A SIMULATION MODEL OF A MULTI-COMPUTER SYSTEM SOLVING A COMBINATORIAL PROBLEM[1]

William M. McCormack,[2] F. Gail Gray, Robert M. Haralick
Department of Computer Science, Virginia Tech
Blacksburg, Virginia 24061

Abstract

This paper describes a simulation of a multi-computer system used to solve computationally intensive combinatorial problems. The model is traced from its conception through to the initial experiments conducted using it and their results. Included are descriptions of the combinatorial problems being solved, the language selection process, the simulated multi-computer system, and the model validation performed.

1. Introduction

For many computationally intensive problems, the most promising direction for improving the speed with which solutions can be obtained and for increasing the size of problems solved is to use many processors working together on the solution of a single problem (5). Combinational problems, which occur in numerous areas of applied mathematics, are one example of computationally intensive problems.

The paper discusses the simulation of a multi-computer system solving combinatorial problems. Simulation is being used to investigate the factors affecting the efficiency of such a system prior to actual implementation. The development of the model through its initial use is described. The next section provides additional background on the problem being studied. In the following sections the language selection process, simulated system, validation process, and first experiment are described. The paper concludes with an evaluation of the simulation and discussion of changes being made to it.

---

[1]Supported in part by the office of Naval Research under Grant N0014-80-C-0689.

[2]Now with Amdahl Corporation, P.O. Box 470, Sunnyvale, CA 94086.

Proceedings of the 1982
Winter Simulation Conference
Highland * Chao * Madrigal, Editors

82CH1844-0/82/0000-0261 $00.75 © 1982 IEEE

2. Problem Solving on a Multi-Computer System

Combinatorial problem solving underlies numerous important problems in areas such as operations research, non-parametic 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 (6,7,8,10). 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.

Combinatorial problems require solutions which do searching. In a very natural way, the algorithm doing the tree search keeps track of what part of the search space has been examined so far and what part of the search is yet to be examined. The mechanism, which represents the division between that which has been searched so far and that which is yet to be searched, can also be used to partition the space which is yet to be searched into two or more mutually exclusive pieces. This is precisely the mechanism which can let a combinatorial problem be solved in an asychronous parallel computer.

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 divides its current problem into two subproblems, hands one off to the idle neighbor and keeps one itself.

The research issues that arise are: what effect does the number of processors and buses have, what is the most efficient way to arrange a given number of processors and buses, and what effect do alternative problem solving techniques and problem passing strategies have on performance.

3. Simulation Language Selection

To put the language selection process in perspective, when the project began only general ideas about the system to be simulated were available. The physical system was to be modeled, but only in enough detail to study processor communication, utilization and bus contention. In addition, it would be necessary to simulate the workload for the system.

Emshoff and Sisson (2) discuss two considerations in the selection of a language: operational characteristics (e.g. availability, costs) and problem-oriented characteristics (world view, flexibility, random variate generators, etc.). At Virginia Tech most general purpose and simulation languages were available; however the simulation languages were only available on the university's configuration and not department computers for which there would be no charges. Ultimately, the chief considerations in the selection were the type of system to be simulated and the expected complexity of the task. By using a simulation language, the need to develop basic simulation tools would be eliminated and full attention could be focused on the problem itself. Secondly, since the basic entity in the simulation would be a processor solving a problem and communicating with other processors, it would be much easier to model the system using a process world view. Because of these, the decision was made to use a simulation language supporting a process world view over using a general purpose language with an event world view.

Once the type of language had been chosen, the actual choice of language was straightforward. GPSS is not flexible enough and numerous changes were expected to be necessary to the initial design. The choice then was either Simscript or Simula, both known to the author and available. Simula was selected since the process view is a natural part of the language and was not added later. In addition the clear programming structure of Simula would mean later modifications should be easily and quickly done. This was indeed the case. Major changes were made on a number of occasions as new ideas were found to be worthy of investigation. Some of the ideas were a direct result of being able to focus on the problem by using a process view language.

4. Simulation Model

It is easiest to describe the simulation model by dividing it into three parts. These are specification of the physical system, specification of the combinatorial problem to be solved by the system, and how the relation of the two result in numerous factors affecting problem solving on a multi-computer system. The program collected various statistics to measure the performance of the system, to aid in understanding results, and for use in validating the model. Some of these statistics were the time the problem was completely solved (the key performance measure), counts of subproblems passed between processors, the percentage of time each processor was in each state (busy, idle, etc.), and usage statistics for each bus.

4.1 Physical System Specification

The processor/bus configuration can be described using a labeled bipartite graph. The nodes are either labeled as being a processor or as being a bus. A link between a pair of nodes means that the processor node is connected to the bus node. In the model, it was restricted that each processor (bus) be connected to an equal number of buses (processors). For a given number of processors and buses, many interconnection configurations are possible and can affect system performance. Separate work is being done to generate all graphs of given characteristics and will provide input to the simulation model.

Each processor is assumed to be the same and have its own local memory. Thus, there is no memory contention, but contention can occur for buses shared by processors. A processor can be in one of four states: working on a portion of the combinatorial problem; sending a subproblem to another processor; receiving a subproblem from a processor, and idle. A working processor with extra subproblems periodically checks neighboring processors and, if one is found idle, initiates a subproblem transfer.

4.2 Problem Specification

The combinatorial problem solved by the simulated system is the consistent labeling problem which arises in artificial intelligence problems including scene analysis, graph coloring, and proposition theorem proving (4). The problem is to determine all possible labelings (assignment of L labels to U units) without violating consistency constraints. The constraints are typically constraints on pairs of unit-labels, e.g. whether unit 2 having label A is consistent with unit 4 having label B. The N-queens problem, placing N queens on an N by N chessboard so that no capture is possible, can be formulated as a consistent labeling problem.

Besides specifying U and L, it is necessary to specify the constraints. Two ways are use an actual problem (such as the N-queens) or randomly assign which constraints are true and which are false. Both of these require, when doing the tree search, to keep numerous tables and to do all necessary constraint checks to test a partial labeling of the units. A third way is mathematically solve the random constraint version. This permits simulation of the problem testing which results in considerable savings of execution time and memory for large values of U and L.

4.3 Factors Affecting Problem Solving

The factors that can affect problem solving on a multi-computer system can be divided into two classes, those independent of the architecture and those related to it. Some of these were only uncovered while testing the simulation and it is not yet clear how major an effect they will have.

4.2.1 Architecture Independent Factors

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 the consistent labeling problem (4), the forward checking pruning algorithm was found to perform the best of six tested and standard 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. Only forward checking and backtracking have been implemented so far, since only for those algorithms have mathematical solutions to the random constraint problem been obtained.

Search strategy, a second important factor, explains the action taken by a processor after testing a partial labeling - does it next test another label for the most recent unit in the labeling (a breadth-first search) or does it add the next unit to the labeling and test a label for it (a depth-first search)?

When a problem involves finding all solutions, like the consistent labeling problem, the entire tree must be searched. In a uniprocessor system the particular order in which the search is conducted, i.e., depth first of 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 large problems to be solved later (that is, problems near the root of the tree), whereas breadth first search would tend to produce problems of approximately the same size.

The other factors concern a processor that has a number of problems of various complexities to send an idle processor. The question is how many should be sent and of what complexity(ies), especially in 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?

Furthermore, the overhead involved in synchronizing the various processors and transmitting problems to idle ones could eventually reach a point where it will be more than the amount of work left to be done (an analogous situation exists in sorting (7)). In this case, it would appear that a point could be reached where it is more effective for a processor simply to complete the problem itself rather than transmit parts of it to others.

4.2.2. Architecture Dependent Factors

It is clear that the architecture configuration will affect performance (how much is uncertain for a fixed number of processors), but two additional factors arise if the architecture is asymmetric. One is the initial processor to receive the problem. The multi-computer system is envisioned as a special purpose one attached to another system (e.g. a VAX 11/780). The problem is given to one processor initially which, as subproblems are created, distributes it to other processors. It is anticipated the effect of the initial processor will diminish (much like a transient of a simulation) as the problem sizes tested are increased.

The second factor involves a processor with extra subproblems that has more than one idle processor as a neighbor. In the simulation, subproblems are passed to only one processor at a time, thus it is necessary to have a rule to decide to which processor to send work first.

5. Validation

The validation of the model used techniques described by Sargent (12); however many of the techniques could not be used since there was no actual data with which to compare. The major objective was to gain confidence in the model's ability to predict the relative performance of different systems. The principal methods used were internal validity, parameter variability and sensitivity analysis, comparison to other models, and face validity. These were employed after hand-tracing small systems to help insure the correctness of the program.

The same system was to run with different random number seeds to see how much variability there was in results (internal validity check). There was very little which was expected due to the size of the problems tested. This was also important as it meant fewer replications would be needed when conducting experiments.

A second test was of the system's sensitivity to some of the parameters chosen to represent the speed of the processors and buses. A range of reasonable values were tried and, although there were some changes in absolute results, there were no changes in relative performance. Other parameters were varied to see if the results produced (e.g. problem completion time, processors' utilization, etc.) changed as expected. In a few cases this was not initially true, but after further tests and analysis, it was apparent the model was correct and our intuition was in error.

The other major way used to validate the model was to compare the performance measures for the simulation model to the measures obtained using other models and submodels. The effect of simulating the problem solving was tested by comparing the results of simulating the testing and actually

doing the tests for the random constraint version. The closeness of the results confirmed earlier results, albeit on a single processor system (4), of the validity of simulating the testing. Finally, statistics gathered on the work done by the system and the tree search were in agreement with analytical results obtained.

Thus, through these steps confidence was achieved in the ability of the model to accurately predict the performance of the system under a variety of configurations, workloads, and parameter values.

6. First Experiment

Because of the large number of factors, it was decided to concentrate on those factors independent of the architecture. This was done because these factors were better understood and because of the large number of architectures we wished to test. The goal was to select the best combination of the problem-solving factors and to understand the interactions between them before proceeding to the architectural features and factors. This experiment is described in detail in (3). The model is a terminating simulation (9), the terminating condition is when the problem has been completed (all labelings have been found); thus, there is no initial transient to remove. The performance measure is the time the simulation terminates.

In this experiment each problem factor was tested at two levels in a full factorial design. The factors and levels tested are given in Table 1. From previous testing and validation it was very clear that forward checking was significantly better than backtracking, so all experiments used the forward checking algorithm. 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 (1) were symmetric to eliminate the need for assumptions about the architecture related factors discussed earlier. The ring architecture, due to the limited interconnection structure, will have difficulty passing work from the initial processor to distant processors. The Boolean 6-cube, however, should be able to effectively utilize most of the 64 processors.

Finally, one replication was run for each combination. More would have been desirable, but the simulation was very costly and time consuming to run. From earlier testing and validation it was felt performance differences would be much greater than the random error. This was found to be the case.

Analysis of variance was used to determine the significance of the problem related parameters and to determine interactions of the parameters (11). The analysis showed statistically significant differences in the means (at a level of Ø.ØØØ1) for

TABLE 1 - EXPERIMENT SUMMARY

FACTORS TESTED

FACTOR	LEVEL 1	LEVEL 2
search strategy	depth-first	breadth-first
size of sub-problem passed	largest	smallest
no. of sub-problems passed	50% of total	1 sub-problem
cutoff point	none	4 units left

EXPERIMENTAL CONDITIONS

Architecture	Ring	6-cube
no. of processors	64	64
no. of buses	64	192
Size of problem (no. of units and labels)	12 (small)	16 (medium)

all main effects, 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 (11) to divide the levels into groups with similar performance. This test showed that one combination is clearly superior, depth-large-50%, and should be used in further experiments. There is a logical explanation for this result. 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. Using this idea of a positive level for each factor, only one combination has all 3 levels positive and this was found to be the optimal combination.

The analysis of variance also indicated significant interactions between the combined factor and the experimental conditions of problem size and arhitecture. These could be easily explained and clearly showed the importance of the correct problem solving combination on performance (3).

7. Evaluation

This paper has described the simulation process, from design to initial experiment, used to analyze the performance of a multi-computer system. It seems appropriate, therefore, to evaluate the effectiveness of the simulation in light of subsequent developments.

The simulation already has proved invaluable in understanding the system to be built, even though experiments are just beginning. Testing with it is allowing critical decisions, and mistakes, to be made now prior to implementation. Simula worked quite well by allowing concentration on the model and its flexibility and structure allowed the necessary changes to be made easily.

Unfortunately, when the experiments began problems were discovered which have resulted in a change in the language used for the simulation. Because of the number of cases to test, it was convenient to do many experiments in one run. This requires leaving and reentering the Simula "simulation class" to reset the clock, empty sequencing set, etc. to do the next experiment. This would usually result in a runtime error for the large problems on our system. At this point it was discovered that Simula was no longer being maintained by our computing center. Thus, it was necessary to run the experiments, and collect the results one by one.

In addition, the next experiment planned would be much more expensive due to the variety of architectures to be tested and the architecture related factors. Finally, the architecture-graph-generator program, which provides much of the simulation's input, had been moved to a different system (VAX), in part for economic reasons, and there was no easy communication between it and the university's IBM system.

Thus, the decision was made to convert the entire program (about 2000 lines) into Pascal to run on the department VAX. This would eliminate costs to run, allow multiple experiments with one run, and immediate access to the input architectures. Because Simula and Pascal are similar and the code was very modular, conversion to Pascal was not too difficult. It was only necessary to write a few random variate generators, a time flow mechanism, future event list handler, and turn the process world view into an event view.

Despite this conversion, the choice of Simula was still correct as the system was not understood well enough earlier to begin with Pascal and an event world view. The conversion and testing are complete and further experiments are underway.

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. Emshoff, J.R. and R.L. Sisson, Design and Use of Computer Simulation Models, MacMillan Publishing Co., Inc., New York, 1970.

3. Gray, F.G., W.M. McCormack and R.M. Haralick, "Significance of Problem Solving Parameters on the Performance of Combinatorial Algorithms on Multi-Computer Parallel Architectures," 1982 International Conference on Parallel Processing, 1982.

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

5. Haynes, L.S., "Highly Parallel Computing: Guest Editor's Introduction", Computer, Jan. 1982.

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

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

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

9. Law, A.M., "Statistical Analysis of the Output Data from Terminating Simulations", Naval Research Logistics Quarterly, Vol. 27, March 1980, pp. 131-143.

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

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

12. Sargent, R.G., "Validation of Simulation Models", Proc. Winter Simulation Conference, 1979, pp. 497-503.