COMPUTATIONAL PROCESSOR DEMANDS OF ALGOL-60 PROGRAMS*

by

Robert E. Brundage** and Alan P. Batson
Department of Applied Mathematics and Computer Science
University of Virginia, Charlottesville, Virginia 22901

The characteristics of computational processor requirements of a sample of Algol-60 programs have been measured. Distributions are presented for intervals of processor activity as defined by input-output requests and segment allocation requests occurring within the Johnston contour model and within a stack model in which array allocations are treated separately. The results provide new empirical data concerning the behavior of this class of programs. Some implications of the empirical results which may influence computer system design and performance are presented and discussed.

Keywords: program behavior, contour model, Algol-60, processor distributions, resource allocation.

CR Categories: 4.22, 4.35, 4.6, 6.21

1. Introduction

An earlier report (1,2) described measurements of the virtual memory requirements of a collection of Algol-60 programs, and we present here some additional results which complement the previous study. The measurements presented below pertain to the computational and input-output processing requirements of the same collection of programs. For the computational processor, we give distributions for intervals of processor activity determined by certain events which cause a process to become blocked on a resource request; for the input-output processor we present the characteristics of intervals in process time between successive input-output requests. We first describe briefly the conceptual models used in this study.

2. Conceptual Models

(i) The contour model, presented by Johnston (3), has been used for the description of the semantics of several block-structured languages. Basically, a process is described by an algorithm (i.e. the symbolic program itself) and a time-ordered sequence of "snapshots" or records of execution. The contours enclose the blocks of the algorithm, and the execution record immediately following block entry will contain a fresh copy of the contour which has been entered. Thus a procedure call in Algol-60 corresponds to the "activation" of a contour as the site of execution in the contour model. The importance of the contour model for our purposes is that it provides a conceptual framework and a terminology with which to describe and delimit execution phases of a process in terms of its use of the virtual machine resources. Further details of the contour model and Johnston's contour model machine may be found in (3).

(ii) The resource model is a primitive representation of a virtual machine which executes Algol-60 programs, and where the representation is chosen so as to portray the consumption of resources during process execution. This machine is shown in Figure 1. We represent three distinct

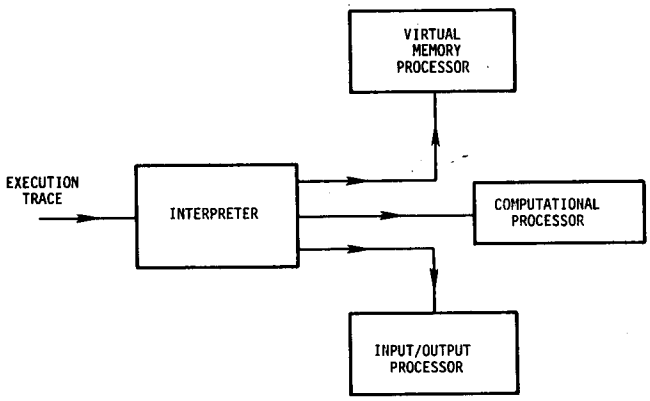

Figure 1 Algol Process Machine

resources - a computational processor, an input-output processor, and a virtual memory processor. An executing Algol program can be represented as a linear sequence of requests for these resources, which is called an execution trace. Figure 2 gives an example of the execution trace of a

*This research was supported by NSF Grant GJ-1005

**Present address: Department of Mathematics, Florida State University, Tallahassee, Florida 32306

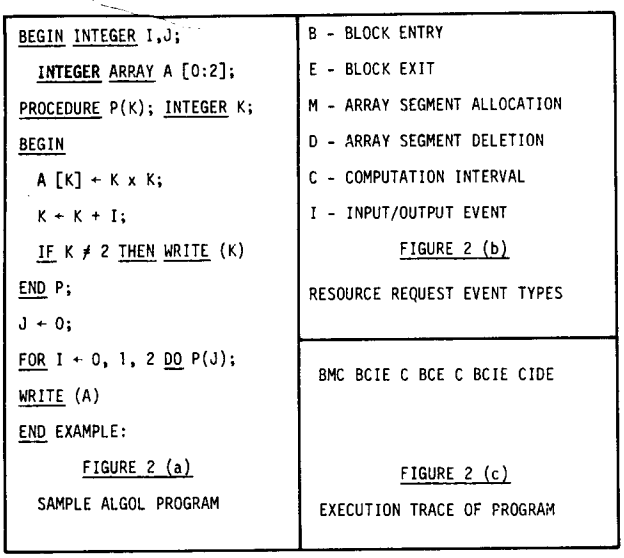

Figure 2 Algol-60 Program Example

simple Algol-60 program in terms of the symbols representing requests for the virtual machine resources. The symbols and their descriptions are given in Figure 2b; the sample execution trace in Figure 2c. Each occurrence of the symbol C represents an interval of computation on the computational processor, while each symbol I represents an input-output request. The language of execution traces generated by Algol-60 programs has a simple formal BNF-syntax description which is given in (4).

The operation of the abstract virtual machine in Figure 1 can be visualized as follows: The execution trace, representing the stream of resource demands made during process execution, is input to the interpreter, which directs the resource requests to the appropriate processor. In the example in Figure 2, the sequence BCIE representing one instance of activation of procedure P corresponds to (i) a request to the virtual memory processor for the memory associated with entry to a contour (symbol B), (ii) a request for services from the computational processor (symbol C), (iii) a request for input-output by the I/O processor (symbol I), and finally, (iv) a request to the virtual memory processor for exiting a contour.

We next specify the measurement units for the resource model. We have chosen to define time in units of work performed by the computational processor; thus in our example above, the lifetime of the contour associated with an activation of procedure P is simply the processor time indicated by the resource request C in the trace sequence BCIE. In the model, the occurrence of symbols such as B, E, and I in the execution trace do not "consume" time but they do signify that the computational processor has been halted because of a request for service by a different processor. Thus the process is blocked from using the computational processor until this request has been satisfied. We now give a brief description of the measurement technique which was used and some information on the sample of programs.

3. Measurement Technique

The hardware and software of a Burroughs B5500 were modified to permit the acquisition of a magnetic tape of execution events with each event precisely time-stamped. A software-controlled hardware counter was designed and constructed by A. Q. Baxter (5). The 1 MHz clock pulses of the B5500 processor were input to the counter, which could be started, stopped, read, and reset under program control. The B5500 Algol compiler and the operating system were modified such that each event of interest during execution of an Algol program caused a time-stamped event record to be written on magnetic tape. This trace tape was used, in conjunction with an inverse symbol table, to generate symbolic trace tapes which were processed to obtain the results presented here. More complete details of the technique for data collection and reduction is described elsewhere (2,4). All times given in the results are for the B5500 equivalent of our computational processor and do not include time normally spent on that real processor for virtual memory allocation or for I/O processing. These activities were filtered out from the raw trace data during processing to permit presentation of the results in terms of the virtual machine of Figure 1.

The sample consisted of 34 programs, written in B5500 Algol, which were production programs for scientific and engineering problems. They included a BASIC compiler, a linear programming package, and some standard statistical routines. As an example of the range of program sizes, the maximum virtual memory requirement ranged from 162 words to 89,976 words. The largest program contained 99 Algol blocks. For the larger programs, the input data was chosen to keep execution time relatively short in view of the trace data volume, but there was a considerable range in total processor times - the total time used on our 1 Mhz version of the computational processor of Figure 1 for the programs in the sample varied between 90 ms. and 126 seconds. With 34 programs selected partly with the goal of achieving diversity in size and execution time, there is some danger that the smaller, shorter programs are relatively under-represented in the cumulative data, since many typical mixes contain a preponderance of short, small programs. However, for many of the statistics to be presented here, it would seem likely that programming style, rather than a program's size or execution time, is likely to be a greater biassing factor in the results. This possibly is being actively explored, and some early results which confirm this hypothesis are described elsewhere (6).

4. Results

The measurement results are displayed in the form of cumulative distributions of sample values obtained from the collection of 34 Algol programs. The sample values from a given program are not independent because of the cyclic characteristics of computer programs, and in most cases we present two distributions for items of interest. The first is a distribution of averages of values

arising from a single source, such as the execution lifetime of a particular block, and the second is a distribution of all values obtained (the "dynamic" distribution) where each source may contribute multiple values. Both distributions have meaningful interpretations which will be discussed below.

The sample values for the first three distributions to be presented are determined by the process time intervals between the occurrence of either

1. requests for allocation or deallocation of virtual memory;
2. requests for input-output processor assignment.

As mentioned earlier, the requests in both 1) and 2) above cause the computational processor to halt, denoting the end of a process active interval. The processor is relinquished, and the process enters a blocked state, waiting until the request is fulfilled. An important point to be noted here is that the "standard" B5500 Algol-60 procedures (sin, sqrt, etc.) were not treated as procedure calls during data collection and thus do not give rise to a contour crossing in our measurements. However, the computational processor time for the standard function is included in the measured processor time.

Contour Transition Intervals. The first distribution consists of the intervals between contour transitions, which occur as each contour (block) is entered or exited. The crossing of a contour causes a request for either allocation of a contour data segment (block entry) or deallocation of a contour data segment (block exit). A contour data segment (1,2) is the virtual memory allocated for each procedure activation and contains the parameter list, local declarations and return address in the contour model. A contour (block) entry will also cause an allocation request for any local arrays or files; this request immediately follows the request for the contour data segment, so that for measurement purposes we consider both requests as a single request for several virtual memory blocks. The distribution of contour transition intervals is given as a distribution of averages in Figure 3a and a distribution of all transitions in Figure 3b. We define the average contour transition interval to be the sum of all intervals occurring within a given block (i.e. when control resides within that block) divided by the number of such intervals, taken over the entire execution of the program.

The medians of the two distributions are not significantly different (0.38 ms for the distribution of averages and 0.36 ms for the dynamic distribution). The lower value for the mean of the dynamic distribution is due to the occurrence of frequently-called procedures. The difference between the two distributions is more strikingly shown by their 90th percentiles: that of the distribution of averages is 28 ms, while that of the dynamic distribution is 2 ms. This pair of

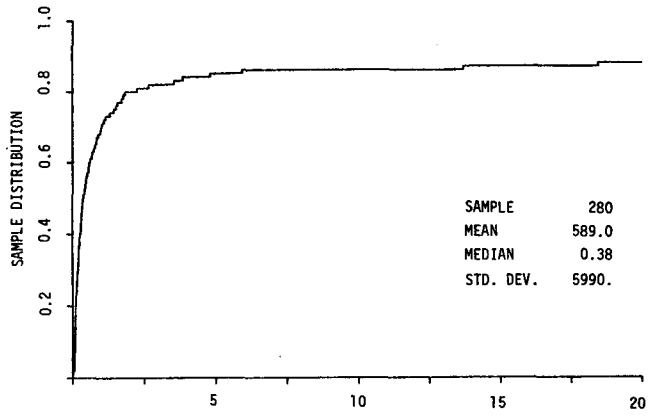

Figure 3a. Contour Transition Intervals - Average (milliseconds)

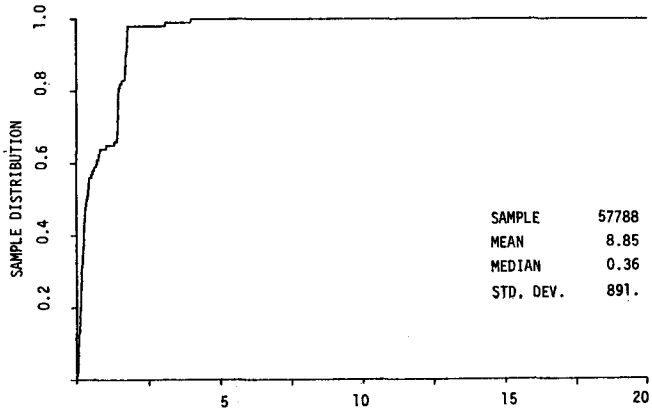

Figure 3b. Contour Transition Intervals - Dynamic (milliseconds)

distributions illustrates the importance of having efficient hardware mechanisms for implementing procedure calls and exits in Algol-like languages. These measurements were made on a 1 Mhz processor, and the times given here should be adjusted proportionally to get equivalent figures for a processor of different speed - e.g. for a 10 Mhz processor, our results indicate that 90% of the computation time intervals between procedure entry/exit events would last for less than 200 μs.

Input-Output Requests. The distribution for input-output request intervals in Figure 4 is only given as a dynamic distribution. The median of 0.16 ms is really quite short, indicating some of the more common structural features of the programs - i.e. the occurrence of input-output statements in tight iterative loops and in close textual proximity in the source language. It should be emphasized that our results simply portray the distribution of the computational work performed between successive read or write statements in Algol-60 programs. These intervals of processor time between input-output requests cannot be directly compared with the corresponding arithmetic processor (CPU) intervals on many "real" machines. Firstly, in our model, no computational processor time is associated with the usual interpretive formatting of the input-output

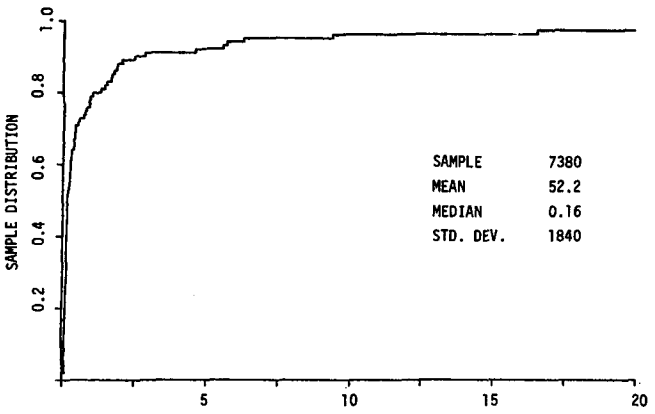

Figure 4 Input-output Request Intervals (milliseconds)

list or with the processes of blocking, buffering, or file control, since these are performed by the input-output processor. Secondly, when the arithmetic processor of a real machine is used for these purposes, then any blocking of the logical input-output requests significantly increases the time between requests for physical input-output. The data given here is in rough agreement with the results given for programs running on the University of Texas CDC 6600 by Sherman et al. (7) when these factors and machine speed are taken into account.

Process Active Intervals. The effect of superimposing the above two process mechanisms is illustrated in the distribution of process active intervals (Figure 5). Not surprisingly,

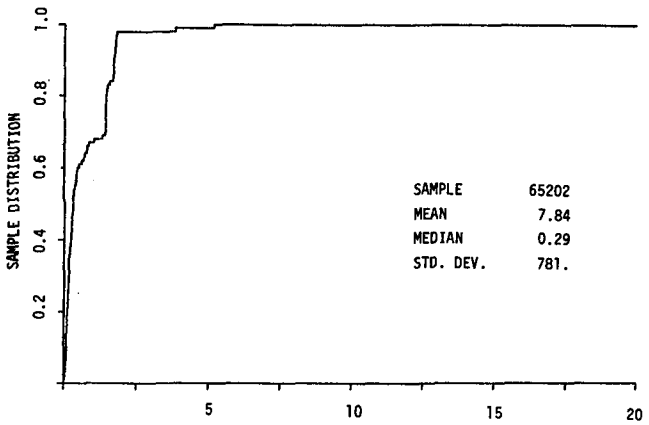

Figure 5 Process Active Intervals (milliseconds)

the contour transition events dominate since they represent over 88% of the sample events. Both the median (0.29 ms) and the mean (7.84 ms) fall between the medians and means of the distributions of contour transition intervals (Figure 3b) and the distribution of input-output requests (Figure 4). The distribution of process active intervals exhibits the characteristic clustering of values evident in several of the dynamic distributions, causing this distribution to be highly skewed towards the lower end of the range. This phenomenon is due largely to the effects of frequently-called procedures as mentioned earlier, where many of the frequently-called procedures also have a short execution time as we show later.

Resource Allocation Model. The three distributions we have just presented may in themselves be used in the context of a simple resource allocation model. Figure 6 illustrates a simple

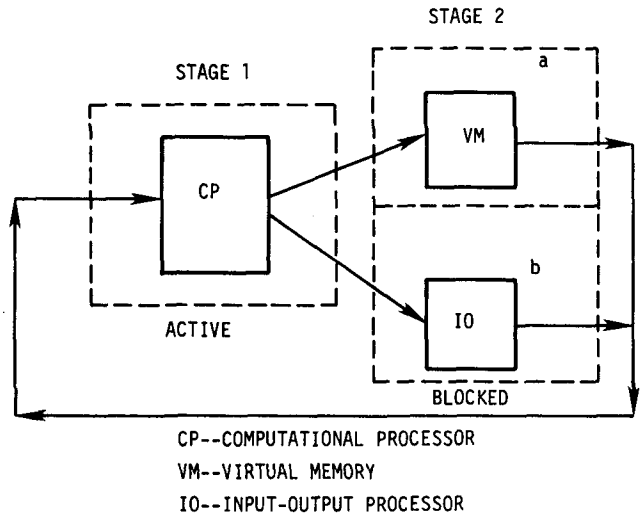

Figure 6 Resource Network Model

cyclic network of three servers corresponding to the three process-machine resources we have been considering. In the active state, represented by stage 1, the process is being executed by the computational processor. Whenever the process issues a virtual memory or input-output processor request, it enters one of two blocked states, represented by stages 2a and 2b of the network. Stage 2a represents the process state "blocked on virtual memory" while stage 2b represents the process state "blocked on input-output". In most of the current network models to be found in the literature stage 2a also includes the occurrence of process stoppage caused by information segments being missing from executable memory. These interruptions of process activity, although related to the semantics of process execution, are dependent on the policies which define how virtual memory components are loaded into main memory by the memory management algorithms within the system. In our model, the interruptions caused by segments missing from main memory are not classified as virtual memory requests, since the segment already has been allocated in virtual memory. Any extension of the state "blocked on memory" to include requests for segment transfers between levels of the memory hierarchy necessitates some assumptions or empirical data regarding the characteristics of information referencing patterns at the symbolic level (8).

In the resource allocation model of Figure 6, the distribution of contour transition intervals (Figure 3b) models the processor service time associated with transitions between stages 1 and 2a, while the distribution of input-output requests (Figure 4) measures the effect of

transitions between stages 1 and 2b. The distribution of process active intervals (Figure 5) is the superposition of these two effects. Analysis of the model in Figure 6 could be carried out in a queueing theoretic framework once the distributions for service at stages 2a and 2b were determined. Network analysis such as (9) could then be used to study the effects of varying service time rates on each of the three resources to isolate potential "bottlenecks" which would cause significant performance degradation in real systems.

Resource Model of a Stack-Based Machine. The results given above have all been related to the behavior of a contour model machine, where all virtual memory allocations are made on demand. In stack-based machines such as the Burroughs B5500/B6700 series, contour data segments are placed in the stack, which is a single data segment allocated only once at the time of process initiation. In such machines, therefore, the only requests for virtual memory occur on entering those blocks having array or file declarations. The distribution of intervals between these data segment requests is given in Figure 7. The first

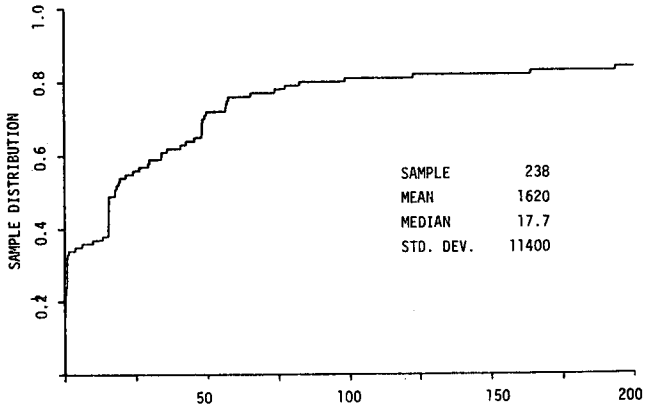

Figure 7 Data Segment Allocation Request Intervals (milliseconds)

characteristic noticed in comparing this distribution with the analogous distribution (Figure 3b) for the contour-based machine is the extreme difference in the sample sizes, with 238 virtual memory requests in the stack-based machine as compared with 57788 in the contour-based machine. A second striking difference between these two distributions is that the median interval between requests in the stack-based machine is 18 ms, while in the contour-based machine it is only 0.36 ms. This illustrates a possible inefficiency which must be dealt with before a contour-based machine can become a practical base for implementation. This observation must be tempered by noting that a number of new block-structured languages, such as Oregano (10) and Algol-68 (11), cannot be implemented on a purely stack-based machine.

To make further comparison of the stack-based and contour-based machines, we can determine the distribution of intervals between all resource requests in the stack-based machine: these are delimited by input-output requests and entry/exit to blocks containing arrays/files. We find that in this distribution the input-output request events almost totally dominate since they represent about 97% of the sample; thus the distribution appears almost identical to that given in Figure 4 and is not presented here. The mean of this distribution is 50.4 ms and its median is 0.16 ms.

Block Execution Time Characteristics. In the next series of distributions we examine some characteristics of the execution time for individual program blocks. By the execution time of a block we mean the total process time accumulated within a given block epoch between block entry and exit, excluding all process intervals during which control resides at a different dynamic level. The lifetime of a block, defined as the elapsed total process time between block entry and exit, is equal to its execution time only if the block contains no nested inner blocks and no calls on procedures.

In Figure 8a we give the distribution of the

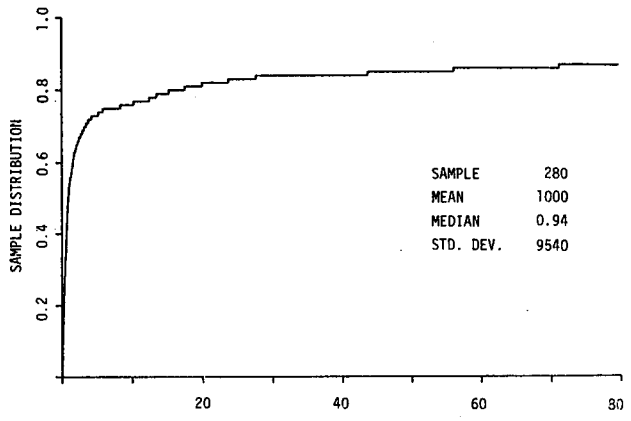

Figure 8a Block Execution Time - Average (milliseconds)

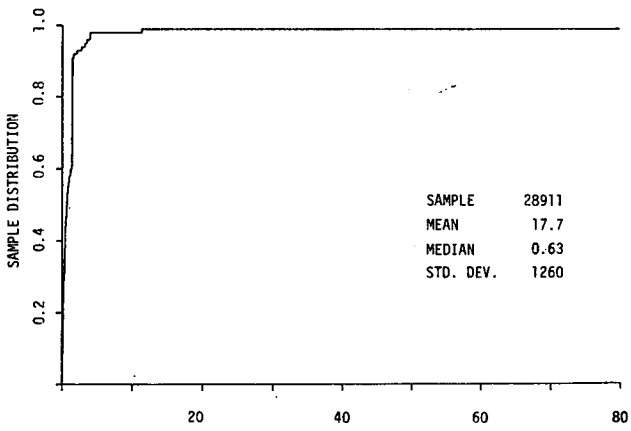

Figure 8b Block Execution Time - Dynamic (milliseconds)

average block execution time for all program blocks. The median of 0.94 ms is surprisingly small. The considerably larger mean of 1 second is essentially due to several blocks in the sample which had an execution time of several seconds. An interesting contrast is found in the distribution of execution times for all block activations, given in Figure 8b. Here the median of 0.65 ms is about 2/3 of the median for the distribution of average block execution time, while the mean of 18 ms is two orders of magnitude smaller. This illustrates that in the dynamics of process execution, most block execution times will be quite short. The 90th percentile of the dynamic distribution is less than 2 ms.

The sample of block execution times can be viewed in another interesting way. Dividing each sample block execution time by the total processor time consumed by the entire program of which it is part yields a "normalized" block execution time. A distribution of such values should not only be essentially independent of the timing characteristics of the computational processor, but more importantly, should provide some insight into the way in which real Algol programs make use of sub-program structure. The results of normalizing the data of Figures 8a and 8b in this way are given in Figures 9a and 9b, respectively. The results are rather surprising: the median of the distribution given in Figure 9a (average) is 0.0006; the median of the distribution of Figure 9b (dynamic) is 0.00007. Putting this another way, we see that if all the events of Figure 9b had been generated by a single program, then the cumulative time taken up by 50% of its block executions would account for much less than 3% of the total process time. This again illustrates that the time required to activate a block or procedure may be a significant factor in total process execution time in a real system.

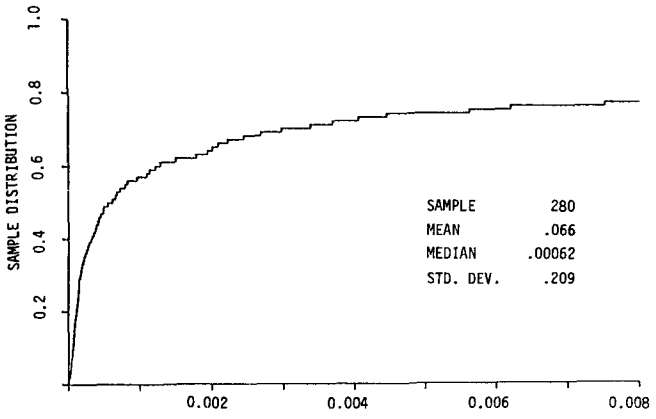

Figure 9a Block Execution Time (Average, normalized)

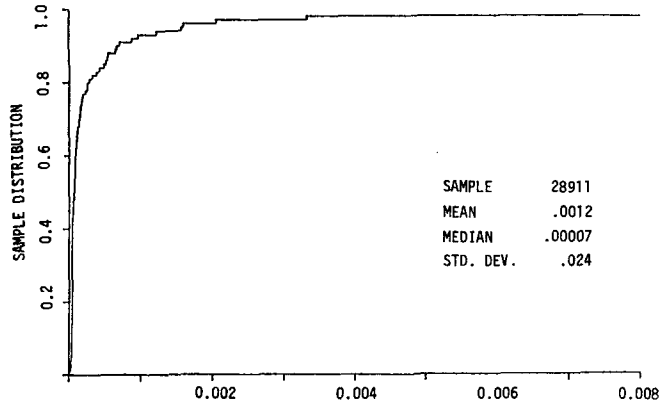

Figure 9b Block Execution Time (Dynamic, normalized)

There are other observations which can be made about the block execution times. The first is that the mean block execution time on an equivalent 10 Mhz computational processor will be less than 2 ms. This is considerably shorter than the time required to transfer a segment or page from conventional rotating mass storage. On a paged virtual storage system, a large, highly modular program may produce a high rate of page faults if the object code modules and data structures are not properly packed into the (linear) page space. The work of Hatfield and Gerald (12) has given some indication of the importance of this effect on the performance of current virtual systems.

A second method of analyzing block execution times is based on normalizing each sample value by its associated block lifetime. This distribution is given in Figure 10a for the average ratio for each block: the distribution of all normalized block execution times is given in Figure 10b.

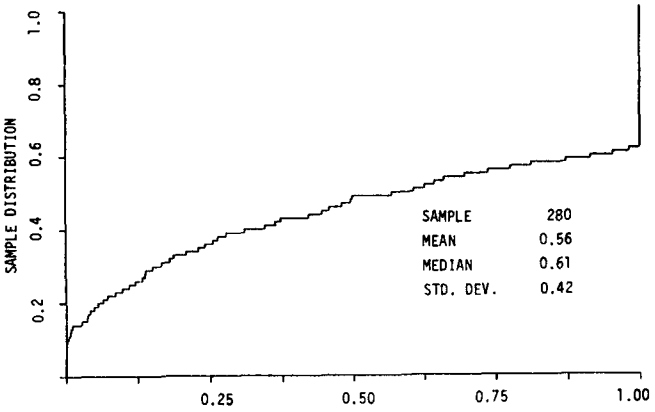

Figure 10a (Execution time)/(Lifetime) Average

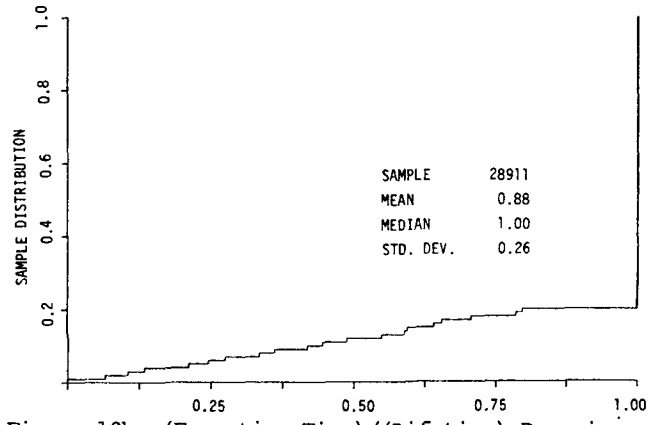

Figure 10b (Execution Time)/(Lifetime) Dynamic

Here we see a striking phenomenon--in the dynamic distribution, about 80% of the block activations (consisting almost entirely of procedure blocks) are due to blocks which did not cause entry to a nested block and did not make any procedure calls. This figure would be considerably greater than 80% if the standard functions of Algol-60 had been included as procedure activations. This fact strongly suggests that mechanisms for implementing procedure calls should incorporate a special, highly efficient mode for calling "simple" procedures.

The sample values of block lifetimes, used above, are interesting in themselves. These lifetimes are in fact the same as the lifetimes of the contour data segments, which have been presented elsewhere (1,2). Briefly, the median lifetime for all instances of block activation is 1.0 ms. The 90th percentile of the dynamic distribution of lifetimes is 3 ms (as compared with 2 ms for block execution time) and is only 40 ms at the 99th percentile level. Again, we can see the crucial importance of the frequently-called, short-lived procedure block in the dynamics of program behavior.

5. Discussion

In this paper we have examined several characteristics of how Algol-60 processes make use of computational and input-output resources. The empirical distributions were presented in the context of a formal model for describing how a process requests and makes use of these resources, and we point out here that these distributions can be related to real systems in several ways. For example, the distribution of process active intervals in Figure 5, which gives the distribution of intervals for the type C symbols in the resource model, represent a sample of processor burst times for a real system where virtual memory allocation and unbuffered input-output are performed by separate processors. Similarly, the distribution of input-output request intervals (Figure 4) corresponds to the distribution of processor burst times on a real, non-virtual memory, system in which all input-output processing (formatting, blocking, buffering, etc.) is performed by a separate processor.

The empirical performance data presented here portrays aspects of program behavior at the symbolic level, rather than in more primitive machine-oriented terms. The data reflects the performance of Algol programs, whereas many contemporary measurements describe the behavioral characteristics of the machine code generated by the complex of compilers, loaders, and other software which serves to implement a high level language machine on contemporary hardware. Our results therefore can be construed as grist for the mill of those dreamers who advocate that machines should be designed so as to be suitable instruments for the tasks specified by programmers, in that this research is an attempt to specify some of the characteristics of these tasks. Knuth (13) has made a somewhat similar argument in a study of Fortran program characteristics, where the emphasis was placed on the incidence of different statement types and their complexity.

We believe that these results, together with those reported earlier on virtual memory allocation (1,2), serve as useful data for comparative design studies of computer systems. The implications of the results are relatively obvious for the design of a machine which directly executes Algol-60, but there are equally important, and perhaps more practical, inferences to be drawn for the design of more conventional hardware systems and their software. To give just one example, the results on block lifetimes and contour transition intervals illustrate the importance of having a low-overhead parameter-passing mechanism for procedure calls. Moreover, the extremely large number of calls made on "simple" procedures (which make no procedure calls themselves), indicates that significant performance improvement of a processor can be achieved by the provision of a special, simple, procedure-calling mechanism for such cases.

In short, the results could be useful in design studies for any system which is to support high level languages. The model and the techniques which have been used here are of a general nature, but the results only represent a relatively small sample of Algol-60 programs. Equivalent data from Fortran, and particularly Cobol, programs might have somewhat different characteristics. Further work of this type on large samples of programs written in other languages could prove to be of significant value to designers of computer systems.

REFERENCES

1. Batson, A. P. and R. E. Brundage, "Measurements of the Virtual Memory Demands of Algol-60 Programs", (Extended Abstract), Proc. 2nd Annual ACM SIGMETRICS Sym. on Measurement and Evaluation, Montreal, 1974, pp. 121-126.

2. Batson, A. P. and R. E. Brundage, "Segment Sizes and Lifetimes in Algol-60 Programs", submitted for publication.

3. Johnston, J. B., "The Contour Model of Block-Structured Processes", ACM SIGPLAN Notices, 6,2 (Feb., 1971), pp. 55-82.

4. Brundage, R. E., Ph.D. Thesis, University of Virginia, 1974.

5. Baxter, A. Q., Ph.D. Thesis, University of Virginia, 1973.

6. Batson, A. P., R. E. Brundage, and J. P. Kearns, "Design Data for Algol-60 Machines"- Submitted for publication.

7. Sherman, S., F. Baskett, and J. C. Browne, "Trace-Driven Modelling and Analysis of CPU Scheduling in a Multiprogramming System", CACM, 15, 12 (Dec., 1972), pp. 1063-1069.

8. Madison, A. W. and A. P. Batson, "Characteristics of Program Localities" - Proceedings of 5th Annual Symposium on Operating Systems Principles, Austin, Texas. November 1975.

9. Hofri, M. and Yadin, M., "A Processor in Series with Demand-Interrupting Devices - A Stochastic Model", JACM, 22, 2 (Apr., 1975), pp. 270-290.

10. Berry, D. M., "Introduction to Oregano", ACM SIGPLAN Notices, 6, 2 (Feb., 1971), pp. 171-190.

11. Van Wijngaarden, A. (ed.), "Report on the Algorithmic Language Algol-68", Numer. Math., 14, 2 (1969), pp. 79-218.

12. Hatfield, D. J. and J. Gerald, "Program Re-Structing for Virtual Memory," IBM Sys. J., 10, 3 (1971), pp. 168-172.

13. Knuth, D. E., "An Empirical Study of FORTRAN Programs", Software-Practice and Experience, 1 (1971) pp. 105-133.