EFFICIENCY IN THE USE OF A COMPUTER FOR NETWORK ANALYSIS*

ROBERT M. MUÑOZ
Ames Research Center
Moffett Field, California

S. PARK CHAN
University of Santa Clara
Santa Clara, California

INTRODUCTION

Because the computer is being used effectively in many fields, especially in the fields of circuit and system design, a real motivation exists for efficiency. Many software packages such as SEPTRE, ECAP, and NET 1 have proven the point that circuit and system design engineering can be effectively aided by intelligent use of the computer and this use is limited not only by available computing equipment but also by methods of computing, methods which vary widely in efficiency. Electronics circuit analysis programs of today are very sophisticated and capable of analyzing large networks. They very often require memory capacity and computing times that tax the resources of a small computing installation. The use of such programs represents a cost that can be measured by a number of factors including computing time and memory capacity and this cost must be compared with the benefit derived. Some programs are less costly than others when performing the same analysis function and some programs are definitely unsuited to performing certain computations, not because they lack the capability, but because the cost of doing these computations is too great.

This paper has been written in an attempt to discuss the problem of efficiency of computation in the digital computer in a conceptual way. The cost of a computation as measured by the time necessary for performing the required operations and the memory capacity or other hardware requirements is treated. A method of analysis utilizing the graph theoretical approach for evaluating the cost of computation is introduced. This method is new insofar as it makes a logical distinction between the linear graph of a computation and the linear graph of a program and shows in generality the importance of the relationships between these two graph models without going into any great depth to describe the algebra of these relationships. A number of distinctions in graph structure such as cyclic operations and variations in the types of operations such as deterministic or probabilistic are discussed. The optimization problem is viewed as it relates to the assignment of priorities among processors in a multiprocessing hardware configuration.

*This work was supported in part by the National Aeronautics and Space Administration, Grant No. NGR-05-017-012.

Elements of the Cost of Computation

The fundamental and prerequisite costs of doing electronic circuit analysis and design by means of a digital computer is that cost associated with the acquisition of a computer of computing service; however, this acquisition or rental cost of hardware will not be discussed here. In this paper, it will be assumed that digital computing equipment is available and a motive to make efficient use of this equipment exists. The more important elements of the cost of computing depend upon (1) computing time, (2) computer capacity and (3) computer error probability.

The computational task usually begins with programming. Once the programming effort is completed and information for program input is presented in some acceptable form to the computer, a series of events takes place which constitute the more important time intervals in the computing process. These time intervals recur each time a problem is solved by the computer and can be broken up into the following general categories: (1) input time, (2) compile time, (3) load time, (4) run time, and (5) output time. Input time is self-explanatory. This is the time to enter the problem into the computer by means of whatever man machine interface is available, be it teletype, punched cards, graphics terminal, or the like. It includes time spent in input system software manipulations. Compile time is that time necessary to translate the language of the program into symbolic machine language. In most computing systems a higher level language such as Fortran IV is compiled by means of a system function or utility program and an output, typically a binary output, is made available for the computer or some intermediate storage means such as disk or tape memory so that the compiled program can be loaded into the computer at an another time. Load time is the time necessary to input the compiled program into the active computer memory. Loading is also done typically by means of a utility subroutine. Once a problem has been compiled and loaded the run time measures the activity of the computer as it goes about actually solving the problem presented. Very often the run time is a small fraction of the total time spent in the computing system. Typical experience with a topological circuit analysis programs of Calahan shows that compile and load time on an IBM 7040/7094 DCS system is of the order of one to two minutes whereas run time seldom exceeds one-half minute. Output time is that time necessary to present answers in a form suitable to their end use. Input, compile, load and output times can be considered as computer overhead. These times will be expended no matter how simple the problem is. As the problem grows in size the magnitude of run time increases to a point where it is equal to or greater than the overhead time as shown in Figure 1.

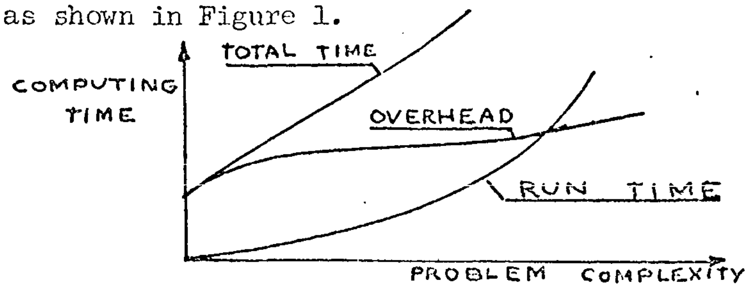


Figure 1
Computing time as a function of problem complexity

Every computer system is different, there are no simple general principles which apply to all. Input and output times depend upon the capabilities of peripheral equipment. Compile and load times depend upon the language of the program and the sophistication of a utility routine. Run time on the other hand is amenable to analysis, it does depend on hardware configuration to some extent but, to a greater extent, it depends upon the techniques used to perform computation. It is this time that forms the center of interest in our discussion of computational efficiency.

ANALYSIS OF THE COMPUTING TASK

Basic Classes of Computer Operations

The instruction repertoire of a computer usually includes all ordinary arithmetic operations: addition, substraction, multiplication and division and with the aid of utility routines, differentiation, integration and other similar operations. These are good examples of operations that can be classified deterministic. They are deterministic in the sense that they can be performed in a finite length of time, which time remains the same or nearly the same whenever the operations are performed. Other operations natural to computing equipment and which fall into the class of deterministic operations are the logical functions "And", "Or" etc. There is another class of operations which have no uniquely defined a priori outcome. The recursive cyclic operations discussed below are such non-deterministic operations. Computer branching operations such as "if" statements and "go to" or "jump" statements fall into this class. These are called probabilistic operations because it is assumed that a finite probability exists that such an operation will produce a given result. The run time mentioned earlier is composed of the time necessary to perform all operations whether deterministic or probabilistic. In the analysis of run time for some problems, it may be more meaningful to view the computation process as entirely probabilistic or for some computations it may be more advantageous to view the operations as deterministic or a combination of deterministic and probabilistic. A more detailed discussion of these topics will be given below.

Cyclic Operations

The computing task is seldom devoid of cyclic operations. Computing cycles are of two basic types, recursive and indexed. The recursive cycle is a series of computing operations repeated until some test performed on the data computed conforms to a test criterion at which point the operation is complete. It is not possible beforehand to determine the number of cycles in the operation without at first knowing something about the character of the input data in the cycle of operations. The indexed cycle, on the other hand, is a fixed number of repetitions of the same computing operations. This number is determined a priori and remains the same for all evaluations of input data. Cyclic operations typically multiply computing times by large factors.

Background

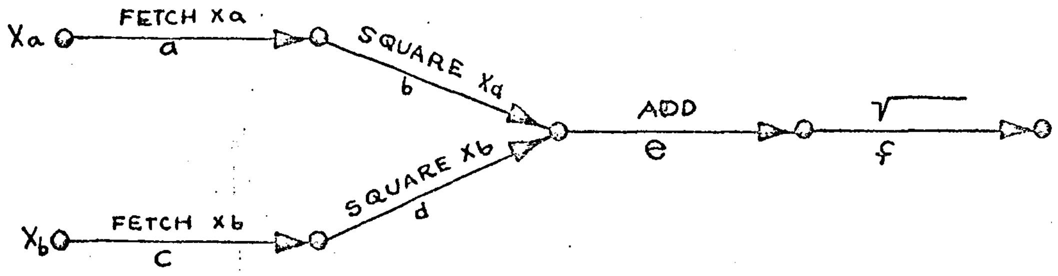
There has been relatively little theoretical work on the analysis of the computing task reported in the literature. This is probably so partly because of the complexity of performing such analysis and partly because mathematical techniques have not been fully developed to a sufficient degree that such analysis is entirely possible or practical. At the present time, the consensus of those active in the literature seems to indicate that the theory of linear graphs can and should be used as a tool for analyzing the computing task. Elmaghraby [1] has developed a mathematical technique for the analysis of generalized activity networks such as those used in scheduling of business operations; however, much of this work is directly applicable to the analysis of the computing task. Elmaghraby's work is monumental in the sense that his mathematics for the algebra of generalized activity networks forms an excellent unifying concept which allows other graph theoretical concepts such as the signal flow graph and the PERT type scheduling flow graphs to be treated as a special case. Pritsker [3] and others have developed the theory of the generalized activity network especially as it applies to the scheduling problem. Martin and Estrin have done some experimental work using computer models of mathematical computation processes to analyze such quantities as total computing time and the probability of occurence of a given output. Karp [5] and Marimount [6] have developed some useful techniques for the analysis of the computer programming task which are more or less applicable to the analysis of the entire computational task. There are, however, important distinctions which will be discussed later. Analysis of the computing task is at best a difficult job and a job for which no altogether suitable mathematical techniques have been developed. A considerable amount of work needs to be done in this field before satisfactory theory is developed and before it will be possible to accurately estimate the cost of all computational tasks. In this section we will discuss some of the theory already in existence along with some new applications of older graph theoretical concepts and we shall attempt to indicate where possible those areas which could provide the greatest benefit from future theoretical work, also a new distinction between computation graphs and program graphs is made.

Graphical Model of a Computation

The first step in the analysis of a computational task is the generation of a linear oriented graph known also as directed graph or digraph representing the process. There are at least two basic methods of assigning qualitative and quantitative information to the elements of a linear oriented graph. The first used by Estrin and Turn [10] Marimount

[6] and others assigns a quantitative evaluation to the nodes or vertices of the graph and a precedence relationship or the essential topology of the graph to the directed edges. This philosophy of assignment of vertex and edge significance is very different from that used by Karp and others wherein the edges are assigned a quantitative value such as execution time and probability and vertices are used to formulate the topology and to establish precedence. In a sequence of operations within a computational task, vertices can be thought of as representing temporal milestones or the instant that a particular operation begins or ends. It is this type of vertex and edge assignment that will be considered exclusively in this paper, partly because a greater body of theoretical knowledge is available for graphs with these qualities and partly because of the value of such a graph as an aid to visualizing the operations contained within a computing task. Such a graph corresponds more closely to the signal flow graph and the graph used in communications theory thus such a graph benefits by mathematical developments in these related fields.

A typical simplified model of the linear oriented graph of a computational task is given in Figure 2. This graph has been derived from the mathematical formula for computing the magnitude of a vector from its orthogonal components (the Pythagorean theorem).



$$Y = \sqrt{X_a^2 + X_b^2}$$

Figure 2
Linear Oriented Graph of a Computing Task

Note that the mathematical formula establishes certain precedence relationships, that is, the squares must be taken before summing and each must be complete before the final square root operation is performed. Every computational task has certain precedence relationships which remain inviolate. These are adequately expressed in the form of a linear oriented graph and it is this graph which can be thought of as the fundamental means of analysis.

The topology of a graph representing a computation is very different from the topology of a graph representing a computer program as will be demonstrated by writing a simple program for the function given in Figure 2. VIZ:

Program event	
a.	fetch X_a
b.	square X_a and hold
c.	fetch X_b
d.	square X_b and hold
e.	sum $X_a^2 + X_b^2$ and hold
f.	take square root of $X_a^2 + X_b^2$

In this simple program, all events take place in sequence and the entire computing operation can be represented as shown in Figure 3.

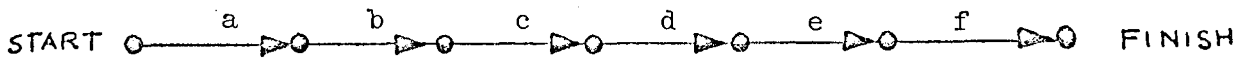


Figure 3
Linear Oriented Graph of the Computer Program of the Pythagorean Theorem

A set of rules or an algebra could be developed for generating the program graph from the computation graph in a systematic way. The value would be great in providing a systematic way whereby the precedence relationships could be preserved and the total computing costs determined a priori. For the more complex programming tasks where a multiprocessing capability exists the program graph can be used to evaluate effects of parallel computing. The program graph for the simultaneous use of two processors for this example would be topologically identical to the computation graph and the task would take less time as a result. From this it should be clear that the program graph is dependent not only on the topology of the computation graph but also on the capability of processing hardware. Additional research is needed in this area especially on the problems of optimum multiprocessing.

Significance of Edges, Vertices, Paths and Loops

So far, the development of this method of analysis by means of the graph model of a computational task has been thoroughly general in that it applies equally well to schedules, signal flow graphs and any other process which depends on a quantitative edge nomenclature and precedence relationships between edges but in the analysis of the computation task we shall ascribe the following meaning to edges, vertices, paths, and loops:

1. Edges represent computing operations and the weight or value assigned to an edge shall represent the set of quantities that measure the computing operation such as probability of execution, execution time, memory requirement, chance of error, etc.

2. Vertices as mentioned earlier represent the beginning and end of an operation.

3. Paths represent a sequence of operations with weights or values computed as joint probabilities, total time of execution and total chance of error, etc. for the edges traversed.

4. Loops represent cyclic operations or a recurring set of identical operations. Recursive and indexed loops are possible as indicated earlier under cyclic operations.

To illustrate these relationships, we will use another simplified example of a computing task as shown in Figure 4 for a graph containing a deterministic cycle.

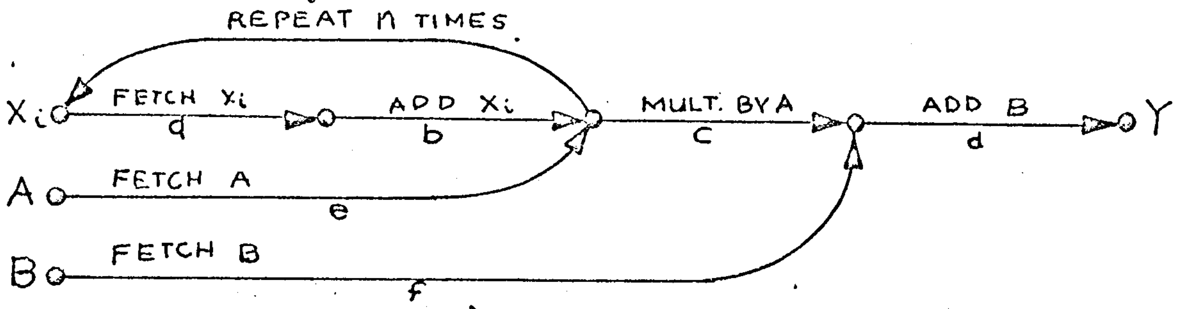


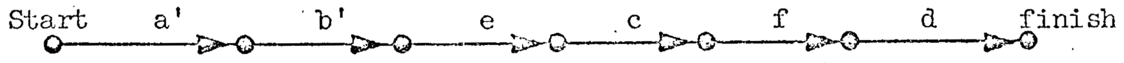
Figure 4
Computation Graph for the Expression $Y = A \sum_{i=1}^{n} X_i + B$

The quantitative information associated with the edges of the graph of Figure 5 is as follows:

Edge	Operation Time (sec)	Probability of occurrence
a	1 sec.	1
b	2	1
c	3	1
d	2	1
e	1	1
f	1	1
g	n (a multiplying factor)	1

In order to evaluate this graph quantitatively, it will first be necessary to discover all cyclic operations or loops, and transform these into acyclic equivalents. Then, a program graph can be developed on the basis of some fixed computing hardware configuration. From the program graph will come the results desired in the form of the cost of the computation task measured in seconds and the probability of occurrence of the output Y. Clearly, in this example, the output Y is a certainty because the probability of occurrence of each edge is unity.

The cyclic to acyclic transformation in this case simply infers that edges a and b are replaced by edges a' and b' having operation times n times larger than the original operation times. For serial computing hardware, the single path graph of Figure 5 represents the desired result. Martin and Estrin [8] have discussed cyclic to acyclic transformations of a program graph in a general way.



Edge	Operation Time	Probability of occurrence
a'	n(1) = n	1
b'	n(2) = 2n	1
c	2	1
d	3	1
e	2	1
f	1	1
g	0 (not in the graph)	0

Figure 5
Single Path Program Graph of the Function $Y = A \sum_{i=1}^{n} X_i + B$

Total computing time is seen to be (3n + 8 sec) the sum of all operation times from start to finish and the probability of occurrence of output is 1, the joint probability of occurrence of all edges.

If hardware variations are possible so that the greatest amount of parallelism in computing is allowed, the minimum computing time can be found to be proportional to the length of the longest path of Figure 5 after cyclic to acyclic transformation; namely, a', b', c, d or (3n + 5) seconds. These results are true in general, that is, the shortest possible time for computation is the maximum length path from start to finish of the computation graph and the longest is the sum of all operation times after cyclic to acyclic transformation has been considered.

The Probabilistic Computation

Consider the following probabilistic computation and the associated graph shown in Figure 6.

$$Y = \begin{cases} AX + B & \text{for } X \leq 0 \\ \dfrac{CX}{D} & \text{for } X > 0 \end{cases}$$

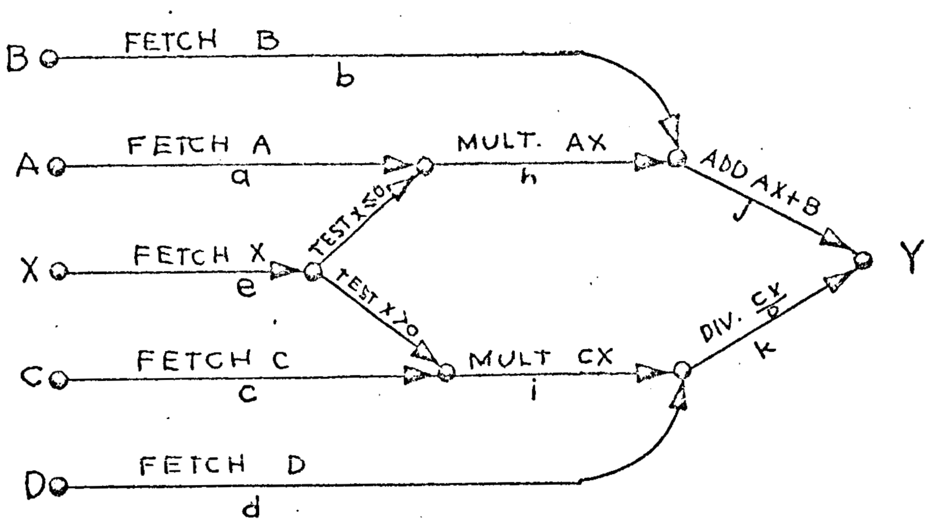


Figure 6
Graph of a Probabilistic Computation Process

In this graph, Vertex 6 represents the start of a branch or test operation, the outcome of which depends on the magnitude of X. Edge f represents an edge which is traversed only if X ≤0. And edge g represents an edge which is traversed only if X >0. To these edges is assigned the finite probability 0.5 as indicated in the following table:

Edge No.	Operation Time	Probability of Occurrence
a	1	1
b	1	1
c	1	1
d	1	1
e	1	1
f	2	.5
g	2	.5
h	4	1
i	4	1
j	1	1
k	8	1

The program graph of this function using serial computing hardware is not a single path but two alternate paths as shown in Figure 7.

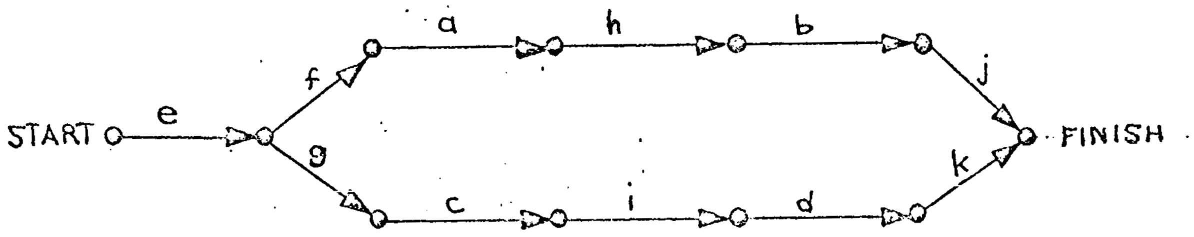


Figure 7
Program Graph for a Probabilistic Function of Computation

The average time of computation for a random variation of values of X is computed in the following way:

$$t_{av} = t_e + P_f\,(t_f + t_a + t_h + t_b + t_j)$$
$$+ P_g\,(t_g + t_c + t_i + t_d + t_k)$$

where t represents operation time and P represents probability of occurrence of an edge.

$$= 1 + .5\,(2 + 1 + 4 + 1 + 1)$$
$$+ .5\,(2 + 1 + 4 + 1 + 8)$$
$$t_{av} = 13.5$$

The probability of output is the sum of the joint probabilities through each path:

$$P_y = 0.5 + 0.5 = 1$$

By similar means, any number of probabilistic operations can be analyzed and the probability of output along with average execution times can be developed.

OPTIMIZATION OF A COMPUTATION

The problem of optimization of a computation process in a digital computing system includes, but is certainly not limited to the costs of computation as measured by computing time, hardware requirements and error probabilities. Optimization infers that certain criteria which may vary from one computation to another are satisfied. In one case optimization may mean allowing a certain computation to be performed with a minimum memory requirement in the hardware, whereas in another case, it may be required to reduce the total error probability. One minimization problem may deal with a completely serial set of operations whereas another may involve multiple processors. There are few general statements which can be made about the problem of optimization but one which does apply in most cases is this: A computation should make efficient use of the computing hardware to satisfy a primary objective such as speed or reliability while at the same time satisfying secondary objectives such as accuracy. There certainly are exceptions to this generality but our attention in this paper has been directed toward those applications where it applies. One possible exception would be computing in a system which has more than enough basic computing capacity. In such a system, it might be desirable to minimize the total programming effort by using relatively inefficient but easy to program subroutines. This problem and others of the type can be considered outside of the class of problems needing optimization.

The first step in any problem of optimization is purely mathematical. It involves the well-known techniques of algebraic simplification and similar means to guarantee that the answers which are sought are arrived at mathematically in the most expeditious way. Where two or more methods of mathematical analysis for the same problem exist, it may be necessary to analyze each method rigorously before it can be determined which is the more efficient. A very simple example of mathematical simplification is factoring. If a common factor exists in a number of terms in an algebraic expression, it is computationally more efficient to extract this factor, compute the indicated intermediate result, and multiply the intermediate result by the common factor before proceeding. Many similar operations and transformations are possible.

After one is assured that the simpliest possible mathematical expression for a given computational process is achieved, it is then necessary to evaluate this process quantitatively and qualitatively by means of the computation graph. The computation graph itself may point out overlooked possibilities for algebraic simplification. When the graph has been formed, an analysis of the graph may then be made. Serial chains of non-branching edges are then reduced to single edge equivalents. Also all loops or cyclic operations are found. Then, the graph is transformed into the equivalent acyclic reduced graph. This allows an even more accurate intuitive understanding of the topology of the computation graph.

The next and most critical step in the optimization process is the formation of a program graph for a computing system. With only serial computing capability, the problem is considerably reduced. In fact, since all necessary operations must be performed in sequence, certain general features of this type computing process can be discerned directly from the computation graph. For instance, the total time of computing using serial processing will be exactly the sum of all computing times evaluated using the computation graph alone. This of course is not the case with parallel computing hardware. A parallel computation requires considerably more effort to optimize. Martin and Estrin [8] have proposed an urgency criterion for optimization by an essentially trial and error process. The urgency criterion is a measure of path lengths from an operation in question to the end of the computation. If a program graph is being developed for parallel processors and the criterion of performance is minimum computing time, one can inspect the computation graph analytically to find the longest possible independent sub-graph which can be assigned to one of the processors. This sub-graph must be a serial chain of operations that can be performed simultaneously with the other operations of the computation and within the limitations of the available hardware.

Figure 2 is a simple example of a computation graph which could be programmed effectively in two parallel processors reducing total computing time by the smaller of the two times $(t_a + t_b)$ or $(t_c + t_d)$. The program graph in this instance would be identical to Figure 2, the computation graph. If a computation graph has no parallel paths then it is not possible to reduce total computation time by the use of multiple processors. On the other hand, if more parallel paths are possible than there are parallel processosrs available, the best assignment of the available equipment would be that assignment which produces the most nearly equal workload for each of the processors. There are many possible rules which could be used in the assignment of priorities for developing the program graph of a multiple processor system and to our knowledge no complete optimal solution to this problem has been proposed. Graph reduction beyond the cyclic to acyclic transformation may be a great help in the priority assignment. If all edges of a computation graph contained in a single non-branching chain are combined into an equivalent single edge with the weighting factors computed from the combination of weighting factors found in the original chain, a simpler and more realistic interpretation of the computing requirement is possible. In this reduced form of graph, decisions for priority assignment may be made largely on the basis of comparison between sets of single edges within the graph.

The optimization problem is certainly not a simple problem, but it is exceedingly interesting especially where the multiple processing capability exists. Once the criteria for optimization have been established, then certain methods based on the analysis of the computation and program graph are possible tools for optimization. The computation graph sets the scene and indicates what parallelism if any is possible. The graph reduction by cyclic to acyclic transformation and the reduction of serial non-branching chains to single elements considerably simplifies the analysis. Here there is certainly a large area of unexplored technical territory which needs further development.

SUMMARY AND CONCLUSIONS

The purpose of this paper was to discuss the problem of efficiency in the digital computer, considering the structure of cost of computation. Some ideas relating to methods of analysis of computational processes and methods of reducing computing cost were treated in a general and conceptual manner emphasizing possible fruitful avenues of detailed investigation.

Much work is still left unfinished. A better understanding of the algebra of relationships between the computation graph and the program graph is needed. Decision rules for priority assignment of multiple processors are also needed and optimization methods must be developed whereby the best use of computing hardware can be permitted.

REFERENCES

[1] ELMAGHRABY, S. E., "An Algebra for the Analysis of Generalized Activity Networks," Management Science, pp 495-514, April 1964.

[2] HOHN, F. E.: SESHU, S. and AFENKAMP, D. D., "The Theory of Nets," IRE Transactions of Electronic Computers, Vol. EC6, No. 3, pp 154-161, Sept. 1957.

[3] PRITSKER, A. B., ."Gert Graphical Evaluation and Review Technique," NASA Report RM4973, April 1966.

[4] PRITSKER, A. B. and DREZNER, S. M., "Network Analysis of a Countdown," NASA Report RM4976, March 1966.

[5] KARP, R. M., "A Note on the Application of Graph Theory to Digital Computer Programming," Information and Control, Vol. 3, pp 179-190, 1960.

[6] MARIMOUNT, R. B., "Applications of Graphs and Boolean Matrices to Computer Programming," SIAM Review, Vol. 3, No. 4, pp 259-268, Oct. 1960.

[7] MARTIN, D. F. and ESTRIN, G., "Experiments on Models of Computations and Systems," IEEE Transactions on Electronic Computers, Vol. EC-16, No. 1, pp 59-69, Feb. 1967.

[8] MARTIN, P. F. and ESTRIN, G., "Models of Computation Systems - Cyclic to Acyclic Transformations," IEEE Transactions on Electronic Computers, Vol. EC-16, No. 1, pp 70-79, Feb. 1967.

[9] MARTIN, D. F. and ESTRIN, G., "Models of Computation Systems - Evaluation of Vertex Probabilities in Graph Models of Computations," Journal of the Association for Computing Machinery, pp 282-299, Spring, 1967.

[10] ESTRIN, G. and TURN, R., "Automatic Assignment of Computations in a Variable Structure Computer System," IEEE Transactions on Electronic Computers, Vol. EC-12, pp 755-773, Sec. 1963.

[11] KIM, W. H. and CHIEN, R. T., Topological Analysis and Synthesis of Communications Networks, Columbia University Press, 1962.