, "A primer on bit-slice processor," *Electronic Design*, vol. 25, pp. 52-60, Feb. 1977.

14) G. F. Muething Jr., "Designing the maximum performance into bit-slice processors," *Electronics Design*, vol. 25, pp. 52-60, Feb. 1977.

15) G. C. Feth, "Memories: Smaller, faster, and cheaper," *IEEE Spectrum*, vol. 13, pp. 36-43, June 1976.

16) N. Lindgren, "Semiconductors face the '80s," *IEEE Spectrum*, vol. 14, pp. 42-48, Oct. 1977.

17) T. S. Bush, "Local memory for a high-speed digital test system," *IEEE Trans. Instr. and Meas.*, vol. IM-26, pp. 217-220, Sept. 1977.

18) P. Franson, "Special report: Semiconductor memories," *EDN*, vol. 22, pp. 46-58, June 20, 1977.

19) E. A. Torrero, "Bubbles rise from the lab," *IEEE Spectrum*, vol. 13, pp. 28-31, Sept. 1976.

20) P. Franson, "IC's and semiconductors--suppliers rush to perfect new processes," *EDN*, vol. 22, pp. 88-95, Dec. 15, 1977.

21) R. Greene, G. Perlegos, P. J. Salsbury, and W. L. Morgan, "The biggest erasable PROM yet puts 16,384 bits on a chip," *Electronics*, vol. 50, pp. 108-111, Mar. 3, 1977.

22) R. Proebsting, "Dynamic MOS RAM's: An economic solution for many system designs," *EDN*, vol. 22, pp. 61-66, June 20, 1977.

23) E. A. Torrero, "Solid-state devices," *IEEE Spectrum*, vol. 15, pp. 36-40, Jan. 1978.

24) E. R. Hnatek, "Current semiconductor memories," *Computer Design*, vol. 17, pp. 115-126, Apr. 1978.

25) E. A. Torrero, "The multifacets of I^2L," *IEEE Spectrum*, vol. 14, pp. 28-36, June 1977.

26) L. G. Gardner, "A survey of some recent contributions to computer arithmetic," *IEEE Trans. Comp.*, vol. 25, pp. 1277-1282, Dec. 1976.

27) S. Sanyal, "An algorithm for nonrestoring division," *Computer Design*, vol. 16, pp. 124-127, May 1977.

28) S. Waser and A. Peterson, "Real-time processing gain ground with fast digital multiplier," *Electronics*, vol. 50, pp. 93-99, Sept. 29, 1977.

29) "Multiplier integrates accumulator, provides 175-nsec multiply-accumulate," *EDN*, vol. 22, p. 92, Aug. 5, 1977.

30) "24-bit single-chip multiplier eases floating-point computations," *EDN*, vol. 23, p. 156, June 5, 1978.

31) "Am9511: Arithmetic processing unit," prepared by Advanced Micro Devices, Inc. (Sunnyvale, Calif.), 1977.

32) "Number-processing controller cuts cost of software development," *Computer Design*, vol. 16, pp. 128-129, Aug. 1977.

33) P. D. Fisher and S. M. Welch, "Wedding calculators and instruments," *Electronics*, vol. 47, pp. 100-109, May 16, 1974.

34) W. W. Moyer, "Interfacing calculator chips as microcomputer preprocessors," *Computer Design*, vol. 17, pp. 187-191, May 1978.

35) T. P. Hughs, D. H. Sawin III, and D. R. Hadden Jr., "LSI Software," *Proc. IEEE Microcomputer '77 Conf.*, pp. 46-53, Apr. 1977.

36) P. Franson, "Advent of standard chip software sure to ease system implementation," *EDN*, vol. 22, pp. 21-22, Oct. 5, 1977.

37) J. L. Baer, "Multiprocessor systems," *IEEE Trans. Computers*, vol. 25, pp. 1271-1277, Dec. 1976.

38) A. J. Weissburger, "Analysis of multiple-microprocessor system architectures," *Computer Design*, vol. 16, pp. 151-163, June 1977.

39) J. W. Bowra and H. C. Torng, "The modeling and design of multiple function-unit processors," *IEEE Trans. Computers*, vol. 25, pp. 210-221, Mar. 1976.

40) P. M. Russo, "An interface for multi-microcomputer systems," *Proc. IEEE Fall COMPCON*, pp. 277-282, Oct. 1976.

41) W. C. Giauque and T. C. Peebles, "Application of multidimensional utility theory in determining optimal test-treatment strategies for streptococcal sore throat and rheumatic fever," *Opns. Res.*, vol. 24, pp. 933-950, Sept. 1976.

42) J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, N.J.: J. Wiley and Sons, 1967, pp. 15-31.

43) R. L. Keeney, "Quasi-separable utility functions," *Naval Res. Log. Quart.*, vol. 15, pp. 367-390, 1968.

44) R. L. Keeney, "Multiplicative utility functions," *Opns. Res.*, vol. 22, pp. 22-34, Jan. 1974.

45) *ibid.*, p. 24.

46) K. R. MacCrimmon and M. Toda, "The experimental determination of indifference curves," *The Review of Economic Studies*, vol. 36, pp. 433-451, 1969.

47) R. M. Dawes and B. Corrigan, "Linear models in decision making," *Psychological Bull.*, vol. 81, pp. 97-106, Feb. 1974.

48) H. J. Einhorn and R. H. Hogarth, "Unit weighting schemes for decision making," *Organizational Behavior and Human Performance*, vol. 13, pp. 171-192, Apr. 1975.

49) W. Edwards, "How to use multiattribute utility measurement for social decision making," *IEEE Trans. Sys. Man and Cybern.*, vol. SMC-7, pp. 326-340, May 1977.

50) H. Raiffa, Decision Analysis: *Intro. Lectures on Choices Under Uncertainty*. Reading, Mass.: Addison-Wesley, 1968.

51) W. Edwards, "Social utilities," *The Engineering Economist, Sum. Sym. Ser.*, vol. 6, pp. 119-129, 1971.

52) J. M. H. Olmsted, *Advanced Calculus*. New York: Appleton-Century-Crofts, 1961, pp. 524-525.

53) R. Grossman and J. Conway, "Minicomputer Systems Directory," *EDN*, vol. 21, wall chart, June 5, 1976.

54) C. G. Cullen, *Matrices and Linear Transformations*. Reading, Mass.: Addison-Wesley, 1967, pp. 188-192.

55) T. M. Walker, *An Introduction to Computer Science and Algorithmic Processes*. Boston: Allyn and Bacon, 1971, pp. 324-325.

56) D. G. Moursund and C. S. Duris, *Elementary Theory and Application of Numerical Analysis*. New York: McGraw-Hill, 1967, p. 64.

57) J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex Fourier series," *Math. Computation*, vol. 19, pp. 297-301, Apr. 1965.

58) W. D. Stanley, *Digital Signal Processing*. Reston, Virginia: Prentice-Hall, 1975, p. 270.

59) *ibid.*, p. 259.

60) A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, Mass.: Addison-Wesley, 1974, pp. 2-33.

61) F. J. Hill and G. R. Peterson, *Digital Systems: Hardware Organization and Design*. New York: J. Wiley and Sons, 1973, p. 159.