SIGNIFICANCE OF PROBLEM SOLVING PARAMETERS ON THE PERFORMANCE OF COMBINATORIAL ALGORITHMS ON MULTI-COMPUTER PARALLEL ARCHITECTURES

F. Gail Gray, W. M. McCormack, Robert M. Haralick

Dept. of Computer Science and Dept. of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

ABSTRACT [1]

This experiment has determined an optimum problem solving strategy for the consistent labeling problem. One combination of factors, depth first search strategy-transmit large problems-transmit 50% of a processor's work, was found to be statistically best, especially for large problem sizes or for architectures with restricted communications paths. Future work involves experimentation to understand the architecture related factors. The results in this paper indicate that the performance of the system, even using the optimum problem solving strategy, will vary considerably with architecture.

I. INTRODUCTION

Combinatorial problem solving underlies numerous important problems in areas such as operations research, non-parametric statistics, graph theory, computer science, and artificial intelligence. Examples of specific combinatorial problems include, but are not limited to, various resource allocation problems, the travelling salesman problem, the relation homomorphism problem, the graph clique problem, the graph vertex cover problem, the graph independent set problem, the consistent labeling problem, and propositional logic problems [12-15]. These problems have the common feature that all known algorithms to solve them take, in the worst case, exponential time as problem size increases. They belong to the problem class NP.

This paper describes the interaction between specific algorithm parameters and the parallel computer architecture. The classes of architectures we consider are those which have inherent distributed control and whose connection structure is regular.

----

(1)This work was supported in part by the Office of Naval Research Grant N00014-80-C-0689.

Combinatorial problems require solutions which do searching. To help in describing the parallel combinatorial search, we associate with the space yet to be searched the term "the current problem." A representation mechanism which can partition the space yet to be searched can divide the current problem into mutually exclusive subproblems.

Now suppose that one processor in a parallel computer is given a combinatorial problem. In order to get other processors involved, the processor divides the problem into mutually exclusive subproblems and gives one subproblem to each of the neighboring processors, keeping one subproblem itself. At any moment in time each of the processors in the parallel computer network may be busy solving a subproblem or may be idle after having finished the subproblem on which it was working. At suitable occasions in the processing, a busy processor may notice that one of its neighbors is idle. On such an occasion the busy processor divides its current problem into two subproblems, hands one off to the idle neighbor and keeps one itself.

The key points of this description are

1. the capability of problem division
2. the ability of every processor to solve the entire problem alone, if it had to.
3. the ability of a busy processor to transfer a subproblem to an idle neighbor.

The parallel computer architecture research issue is: to determine that way of problem subdivision which maximizes computation efficiency for each way of arranging a given number of processors and their bus communication links.

To define this research issue precisely requires

1. that we have a systematic parametric way of describing processor/bus arrangements and

2. that we have alternative problem subdivision techniques.

This paper addresses the interaction between the processor/bus graph and problem size subdivision transfer mechanism. Once these relationships are determined and expressed mathematically, the parallel computer architecture design problem becomes less of an art and more of a mathematical optimization.

Our ultimate goal is to allow computer engineers to begin with the combinatorial problems of interest and determine via a mathematical optimization, the optimal parallel computer architecture to solve the problems assuming that the associated combinatorial algorithms are given.

II. PROCESSOR-BUS MODEL

In this section we discuss a processor-bus model which can be used to model all known regular parallel architectures [1,3,4,7,8,10,21-26]. The model does not currently include the general interconnection and shuffle type networks.

The graphical basis for the model is a connected regular bipartite graph. A graph is bipartite if its nodes can be partitioned into two disjointed subsets in such a way that all edges connect a node in one subset with a node in the second subset. A graph is connected if there is a path between every pair of nodes in the graph. A bipartite graph is regular if every node in the first set has the same degree and every node in the second set has the same degree. One subset of nodes represents the processor nodes and one subset represents the communication nodes in the parallel processing system. Every edge in the graph then connects a processing node to a communication node.

Any regular bipartite graph can be used to design a parallel computer structure by assigning the nodes in one set to be processors and the nodes in the other set to be communication links (or buses). Notice that theoretically either set of the bipartite graph could be the processor set. Therefore, each unlabeled bipartite graph represents two distinctly different computer architectures depending upon which set is considered to be processors and which set is considered to be the buses.

The notation $B(n_p,d_p,n_c,d_c)$ will be used to denote a regular bipartite graph which represents an architecture with n_p processors (each connected to d_p communication nodes) and n_c communication nodes (each servicing d_c processors). The Boolean 3-cube will then be represented by a graph $B(8,3,12,2)$. In general, the Boolean n-cube will be represented by a graph $B(2^n, n, n2^{n-1}, 2)$. Reversing the assignment of nodes to processors and buses produces the $B(12,2,8,3)$ graph which is called the p-cube by some investigators.

Other common architectures also have representations as bipartite graphs. For example, a planar array of size x^2 connected in the Von Neumann manner is modeled as a $B(x^2,4,2x^2,2)$ graph, the Moore connection results in a $B(x^2,8,4x^2,2)$ graph, the common bus architecture (or star) with x processors is a $B(x,1,1,x)$ graph, and the common ring architecture is a $B(x,2,x,2)$ graph. All existing architecures with regular local neighborhood interconnections can be modeled as a $B(n_p,d_p,n_c,d_c)$ graph.

III. PROBLEM SOLVING FACTORS

Introduction to Tree Searching

In order to make effective use of a multiple asynchronous processor for any problem, a major concern is how to distribute the work among the processors with a minimum of interprocessor communication. Kung [14] defines module granularity as the maximal amount of computational time a module can process without having to communicate. Large module granularity is better because it reduces the contention for the buses and reduces the amount of time a processor is either idle or sending or receiving work. Also, large granularity is usually better because of the typically fixed overhead associated with the synchronization of the multiple processors.

In the combinatorial tree search problems we are considering, module granularity as defined by Kung is not as meaningful because each processor could in fact solve the entire problem by itself without communicating to anybody. For our problem a more appropriate definition of module granularity might be the expected amount of processing time or the minimum amount of processing time before a processor splits its problem into two subproblems, one of which is given to an idle neighboring processor and one of which is kept itself.

When a processor has finished searching that portion of the tree required to solve its subproblem, it must wait for new work to be transferred from another processor. The amount of time a processor must wait before transmission begins and until transmission is completed is time wasted in the parallel environment that would not be lost in a single processor system. Thus, one must expect improvement in the time to completion to solve a problem in the multiple processor environment to be less than proportional to the number of processors. The factors that can affect the performance by either reducing the average transmission time or reducing the required number of transmissions include choice of algorithm, choice of search strategy, and choice of subproblems that busy processors transfer to idle processors.

Choice of Algorithm

In the single processor case, various algorithms have been proposed and studied to efficiently solve problems requiring tree searches. These usually involve investing an additional amount of computation at one node in the tree in order to prune the tree early and avoid needless backtracking. In work on constraint satisfaction [11], the forward checking pruning algorithm was found to perform the best of the six tested and backtracking the worst.

For the same reasons, it seems clear that pruning the tree early should be carried over to a multiple processor system to reduce the amount of computation necessary to solve the problem. There are other reasons as well. Failure to prune the tree early may later result in transfers to idle processors of problems which will be very quickly completed. Since a transfer ties up, to some extent, both the sending and receiving processor, time is lost doing the communication and the processor receiving the problem would shortly become idle.

We would, therefore, expect that in the multiple processor environment the forward checking pruning algorithm for constraint satisfaction would work much better than backtracking. However, in the uniprocessor environment Haralick and Elliott also showed that too much look ahead computation at a node in the search could actually increase the problem completion time. It is not clear that this would be true in the multiple processor case. It may be best to do more testing early reducing future transfers, communication overhead, and delay in contrast to the single processor case where only some extra testing has been found to be worthwhile.

A second consideration in the selection of a search algorithm is the amount of information that must be transferred to an idle processor to specify a subproblem and any associated lookahead information already obtained pertinent to the subproblem. In most cases this is proportional (or inversely proportional) to the complexity of the problem remaining to be solved. Thus the transmission time will be a function of the problem complexity. Backtracking requires very little information to be passed while, for forward checking, a table of labels yet to be eliminated must be sent.

Search Strategy

Search strategy is a second factor of importance to the multiple processor environment. When a problem involves finding all solutions, like the consistent labeling problem, the entire tree must be searched. Thus, in a uniprocessor system the particular order in which the search is conducted, i.e., depth first or breadth first, has no effect. In a multiple processor system, however, this is a critical factor because it directly affects the complexity of the problems remaining in the tree to be solved and available to be sent to idle processors from busy processors.

A depth first search will leave high complexity problems to be solved later (that is, problems near the root of the tree.) This would seem to be desirable in the multiple processor environment because passing such a problem to an idle processor would increase the length of time the processor could work before going idle and thereby reduce the need for communication. On the other hand, a breadth first search would tend to produce problems of approximately the same size. Since the problem is not completed until all processors are finished, the breadth first strategy might be preferable if it results in all processors finishing at about the same time. It might be that the best approach could be some combination of the two; for example, one might follow a depth first strategy for a certain number of levels, then go breadth first to a certain depth, and then continue depth first again.

Problem Passing Strategy

A factor closely related to the search strategy occurs when a processor has a number of problems of various complexities to send to an idle processor. The optimization question is how many should be sent and of what complexity(ies). Further complicating this is a situation where the processor is aware of more than one idle processor. In such a situation, how should the available work be divided and still leave a significant amount for the sending processor?

Further complicating this question is the fact that the overhead involved in synchronizing the various processors and transmitting problems to idle ones will eventually reach a point where it will be more than the amount of work left to be done. An analogous situation exists in sorting; fast versions of QUICKSORT eventually resort to a simple sort when the amount remaining to be sorted is small [13].

In this case, it would appear that a point will eventually be reached where it is more effective for a processor simply to complete the problem itself rather than transmit parts of it to others. Determination of this point will depend on the depth in the tree of the problem to be solved and the amount of information that must be passed (which depends on the lookahead algorithm being used.)

Processor Intercommunication

One decision that has to be made is how the need to transfer work is recognized. Specifically, does a processor which has no further work interrupt a busy processor, or does a processor with extra work poll its neighboring processors to see if they are idle.

The advantage of interrupts is that as soon as a processor needs work, it can notify another processor instead of waiting to be polled. This assumes, however, that a processor would service the interrupt immediately instead of waiting until it had finished its current work. A disadvantage is that when a processor goes idle, it cannot know which of its neighbors to interrupt. Using polling, an idle processor can be sent work by any available neighboring processor instead of being forced to choose and interrupt one. In addition, although an interrupted processor may be working or transmitting (a logical and necessary condition) when interrupted, it may not have a problem to pass when it is time to pass work to the interrupting processor. In fact, the interrupted processor could itself go idle. For these reasons the simulation we discuss in section IV uses polling. Whenever a processor completes a node in the tree, and as long as it has work it could transfer, it checks each neighboring CPU and the connecting bus. If both are idle, a transfer is made.

IV. SIMULATION EXPERIMENTS

In order to better understand the behavior of the tightly coupled asynchronous parallel computer, we have designed a series of simulation experiments using the consistent labeling constraint satisfaction problem. The simulation used to perform these experiments was written in SIMULA [Birtwistle,, Myhrhaug & Nygaard, 1973]. Let U and L be finite sets Let $R \subseteq (U \times L)^2$. We use the simulated parallel computer to find all functions $f: U \rightarrow L$ satisfying that for all $(u,v) \in U \times U, (u, f(u), v, f(v)) \in R$. The goal of the experiments is to determine which architectural and which problem related factors are significant enough to warrant further investigation. This paper presents the results for problem related factors.

In this experiment each problem factor was tested at two levels. The factors and levels tested are given in Table 1. Based on previous experiments [16], it was very clear that forward-checking was significantly better than backtracking so all experiments used the forward-checking algorithm [11]. In order that the results be applicable for different architectures and problem sizes, two problem sizes (small and medium) and two very different architectures (in terms of the number of communication paths) were used. The architectures chosen were symmetric to eliminate the need for assumptions about the architecture related factors discussed earlier. The ring architecture, B(64, 2, 64, 2), due to the limited interconnection structure, will have difficulty passing work from the initial processor to distant processors. The Boolean 6-cube B(64, 6, 192, 2), should be able to effectively utilize most of the 64 processors. Finally, one replication was run of each combination. This involves running the simulation with different random number seeds to create statistically equivalent combinatorial problems. An analysis of variance was used to determine the significance of the problem related parameters and to determine interactions of the parameters [20]. The measure of performance used was the time until the problem was solved.

Results

The analysis of variance was done using the SAS (Statistical Analysis System) package. The analysis showed statistically significant differences in the means (at a level of 0.0001), and second and third order interactions for the search strategy, size passed, and number passed. The means for the two cutoff point levels were not statistically different. Because the three way interaction among strategy, size, and number was significant, the combinations of these three factors were treated as eight levels of one combined factor for further analysis.

Duncan's multiple range test was performed [20] to divide the levels into groups with similar performance. The results, based on the average time to completion for the different experimental conditions, are shown in Table 2.

The key result is that one combination is clearly superior, depth-large-50%, and should be used in further experiments. (This combination also produced the lowest mean for each of the four architecture-problem size pairs.)

There is a logical explanation for the groupings. For each factor one value can be classified as positive (i.e., it should contribute to improved performance regardless of other factors), and the other negative (i.e., it should result in poorer performance). The positive factors are indicated as level 1 in Table 1. For example, passing more than one sub-problem or passing large sub-problems should be preferable as the idle processor should stay busy longer. Since in a depth first search a processor works on small problems, this should leave larger problems to pass. As a result communication time is reduced.

Using this idea of a positive level for each factor, only one combination has all 3 levels positive, three have two positive, three have one positive, and one no positive levels. The grouping produced by Duncan's test confirms this analysis and, in fact, produces a finer partition. Thus, the interaction between these factors agrees with the analysis. The analysis of variance also indicated significant interactions between the combined factor and the experimental conditions of problem size and architecture. To best understand these interactions, the values were plotted as suggested by Cox [6]. (Figures 1,2,3). If there were no interaction, then the curves in each figure would be parallel.

Figure 1 shows a clear interaction between problem size and architecture. For a small problem, a small number of processors is sufficient; thus, the inability of the ring to spread sub-problems to idle processors is not a severe handicap. However, for a larger problem, the performance of the ring is much worse than that of the 6-cube which is able to involve many more of the processors. In each case the time to completion was approximately 3 times longer in the ring architecture. Since the degree of each processor node in the ring is 1/3 of the degree of each processor node in the Boolean 6-cube, it appears that performance _may be_ proportional to the degree of the processor nodes. This has intuitive appeal because more communication paths should improve the ability of processors to keep busy. Later experiments will confirm or deny this conjecture. It is also possible that diminishing returns may set in for extremely large numbers of communication nodes. This plot indicates that the use of an optimum architecture becomes more crucial for large problems.

Figure 2 shows the interaction of the combined problem solving factor with problem size. Clearly, the need to determine the best combinations of problem solving factors becomes more critical as the size of the problem increases because a bad choice has a greater detrimental effect on the larger problem.

Figure 3 shows the interactions of the combined problem solving factor with architecture type. This plot shows that an optimum choice of problem-solving factors tends to reduce the effects of a bad choice of architecture. However, the difference in performance between the architectures using the optimum problem solving strategy is still a factor of 3, so that further experiments to determine an optimum architecture seem justifiable.

REFERENCES

(1) Anderson, G. A., and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", _Computing Surveys_, Vol. 7, Dec. 1975, pp. 197-213.

(2) Armstrong, J. R. and F. G. Gray, "Some Fault Tolerant Properties of the Boolean n-Cube", _Proceedings of the 1980 Conference on Information Sciences and Systems_, Princeton, NJ, March 26-28, 1980, pp. 541-544.

(3) Benes, V. E., "Optimal Rearrangeable Multistage Connecting Networks", _Bell System Technical Journal_, July 1964, pp. 1641-1656.

(4) Batcher, K. E., "Sorting Networks and Their Applications", _Spring Joint Computer Conference_, 1968, pp. 307-314.

(5) Birtwistle, G. M., Dahl, O. J., B. Myhrhaug, and K. Nygaard, _SIMULA Begin_, Auerbach Publishers Inc., Philadelphia, PA, 1973.

(6) Cox, D. R., Planning of Experiments, John Wiley & Sons, Inc., New York, 1958.

(7) Despain, A. M. and D. A. Patterson, "X-Tree: A Tree Structured Multiprocessor Computer Architecture", 5th Annual Symposium on Computer Architectura, architecture, 1978, pp. 144-151.

(8) Finkel, R. A. and M. A. Solomon, "Processor Interconnection Strategies", IEEE Transactions on Computers, Vol. C-29, May 1980, pp. 360-370.

(9) Foster, M. J. and H. T. Kung, "The Design of Special Purpose VLSI Chips", Computer, Jan. 1980.

(10) Goke, R. L. and B. S. Lipovski, "Banyon Networks for Partitioning Multiprocessor Systems", Proceedings of First Conference on Computer Architecture, 1974, pp. 21-28.

(11) Haralick, Robert M. and G. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence, Vol. 14, 1980, pp. 263-313.

(12) Hillier, F. S. and G. S. Lieberman, Operations Research, Holden Day, Inc., San Francisco, 1979.

(13) Knuth, D. E., The Art of Computer Programming, Sorting and Searching, Addison-Wesley Publishing, Reading, MA, 1973.

(14) Kung, H. T., "The Structure of Parallel Algorithms", in Advances in Computers, Vol. 19, edited by M. D. Yovits, Academic Press, 1980.

(15) Lee, R. B., "Empirical Results on the Speed, Redundancy and Quality of Parallel Computations", Proceedings of 1980 International Conference on Parallel Processing, 1980.

(16) McCormack, W. H., F. G. Gray, J. G. Tront, R. M. Haralick and G. S. Fowler, "Multi-Computer Parallel Architectures for Solving Combinatorial Problems", Multi-Computer Architectures and Image Processings: Algorithms and Programs, Academic Press, New York, 1982.

(17) Mead, C. A. and M. Rem, "Cost and Performance of VLSI Computing Structures", IEEE J. Solid State Circuits, sc-14(2), pp. 455-462, 1979.

(18) Mead, C. A. and L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980.

(19) Mirza, J. H., "Performance Evaluation of Pipeline Architectures", Proceedings of 1980 International Conference on Parallel Processing, 1980.

(20) Ott, Lyman, An Introduction to Statistical Methods and Data Analysis, Duxbury Press, North Scituate, MA, 1977.

Table 1 - Experiment Summary

FACTORS TESTED

FACTOR	LEVEL 1	LEVEL 2
search strategy	depth-first	breadth-first
size of sub-problem passed	largest	smallest
number of sub-problems passed	50% of expected total work	1 sub-problem
cutoff point	none	4 units to be tested
EXPERIMENTAL CONDITIONS		
Architecture	Ring	6-cube
number of processors	64	64
number of buses	64	192
Size of combinatorial problem	small - 12 units & labels	medium - 16 units & labels
One replication	random	random

Table 2 - Duncan's Multiple Range Test

GROUPING*	MEAN COMPLETION TIME	ID NUMBER	FACTOR COMBINATION		
			SEARCH	SIZE	NUMBER
A	2,705,274	8	breadth	small	one
B	1,874,887	7	breadth	large	one
C	689,372	4	depth	small	one
D	451,133	6	breadth	small	50%
E	335,267	5	breadth	large	50%
F E	301,774	3	depth	large	one
F	247,667	2	depth	small	50%
G	147,181	1	depth	large	50%

*means with the same grouping are not significantly different
significance level = 0.05

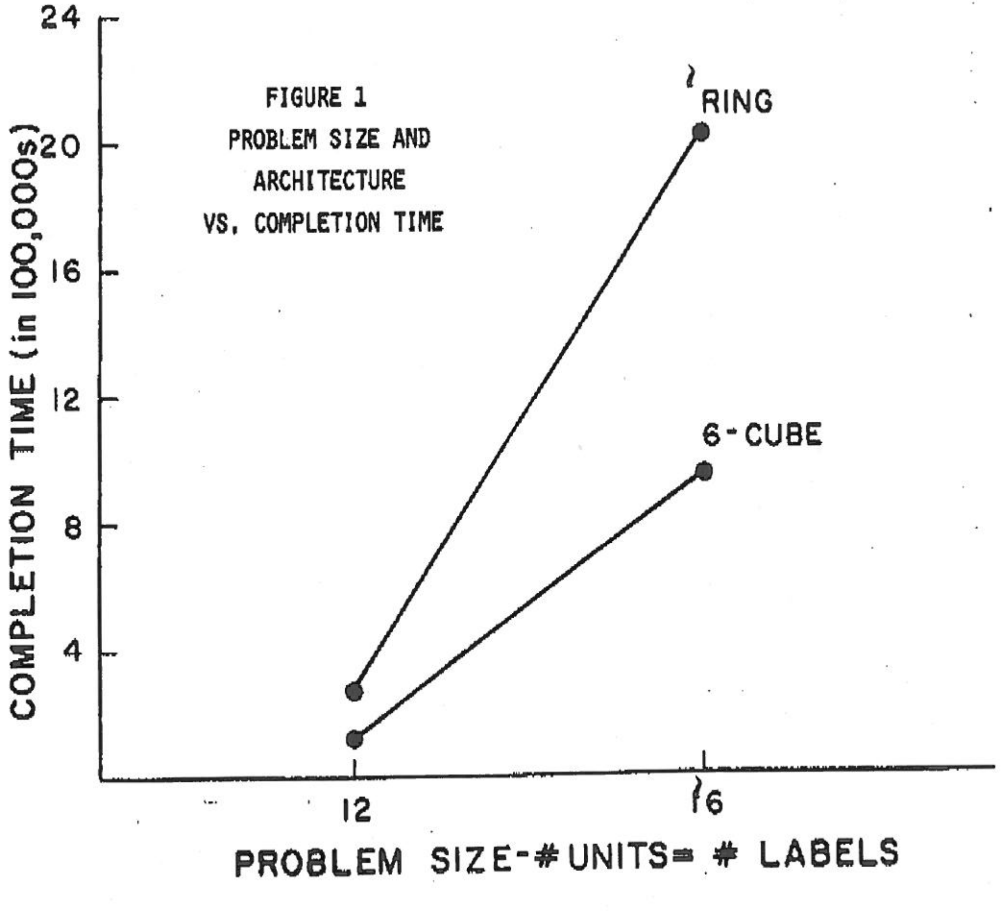

FIGURE 1
PROBLEM SIZE AND
ARCHITECTURE
VS. COMPLETION TIME

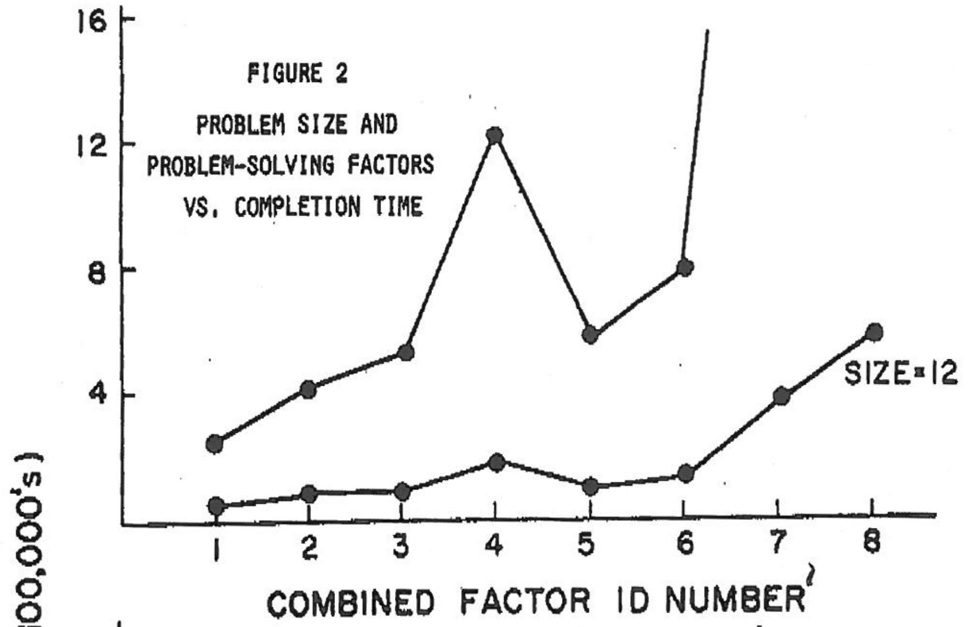

FIGURE 2
PROBLEM SIZE AND
PROBLEM-SOLVING FACTORS
VS. COMPLETION TIME

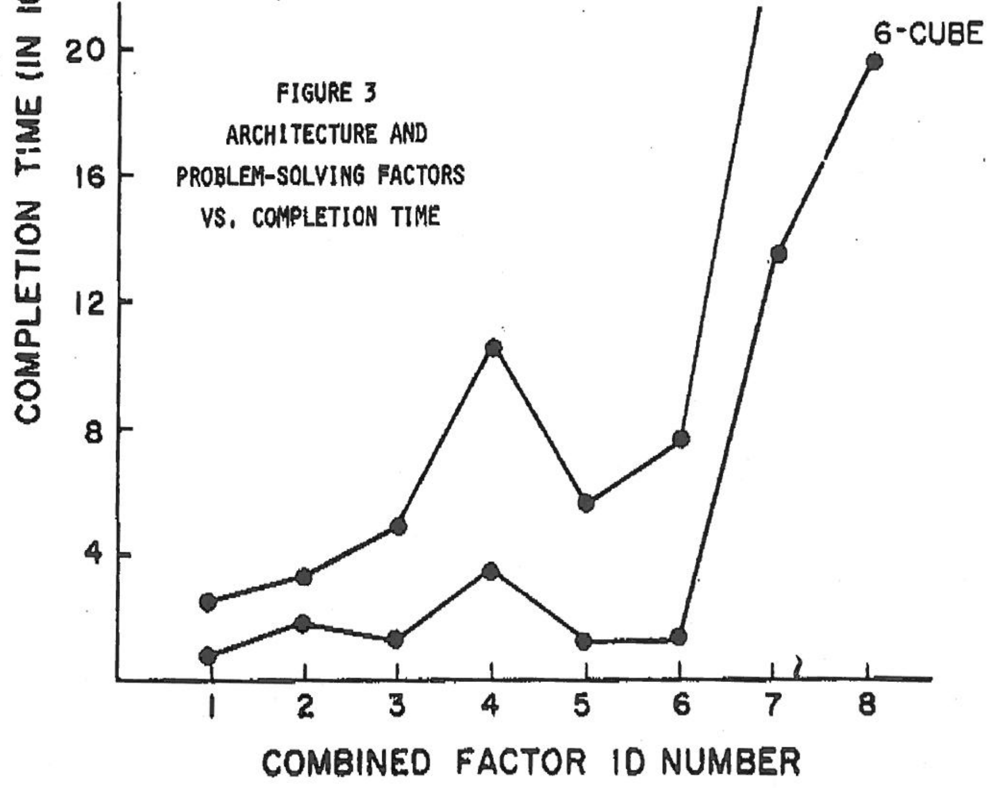

FIGURE 3
ARCHITECTURE AND
PROBLEM-SOLVING FACTORS
VS. COMPLETION TIME