A SYSTEM ORIENTED BENCHMARK FOR TRAINING SIMULATORS

P. S. BABEL and DR. M. L. BIRNS
Aeronautical Systems Division
Wright-Patterson Air Force Base

1. INTRODUCTION

An important task in the development of training simulators is determining the computing system which is adequate for the computational task required. The definition of the simulator computational system as well as ensuring its adequacy has been rendered more difficult by the increasing complexity of available computer systems, the ever changing computer systems market and the growing sophistication of the training simulator computation requirement.

Several selection tools have been previously available, but are generally CPU rather than system oriented. Since these computer selection tools do not address simulation processing parameters, they are of limited utility in selecting computer systems for this application. Hence, the Air Force is striving to develop a computational system selection technique which is based on training simulator requirements and is flexible enough to be used in the various types of computational systems presently found in training simulators. The tool required is a system benchmark which will measure the total capability of the computational systems.

2. BACKGROUND

The digital computation systems originally provided for training simulators were single CPU computer systems. These computers were programmed in assembly language and required relatively slow responses. By relatively slow, we refer to a comparison with the modern training simulator and the necessity to drive an advanced visual system.

The design for these computational systems was initially based on deriving, from the functional requirements of the system, the high-level flow chart for the program and estimating the numbers and types of instructions required. Calculation of a data base was then added to determine the memory requirements of the system, the speed being defined and verified by the estimated calculational requirements.

As design became somewhat more advanced, some scientific mixes of instructions were defined for the typical calculational program, such as the classical gibson mix, and the adequacy of the computational power was determined by measuring the capability of a CPU to handle the assumedly representative instructions repertoire.

Historically, hardware has been evaluated. Williams[1], as well as Stimler and Bruns[2], gives examples of techniques for calculating computer performance and selecting an appropriate machine.

Lucas[3] lists and describes the various techniques used for performance evaluation:

- Timings
- Mixes
- Kernels
- Models
- Benchmarks
- Synthetic Programs
- Simulation

He concludes that simulation is the optimum technique, but recognizes disadvantages, including cost. Boyse and Warn,[4] as well as many others, have described computer modeling for performance prediction. They make the comment, which may be self-evident, but is important, "Modeling can be useful, but any model can easily produce invalid results if its input data or assumptions are incorrect."

Progress was made as more experience was gained by being able to use analogy from one system to another. Based on previous experience, it was possible to get a gross estimate of the numbers of instructions and data required for various simulator functions by using approximations from one simulator system to the next. Similarly some benchmarks were derived based on the actual software of previous simulators. The benchmarks presently available give some indication of comparative machine power, but they are necessarily limited in the amount of functions they can replicate and are in general limited to single CPU investigations.

With the passage of time, the aircraft for which the Air Force must train its pilots have become increasingly complex and the training simulator has followed them in sophistication. This increased sophistication is brought about not only by the increased complexity of the aircraft, e.g., multiple on-board computers, but has also been brought about by the desire to provide more sophisticated training capability.

The expanded use of software to achieve these levels of simulation together with a significant increase in computer performance capabilities has facilitated use of FORTRAN and commercial software. Use of these software tools enhances software supportability. Simultaneously, the increased complexity of the simulation job has brought about the need for multiple CPU computational systems which utilize shared memory, inter-computer channels and sophisticated input/output devices for their recording as well as occasional overlays.

Computer systems design and verification, once performed by manual computations augmented by heavy doses of intuition, can no longer be handled satisfactorily without an automated tool which provides the ability to judge system capability objectively. Note, we talk about systems capability, not computer or CPU capability, since much more than the capability of the CPU is involved. The computational system must be judged on its practical capability, throughput capability, configurational capability, as well as the important software tools such as the operating system, compiler and utilities which form an important part of the total computational systems.

Lucas emphasizes the importance of the programming system stating, "Because the programming system is an integral part of modern computers, the evaluation process must now consider software as well as hardware in assessing performance."

Thus, it is no longer sufficient to attempt to estimate the capacity of a data processing system in terms of a number of average instructions or operations. It has become essential to have a tool which measures all the critical system parameters and which is flexible enough to put appropriate emphasis on those which are most important for a specific training simulator. This leads to the idea of a system oriented training simulator benchmark.

So called "natural" benchmarks also pose some problems. Keller and Denham[5] comment on the difficulty of running a benchmark on all systems, exactly as configured in proposals. This, however, is required if valid results are to be obtained.

An implication of the "system benchmark" concept is provided in Joslin,[6] who also defines an Application Benchmark (Problem) to be "a routine to be run on several different computer configurations to obtain comparative throughput performance figures regarding the abilities of the various configurations to handle the specific application." This implies that the specific routine to be executed is an integral part of the benchmark problem.

As we are interested in allowing varying approaches to a specific functional requirement, we feel that it is more appropriate to start from the functional definition of the set of tasks to be performed. The specific processing design is assumed to be dependent upon the computational system configuration to be used.

The advantages of using totally synthetic programs rather than natural programs are discussed in Oliver, et al.[7] Primary advantages are portability and limitation of processing time. A disadvantage is the time and cost required to prepare programs.

We intend to circumvent these difficulties by providing synthetic, but representative, operational designs, implemented primarily by consistent blocks of FORTRAN code. The constraint of FORTRAN is imposed because of the de facto standardization of FORTRAN as a training simulator language as previously discussed by Babel.[8]

3. MEASUREMENT

The system benchmark must measure both system parameters and machine parameters and must weight these parameters such that the results indicate the capability of the system to do the entire job. System parameters are those system features which impact the entire computational system, not specifically the computational capability of CPU. The system parameters include:

- Operating System Capability
- Operating System Efficiency
- Fortran Efficiency (compiled object code)
- I/O Capability
- Multiprocessing Capability
- Systems Extendability

Naturally, the most important part of the benchmark is to measure the actual computational power of the system. This must be measured across more than one CPU if a multi-CPU system is envisioned. For example, a distributed CPU system may indeed have more computational power than that provided by a single, larger CPU. However, the power of the distributed system does not increase linearly and the benchmark must measure not only the power of the single CPU, but the extent to which processing power is increased by additional CPUs.

Thus, the benchmark must examine such traditional parameters as handling of specific data types, arithmetic capabilities (particularly floating point), general

functional capabilities and logical functional capabilities. It must also be capable of examining such nonclassical capabilities as the interference between CPUs operating on a shared memory, the possibility for distributing computational power, the effect of inter-computer communication, both on computational power and I/O capacity. Finally, the benchmark must be able to assess the capability of the system to perform all these functions in the FORTRAN environment, with respect to the effect on both processing time, memory requirements, and codeability with proposed extensions.

Since many of the parameters which are being measured are intimate features of the specific systems being investigated, it may not be possible to code a particular benchmark program and execute it on all machines. FORTRAN extensions for different systems are different, operating systems have different requirements, and machine-machine communication requirements are different.

It is proposed, therefore, to develop certain blocks of code from representative simulator systems. These blocks of code will be FORTRAN coded and will represent typical simulation tasks. In addition, the functions to be performed by the system will be specified by means of high-level logic descriptions.

An example of the type of functional benchmark design under consideration is shown in block form in figure 1. Program A operates on $input_A$ to produce $data_A$; program B operates on $input_B$ to produce $data_B$. Program C operates on dataA and $data_B$ to produce $output_C$; $program_D$ operates on $data_B$ to produce $output_D$. As a further constraint, the larger of output C or D will be used for the next computation.

Depending on the time frame in which $output_C$ and $output_D$ are required, as well as the available computational speed, the sequence shown in figure 2 might be sufficient. This, obviously, is a single CPU solution. However, in a compressed time frame, one might execute programs A and C in one machine, while executing programs B and D in another. This would yield an executing configuration very similar to figure 1 itself.

A good benchmark should be able to determine whether either proposed configuration can handle the assumed processing load in the context of operational programs, FORTRAN coding, commercial system software and a superimposed environment (interrupts, I/O, etc.). Therefore, it is important to avoid specifying a configurational solution in the specification or the problem.

In order to avoid specifying the sequential or parallel performance of various functions, standard flow charts, with their inherently sequential flavor, will not be used to define benchmark requirements. The type of documentation under consideration would be similar to directed flowgraphs. This will allow the potential parallelism of the problem to be presented.

A directed flowgraph for the process previously described is presented in figure 3. The functional parallelism is evident and the association with a single machine/multiple machine implementation has been removed, thus divorcing the functional requirement from the implementation design.

A brief definition of the symbols used in this description follows. The description is taken from Rubey,[9] which provides more detailed information.

A directed flowgraph represents a computing machine in which the operations are represented by the nodes of the graph, and the transmission and storage of data and control information is represented by connecting links. Control links are represented by lines with open arrow heads; data links have solid arrow heads.

Several classes of nodes are used, two of which are shown in figure 4. Operator nodes represent a function of one or more inputs. Data operators have output connectors, all of which are data. For example, task nodes perform some operation on the data conveyed by the input data links and make the result available on output data links. Task nodes may also have input control links which affect the execution of the node, but do not affect the value produced if the execution proceeds.

A specific operator node which is used in the example of figure 3 is the identity node, labeled "I." This is a data node which transmits its input data value to all data output links with no change. Its utility comes from its use with control links.

The selector node is the means by which alternate sequences of operations may be executed, based on the value or status of some data. Selector nodes have at least one data input link, and always have two control output links. The selector has associated with it a predicate B, which is applied to the data on the data input links. If the resulting value is TRUE, then the "+" control output is ENABLED and the "-" control output is DISABLED. If the result is FALSE, then the converse link status results.

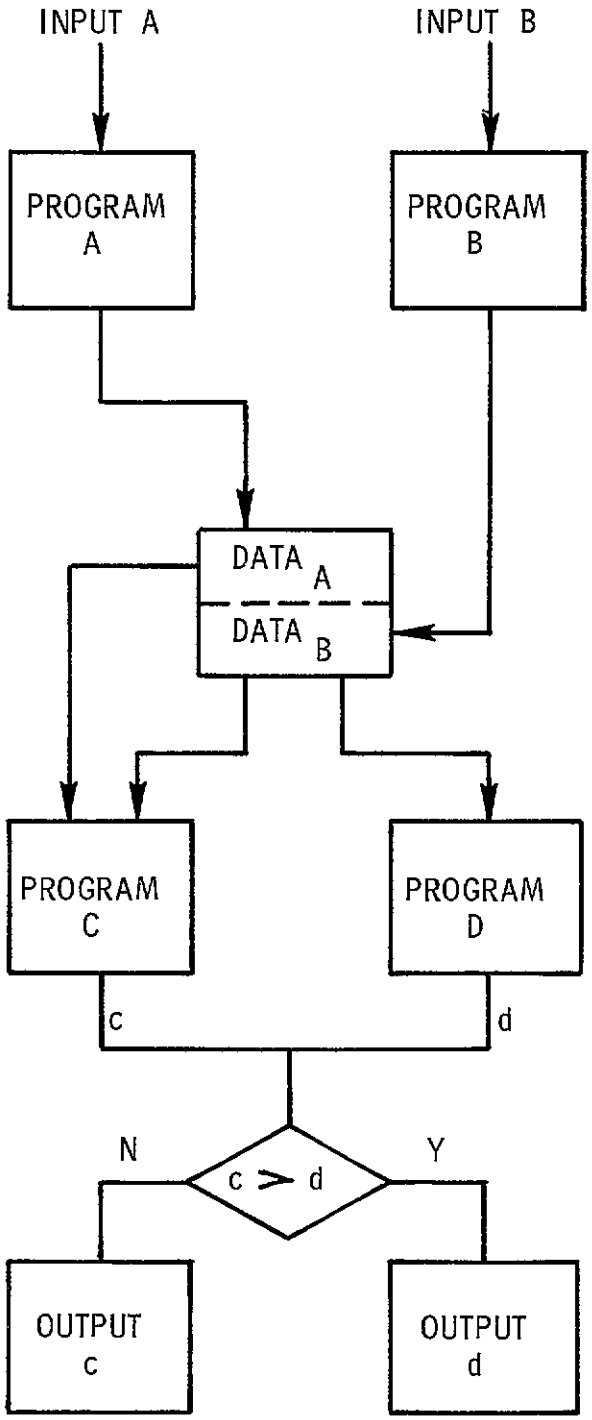


Figure 1. Required Data Flow

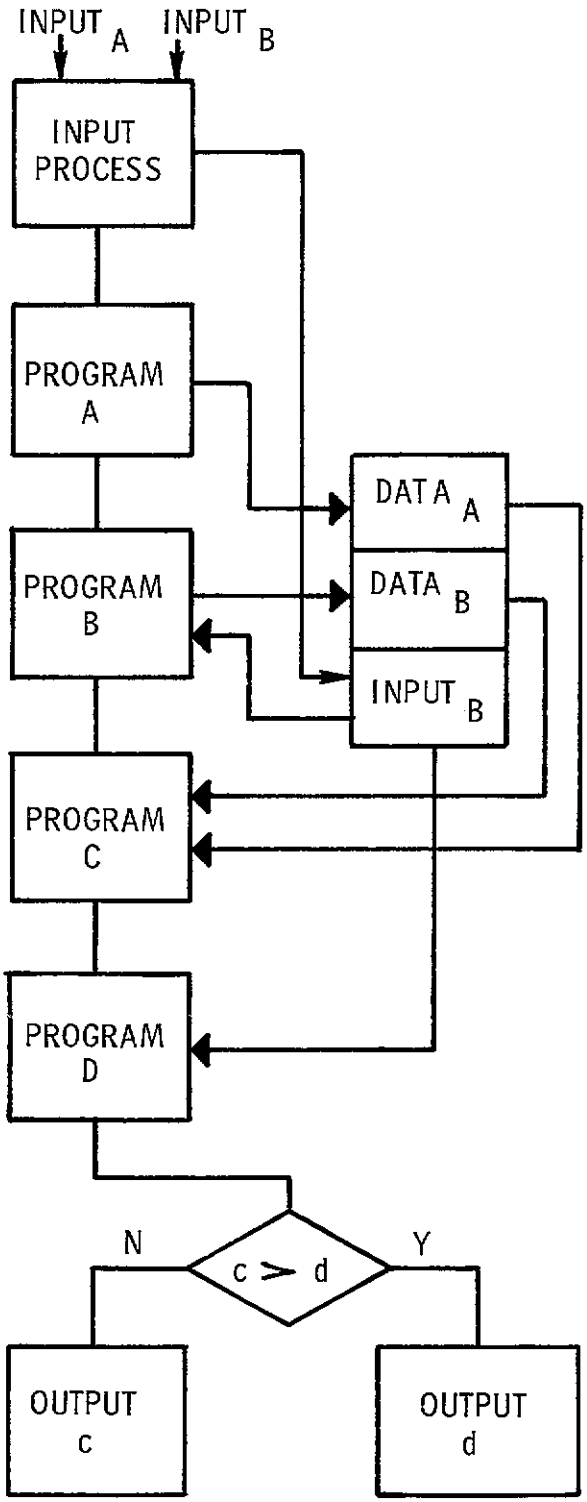


Figure 2. Sequential Data Flow

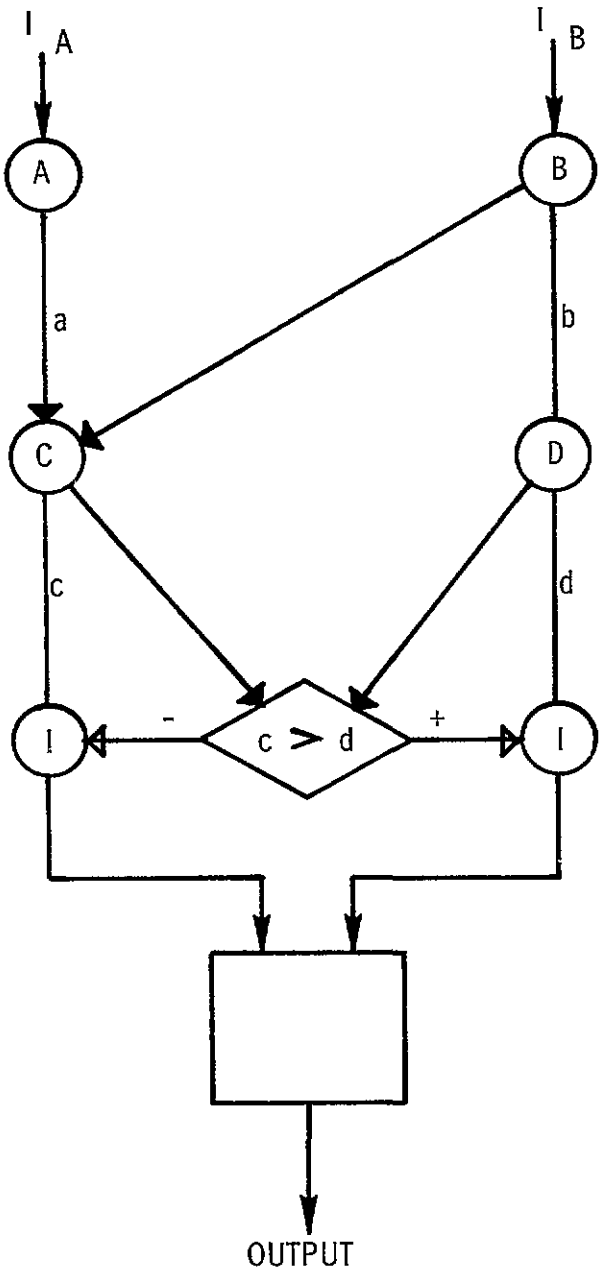
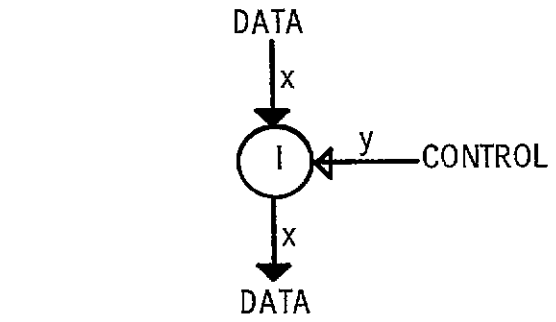
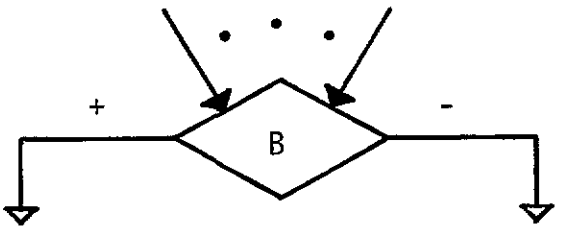


Figure 3. Flowgraph for Task Processing



a) IDENTITY OPERATOR NODE



b) SELECTOR NODE

Figure 4. Sample Flowgraph Nodes

Considering figure 3, we see that the functions delineated by the left and right hand processing chains now appear independent from a processing point of view. They can be implemented in whatever fashion is most appropriate for the particular proposed configuration. The code used for the actual operations A, B, C, and D can be prepared in FORTRAN and compiled and executed regardless of the configuration.

4. SUMMARY

Because of the complex requirements of training simulators, as well as the range of techniques by which these requirements can be met, a semi-automated tool is required to aid in system selection. This tool must evaluate total computational system capability; software as well as hardware.

This requirement is being met by the design of a simulation system benchmark based on natural code blocks configured into a set of synthetic, functional problems. The expression of the problem will be structured so as to avoid any implication regarding the implementation of the solution.

REFERENCES

1. Williams, O., "A Methodology for Calculating and Optimizing Real-time System Performance," Comm. ACM, July 1968.

2. Stimler, S. and Bruns, K. A., "A Methodology for Computer Selection Studies," Computers and Automation, May 1963.

3. Lucas, H. C., "Performance Evaluation and Monitoring," ACM Computing Survey, Sept 1971.

4. Boyse, J. W. and Warn, D. R., "A Straightforward Model for Computer Prediction," ACM Computing Surveys, June 1975.

5. Keller, R. F. and Denham, C. R., "Computer Selection Procedures," Proceedings 23rd National Conference ACM, 1968.

6. Joslin, E. O., "Application Benchmarks: The Key to Meaningful Computer Applications," Proc. FJCC, 1972.

7. Oliver, P., et al, "An Experiment in the Use of Synthetic Programs for System Benchmarking," Proc. FJCC, 1974.

8. Babel, P. S., "Considerations in High Order Language Compiler vs. Assembler for Programming Real-time Training Simulators," ASD/ENCT Technical Memorandum 75-2, 7 February 1975.

9. Rubey, R., "Directed Flowgraphs: An Overview," Softech TP041, 25 February 1976.

ABOUT THE AUTHORS

MR. PHILIP S. BABEL is a Computer Group Leader in the Simulator Division, Directorate of Equipment Engineering, Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio. He has been with the Aeronautical Systems Division since 1966, being responsible for computer systems hardware and software engineering in crew training simulators. He has participated as working group chairman in several Air Force all-command projects to develop approaches and policy for acquisition and support of computer systems embedded in defense systems. Formerly, he worked for the Federal Aviation Agency and RCA as a computer systems engineer engaged in developing automated air traffic control systems. He received a B.S.E.E. degree from the University of Detroit and an M.S. degree in computer and information science from Ohio State University.

DR. M. LEONARD BIRNS is a Technical Advisor in the Directorate of Equipment Engineering, Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio. Previously, he was with Computer Sciences Corporation, specializing in advanced techniques in real-time programming. He was with RCA for two years where, as a principal member of the engineering staff, he participated in several projects, including those involving the design of real-time operating systems for radar control and the development of special purpose display software. At Decision Systems, Inc., where he was Deputy Director of Programming, he was responsible for mathematical modeling and systems analysis for the LEM simulator, and was responsible for development of COMPOSE, a simulation display preparation language and control processor for a Coast Guard helicopter simulator. Dr. Birns received a B.E.E. degree from the City College, New York; an M.S. degree in physics, Magna Cum Laude, from Fairleigh Dickinson University; and a Ph.D. in operations research from New York University.