TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FAA-RD-73-195		

4. Title and Subtitle	5. Report Date
COMPUTER SYSTEM PERFORMANCE MEASUREMENT TECHNIQUES FOR ARTS III COMPUTER SYSTEMS	December 1973
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
V. J. Hobbs, J. G. Gertler	DOT-TSC-FAA-73-22
9. Performing Organization Name and Address Department of Transportation Transportation Systems Center Kendall Square Cambridge, MA 02142	10. Work Unit No. FA403/R4109
	11. Contract or Grant No.
	13. Type of Report and Period Covered Final Report April 1972 - June 1973
12. Sponsoring Agency Name and Address Department of Transportation Federal Aviation Administration Systems Research and Development Washington DC 20590	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Direct measurement of computer systems is of vital importance in: a) developing an intelligent grasp of the variables which affect overall performance; b) tuning the system for optimum benefit; c) determining under what conditions saturation thresholds will be reached; d) understanding the effect of hardware or software alterations; and e) in establishing specifications for future systems.

The potential contribution of direct system measurement in the evolving ARTS III Program is discussed and software performance measurement techniques are comparatively assessed in terms of credibility of results, ease of implementation, volume of data, extent of useful information derived, and computer resource requirements. Hardware Monitors, Simulation and other measurement tools are also described. The applicability of these measurement tools and techniques to the ARTS III system is indicated.

17. Key Words	18. Distribution Statement
Computer System Measurement, Performance Measurement, ARTS III, Statistical Sampling Monitor, Event Monitor, Software Monitoring, Hardware Monitoring, ARTS III Simulation	DOCUMENT IS AVAILABLE TO THE PUBLIC THROUGH THE NATIONAL TECHNICAL INFORMATION SERVICE, SPRINGFIELD, VIRGINIA 22151.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	114	

Form DOT F 1700.7 (8-69)

PREFACE

The work described in this report was performed as part of an overall effort to investigate the application of Computer System Performance Measurement technology to automated Air Traffic Control.

Software measurement techniques were comparatively assessed through prototype implementation on the H-832 computer at the Transportation Systems Center (TSC). Programs required for measurement data reduction and graphical representation were programmed and exercised by Ms. Ellen Kelley and Ms. Sandra Flanzbaum utilizing TSC's H-832 computer and DDP-516 Computer Graphics System.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT.)

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

The measurement of computer system performance is of vital importance in the determination of workload characteristics; the selection of new computer systems; the design and/or modification of Operating Systems (Executive Systems) and applications; the evaluation of existing hardware and software; the assessment of equipment reconfiguration; and the debugging and tuning of system components, both hardware and software, for overall efficiency. Measurement can identify system imbalances, monitor utilization, locate performance bottlenecks and provide environmental profiles. Computer system measurement and evaluation are usually undertaken for a specific purpose. The variables to be measured must be selected in light of the objectives and should consider the design goals of the particular system.[1] The tool(s) and technique(s) employed must be chosen with regard to suitability to the measurement, resource requirements, ease of implementation, and cost.

The ARTS III system is an actively evolving system. Functional capability is being added in a building block fashion. Hardware and software configurations will vary from site to site depending on workload and control requirements. Changing environment and technology impose new considerations and ultimately alterations. Judicious application of measurement methods and intelligent evaluation of measurement data will permit objective analysis of the system in its various stages of development and operation and also provide real-world data for long-range planning and input to various ATC-related studies.

This report highlights specific areas where measurement methodology can be usefully applied to ARTS III systems, identifies variables of interest in those areas, and suggests state-of-the-art tools and techniques appropriate to the measurement of those variables. Various considerations affecting the choice of implementation of measurement tools are also discussed.

It is concluded that measurement and evaluation can play a key role in the ARTS III Program: 1) as a management tool to

assist in planning and to assess the impact of system changes on saturation thresholds and on overall performance; 2) as an aid in program design, development, and debugging; and 3) as a tool in "tuning" the system to optimum performance.

It is recommended that FAA establish a measurement and evaluation activity as an integral part of the ongoing ARTS program.

It is at this time in the evolution of the overall ARTS III Program that the opportunity is ripe to extract substantive data of extreme value to future planning and the eventual proper evaluation of future developmental efforts. Computer equipment has been installed at numerous operational sites. Direct measurement of these sites can improve the accuracy and thoroughness of the data currently being used in various simulations and studies. Performance analysis of such systems can lead to a better understanding of the factors which influence the efficiency of ATC automation.

The credibility of ARTS III simulation efforts can be enriched by pre-simulation measurement which could aid in the generation of required statistical models and in the accurate representation of critical system elements. Measurement is also the logical follow-on to simulation. It is necessary to validate the simulation and to verify simulation predicted system performance.

The response of a computer system to workload and other variation tends to be highly non-linear. Test bed measurement and evaluation of various versions of the ARTS III systems should be conducted as these systems evolve through modification and addition. This should be done with a view towards minimizing potential bottlenecks and performance anamolies, as well as understanding the total system impact of changes. A lack of quantitative data on the performance of a system increases the possibility that future systems will be designed with all flaws preserved.

2.0 PHILOSOPHY OF MEASUREMENT

The purpose of measurement is insight, not numbers.[2] The approach to measurement must be systematic. Experiments must be well planned with the measured variables carefully chosen to provide desired insight into specific aspects of system performance. Otherwise, the result will be an enormous amount of confusing data, analysis of which may be meaningless or, even worse, misleading.

Measurements should be taken at two levels, macroscopic and microscopic. At the macroscopic level, an overview of the general area of interest can be examined in rough form to determine the significance of various variables. Decisions can then be made as to which measurements should be refined to give a valuable microscopic view of those areas of further concern.

Measurements should be taken over a large enough period of time and repeated often enough to assure a reasonable level of confidence in the processed results. The length of time a system is monitored on a single occasion is governed somewhat by the frequency of data extraction and recording. At the macroscopic level, data may be automatically summarized rather than continuously recorded, or it may be concerned with variables whose state changes take place slowly or occur infrequently. In these cases the monitoring can take place for many hours or days. Whereas, obtaining a microscopic view of some aspect of performance usually requires the detailed recording of rapid and frequently occuring events. It would be impractical to extract such data for hours on end because the volume of data recorded would be prohibitive. In any event, it is necessary to repeat the experiments several times on different occasions to get a truly representative sample of activity.

The value of measurement data is often influenced by knowledge of events surrounding the data gathering which may not be represented physically in the data itself. Such as the erratic behavior or temporary malfunctioning of a hardware component, or an atypical workload due to weather or other environmental

considerations. Such events can shade the interpretation of, or badly bias, results and should be noted at the time of data extraction for proper consideration in the analysis.

It is important that measurement tools be subjected to thorough testing in a controlled environment where results can be verified. Initial experiments can be conducted on programs whose behavior is known, on Operating Systems whose workload is simulated and regulated and on hardware whose individual components are specifically exercised. Such procedures will not only assure the proper functioning of the measurement apparatus but allow for the identification and quantification of distortions in the data possibly introduced by the measurement process, and the assessment of the impact on the system of the presence of measurement instrumentation.

3.0 TYPES OF MEASUREMENT DATA

The types of data obtained from a computer system can be considered to fall into three categories; software related, hardware related, and workload related. In general, the determination of cause and effect relationships in a system requires data from all three categories.

The highly complex nature of computer systems gives rise to a myriad of labyrinthine interrelationships between significant variables. It is not reasonable to gather data on all variables of imaginable interest at the same time. Therefore, a basic problem in measurement is the identification of the proper subset of variables to be examined in order to meet a specific objective. Balances must be struck between overhead and flexibility. Overhead can be incurred in processor time and memory space in the monitoring of extra variables of marginal immediate value; however, the presence of any related variable affords flexibility in the manipulation of new data in alternate fashion at a later time without having to re-run the experiment.

Software related measurements yield information such as

program or system module

1. elapsed time,
2. frequency of execution,
3. time density distribution,
4. flow history (traces),
5. queue activity on shared resources,
6. data base utilization.

Hardware related measurements yield information such as

1. CPU busy or idle time,
2. channel use statistics,
3. peripheral device utilization,

4. memory module utilization,

5. memory conflicts,

6. CPU/channel overlap.

Some measurement data can be considered either hardware or software related depending on the nature of the study. These measurements include

1. frequency of interrupts or traps,

2. OP code use statistics,

3. Paging or segmentation statistics.

Workload environmental measurements provide information such as

1. response time statistics,

2. input and output data rates and frequency distributions,

3. application program key variable activity (i.e., average # target reports per radar sector, number of coasted tracks per unit time, etc.).

These measurement data can be combined in a variety of ways to aid in various analyses. For example, the same raw data used to compute the elapsed time of all subroutines in a subprogram can, if properly recorded, be used to produce a trace history of the flow of program control for purposes of debugging or logic verification. It can also be used to analyze the subprogram hierarchy, produce frequency of "call" statistics, and time density distributions.* Measurement results can be reported as percentages, totals, summaries, or combined to show degree of overlap or balance of activity between various components. Graphs and charts are extremely useful in assessing the relationship of variables at a glance and are considered an important evaluation tool.

*See sample output from Event Monitor Trace program.

4.0 ARTS III AND MEASUREMENT

The nature of measurement and evaluation is such that the desire or need for obtaining different combinations of measured variables or measurement of different variables is invariably born out of analysis of previously obtained measurement data. The uses to which measurement tools are put is limited for the most part, only by the talent and imagination of the analyst(s). It is, therefore, not reasonable or practical to attempt to itemize all possible measurement data obtainable from the ARTS III system, to speculate as to all its applications, or to define all the conceivable combinations and permutations of that data which could be edifying. It is our intention to outline some of the major areas where measurement can and should be applied and from which further exploratory efforts may be launched.

4.1 EXISTING OPERATIONAL SYSTEMS

The ARTS III systems currently installed are single or dual Beacon Level Tracking systems. The Data Processing Subsystem (DPS) contains a single IOP; or in dual beacon systems two IOPs, where each IOP receives data from its own Data Acquisition Subsystem (DAS) for transmission to its own Data Entry and Display Subsystems (DEDS). Both IOP's have access to all memory modules and the functional workload is divided in a pre-determined fashion with each process dedicated to specific tasks. The functions performed by these systems are Target Detection/Declaration, Beacon Level Tracking, Display Processing, Flight Plan Processing, Interfacility Communication and various related systems "housekeeping" chores. The tasks are scheduled and dispatched through an Executive program on the basis of time and priority.[3,4] As a real-time system, one of the major goals of ARTS III system is the maintenance of critical response times while performing its functions.

4.1.1 Measurement of Existing Operational Systems

The multi-faceted character of the ARTS III system involves a fairly large quantity of operational and environmental (workload) parameters which are subject to variance. The exhaustive measurement of existing ARTS III systems can influence the long range direction of ATC automation.

It is at this time in the evolution of the overall ARTS III Program that the opportunity is ripe to extract substantive data of extreme value to future planning and the eventual proper evaluation of future developmental efforts.

Data obtained through direct measurement of existing systems can aid in future planning by defining the operational, performance and automation characteristics of ATC applications. Such information can be used in assessing the ability of existing systems to handle new functional tasks and increasing air traffic, and in studying the effect of proposed hardware/software modifications through the use of simulation, emulation, and/or analytic techniques.

The operational efficiency of the present system and the validity of algorithm implementation can also be determined with the aid of measurement. The system(s) can be tuned and tested for optimum performance of current design and may point the way to major design changes to be incorporated in future systems.

In many instances the same measurement data can be massaged in a multiplicity of ways and analyzed from differing viewpoints to satisfy more than one purpose.

4.1.1.1 Operational Characteristics - The operational characteristics of an ARTS III system can be defined in terms of its hardware/software configuration, site dependent parameters, and ambient ATC load. The first two of these items are available without measurement and to some extent so is the third. However, the accuracy and thoroughness of the data describing the operating environment of a given installation can be improved through direct measurement.

Variables of interest in the definition of ambient ATC load might include all or a subset of the following.

No. of Flight Plans

No. of Discrete Beacon automatic acquisitions

No. of non-discrete beacons

No. of Beacon replies

No. of hits, misses

No. of Beacon Reports

No. of Target Declares

No. of turning tracks

No. of initial tracks

No. of coasted tracks

No. of suspended tracks

No. of cross - reference matches

No. of normal tracks

No. of total tracks

No. of tabular lines

No. of single symbols, Full Data Blocks, Limited Data Blocks

No. of handoffs

No. of controller keyboard requests (by type)

No. of interfacility messages

Useful information about the activity of these variables might include

1. Distributions by time of day/week, (incidence rate).
2. Distributions over the 32 radar sectors.
3. Averages and maximums by sector or time of day/week.
4. Distribution over active displays.

Environmental data of this nature combined with site dependent information provides a detailed operational profile of an installation or some specific aspect of that installation.

Site dependent information includes

number of IOPs or CPMs

number of DASs

number and/or type of displays

number of keyboards and controllers

ATC functional framework

Beacon PRF

number of hits/and or sweeps to declare targets

number of correlations required for acquisition

4.1.1.2 Automation Characteristics - Measureable automation characteristics would include statistics such as

1. counts of each OP-CODE (instruction type) used in the ATC application.
2. Frequency distributions by class of OP-CODE.
3. Distribution of OP-CODE usage among program elements.
4. Number of memory accesses per OP-CODE.
5. Average instruction execution time.
6. Actual memory cycle time and memory queuing.
7. Channel Data rates & channel queuing.
8. Number of memory accesses for I/O transfer.
9. Frequency of interrupts.
10. Distribution by class of interrupt.

4.1.1.3 Performance Characteristics - The performance characteristics of the system can be defined in terms of the utilization of critical resources as a function of the workload, and the ability

of the system to meet specified response time criteria without consistent data loss. In general, the response of a computer system to workload variation tends to be highly non-linear. The actual utilization patterns of ARTS III system resources can be examined as a function of various aspects of the workload if the measurement data pertinent to workload and resources is concurrently recorded.*

It is possible to obtain hardware resource measurement relative to IOP utilization, channel activity and memory utilization, and software resource measurement associated with subprogram activity, queuing, and data base utilization.

Variables of interest here include

1. Elapsed time of subprogram modules or program segments.
2. Percentage of IOP utilization for each subprogram or routine.
3. Executive Overhead.
4. Number of accesses to specific data bases.
5. Maximum size reached on data bases.
6. Percentage of busy time for each channel.
7. Queue time of various tasks (when several are eligible for execution simultaneously).
8. Total busy time of IOP.
9. Distribution of memory references across memory modules.
10. Operational data of specific interest such as No. of tracks, No. of beacon replies, No. of active displays, etc.

A particular hardware measurement relative to the ARTS IOP channels would be the delay (if any) introduced by the fact that each set of four channels shares an output register.

*This aspect of ARTS III measurement is the subject of a Prototype measurement package developed by TSC at the Minneapolis Test Bed.

Some of the areas in the ARTS III system where exacting demands are made on response time are as follows

1. The processing of Beacon replies (input to the DPS from DAS approximately every 2.5 milliseconds - a function of the PRF).
2. The overall processing of a target from declaration to display (must not exceed 1.2 seconds for tracked targets).
3. The display refresh rate (must be maintained at a 24 Hz minimum, 30 Hz maximum).
4. The scheduling of tasks whose "time to execute" has arrived.
5. Keyboard input and Interfacility message processing.

Measurements to study these response times include

1. Elapsed time of Beacon Input Processing modules (totals, averages, minimums, maximums).
2. No. of Beacon replies received each DAS interrupt (totals, averages, minimums, and maximums).
3. No. of Beacon replies processed each sweep.
4. Time of specific target declares (tag a target).
5. Time of specific target display (check for tagged target).
6. Time at beginning of each refresh cycle for each display.
7. Queue time for scheduled tasks (time between elegibility for execution and actual scheduling).
8. Processing time for keyboard requests.
9. Processing time for handoff and other interfacility operations.
10. Pertinent workload data for the monitoring period such as total tracks in system, total active displays, volume and type of display data, i.e., FDB, LDB, SS, Tabular.

4.1.2 Measurement for Systems Management

Clearly, the availability of concrete measurement data on the operational, automation, and performance characteristics of the existing ARTS III systems would provide a deeper understanding of its actual functioning and thereby would enhance the value of any study attempting to predict the future life, requirements, or direction of ATC systems. If prognostications are to be made as to how a system will react to changes, then it is essential to know in specific detail how the system operates in status quo. For example, the determination of whether a given ARTS system can handle an increase in workload (represented by more air traffic, additional displays, or new functions) without adding new equipment or degrading response times requires a thorough comprehension of present resource utilization patterns, and saturation thresholds. The sizing of new computer architectures for ATC applications requires an understanding of the automation characteristics of the present systems. The development of representative Benchmarks for evaluating candidate machines requires knowledge of the operational profiles of the systems to be replaced.

4.1.3 Measurement and Simulation

Simulation is a popular tool for studying the effects of proposed system modification. A number of simulations have been developed by UNIVAC personnel relative to ARTS III. These simulations are being used for site configuration, memory mapping, and scheduling analyses.[6,7,8,9] The problem with simulation is its credibility. Even very detailed simulations are models--abstractions that explicitly recognize a few system characteristics, approximate some, and ignore the rest.[10] UNIVAC has taken pains to use actual measurement data in the development of certain aspects of its models. A trace program was used to gather information about the number of each type of instruction used by program segments, references to selected memory areas, and number of calls to subroutines.[8] An available measurement tool in the form of a timer program (part of the ARTS III operational software) was used to compare simulation-derived data with

real-world operation relative to gross program timing and IOP usage.[9] Nevertheless, any simulation involves judgement as to the value of representing particular features of a system and a certain amount of educated guesswork which takes the form of "assumptions" or "hypotheses". The degree of representation achieved by a simulation is governed by the reasonableness of these judgements and assumptions. The credibility of ARTS III simulation efforts can certainly be enriched by any pre-simulation measurement which could aid in the generation of statistical approximations, support assumptions and validate judgements. Building simulations around measured and interpreted data increases the utility and credibility of the simulation more than enough to justify the measurement effort.[10,11]

Measurement is also the logical follow-on to simulation, Once a particular path has been chosen and a change or design has been implemented, the actual effect of that change should be measured to substantiate expectations or to discover why discrepancies exist. The source of any discrepancy can usually be diagnosed and the simulation can be refined for future application.

4.1.4 Measurement and Design

If a new or better system or program is to be designed, then a good, quantitative understanding of the performance of previous systems is necessary to avoid performance bugs in the new design. We have no reason to suspect that performance bugs are any less frequent or less serious than logical bugs.[2] A lack of quantitative data on the performance of a system increases the possibility that the new systems or programs will be designed with all the key bugs preserved. It is quite possible that thorough measurement and evaluation of ARTS III systems will uncover performance anomalies or hardware difficiencies which cannot be reasonably altered in the current systems, but which certainly could be considered in future designs.

Focusing on specific system functions with measurement methods can point the way to minor design changes on existing systems as

well as in the development of new programs. Consider the following possibilities. Data relative to the frequency of use of the various keyboard function modules correlated with specific displays would reveal how a controller utilizes the system. This in turn might suggest modifications on additional features such as the automatic re-alignment of data blocks to their original orientation after automatic offset has taken place and the overlapping of data blocks no longer exist. Statistics on "coasted" tracks might be useful in studying the effectiveness of the tracking algorithms or the performance of the DAS.

What is the frequency of controller-entered data? Is the present method of detecting keyboard entries efficient? (The Keyboard Interupt Routine is signalled at the end of each refresh cycle for each display to determine whether a change has occured at the keyboard. This occurs up to 30 times/sec per display).

4.1.5 Measuring Performance Efficiency

The operational efficiency of the present ARTS III systems can be evaluated through the use of measurement. It is entirely possible that a computer system can process the required workload within the specified response time tolerences and not be performing the job efficiently.

This can be a matter of less-than-optimum design since there is seldom enough time available to develop software which 1) meets time and space constraints, 2) is reliable, and 3) simultaneously achieves the quintesscence of performance. Inefficiency can be a matter of inelegant or even erroneous implementation. For example, an error in the scheduling algorithm which results in the entry of a task in the wrong priority queue will usually not result in disaster, but can certainly result in delays which will affect overall performance. Analysis of system operational efficiency usually requires a system programmer level knowledge of system architecture and a good understanding of the goals of the system on the part of the analyst. Pinpointing an efficiency problem usually involves telescoping from macrolevel to microlevel in a recursive series of measurement experiments. This effort can

be extremely rewarding in terms of recovering processing time or memory space which can then be applied to new tasks or used in absorbing a greater workload.

The approach to be taken for this purpose is to conduct a measurement experiment which provides an overview (macroscopic) of ARTS III activity to determine where the time is being spent. This avoids spending time and money correcting obvious but minor inefficiencies with no great effect on overall performance. Analyze the reasonablemess of the results. Itemize the possible contributors to any observed pecularity or suspiciously large time consuming activity. Look into the design logic and code involved, correct or change any source of inefficiency encountered. If necessary, plan further measurement, based on the research, which will shed more light or may verify conclusions. A survey of the literature on tuning with the aid of measurement will show that it almost invariably results in improved performance.[2,12] Tracing techniques are particularly appropriate to an in-depth analysis for determining how the system performs its job. A chronological time-stamped record of the entries and exits of the various ARTS III subprograms and tasks could be used to observe the interaction of subroutines, determine whether scheduling is proper, reveal the hierarchy of events as they really occur, and identify subprograms and tasks whose frequency of use or actual time consumption make them candidates for optimization efforts.

System measurement and tuning activities could conceivably extend the useful life of installed ARTS III systems by delaying the necessity for additional hardware to handle new functions or large air traffic volumes.

4.1.6 Measurement and Reconfiguration

ARTS III enhancement plans call for an increasing number of controller decision aiding functions to be automated; such as Conflict Prediction/Resolution and Metering and Spacing. Reliability will be increased through the addition of redundant hardware components and failure recovery procedures in the software.[13,14,15,16] Improvements in Radar/Beacon surveillance and Data Display are anticipated and the addition of Data Link

facilities will pave the way to other automated services.

Each modification or addition will have an impact on the system as a whole. Each new automated function or hardware element must be properly integrated in order to preserve the integrity of the Real-time environment. Due to the variety in hardware configuration and site dependent parameters at the ARTS III installations, the effect of system alteration will vary from site to site. As new functions are added which require processing time and memory space, the capacity of the various systems to handle a given volume of air traffic will be diminished, response times may be affected and the balance of resource utilization may change. Additional processors, memory modules, and possibly other equipment will be added to absorb the load and will pose anew the questions of: where is the processing time being spent? How are the resources used? Is the system operating efficiently? What is its projected life? What are its saturation thresholds? Can it be "tuned" to better advantage? Measurement can help to answer these questions and should be used diligently at each major step along the way, both for through understanding of actual performance and for intelligent planning in anticipation of needed configuration changes.

A comprehensive set of baseline measurement data on the various ARTS III configurations would provide the yardstick against which the effect of each new modification can be gauged.

A specific measurement of interest relative to hardware reconfiguration would be the overall effect of the CMA which is installed as part of the multiprocessor systems. The CMA increases memory access time from 750 to 950 nanoseconds.

4.2 PROGRAM DEVELOPMENT AND DEBUGGING

Measurement methods can be applied during program development for debugging purposes, with the result that program development proceeds at a more rapid pace.

With the aid of traces for subroutine or branch activity, loop timing, and frequency of execution statistics on statements,

subroutines, segments, or data files, a programmer can locate logic errors and tune his individual task to optimum performance before it becomes a part of a larger program or system.

4.3 FAILSAFE/SOFT MULTIPROCESSOR SYSTEMS

The advent of failsafe/soft systems in the terminal ATC program brings to the forefront a host of new problem areas in terms of computer efficiency and the ability to predict the saturation levels of ARTS III systems.

Multiprocessing is a means of increasing computer power in a system through a form of parallelism rather than through increases in the raw speed of a single processor. It has the added advantage in the case of real-time systems of automatically providing a form of redundancy which can be used to enhance the reliability of a computer configuration. Adding a second processor to a computer system does not, however, double the throughput capability or capacity of a system because a certain amount of degradation is introduced through interference among the various processors while attempting to access the same program code, data base, or memory module. With each new processor added to the system, the effect of this interference is compounded until, at some point, it is no longer cost-effective to add more processors.

The level of degradation at each processor increment can be minimized (and predicted) if sufficient information about the actual operating characteristics of the system are collected and analyzed. For example, programs that share data bases may be scheduled so that they are not being simultaneously executed. Programs which can be run in parallel on separate processors may be duplicated in memory so that multiple processors are not accessing the same memory module simultaneously for program code. Data bases which can be accessed in parallel may be distributed throughout memory in such a way as to minimize conflicts. The necessary information to make such decisions can be gathered with the proper combination of measurement parameters and techniques.

Another factor affecting the efficiency of multiprocessing systems is the requirement for Operating System or Executive software which must perform more complex scheduling, communication, control, and housekeeping functions than those necessary to manage a single processor system. The need for such Executive logic has a twofold affect. Firstly, its sophistication will, understandably, incur a greater overhead in both processing time and memory space; and secondly, its complexity supplies fertile ground for design and coding inefficiencies (or errors) which can have a negative impact on the overall effectiveness of the system. Operating System (Executive) performance characteristics can be analyzed in detail with software measurement tools; and, in fact, such analysis is the subject of numerous measurement and evaluation activities in the data processing field. Once the system software resources have been efficiently organized and optimized, system performance can be further studied to obtain a more accurate picture of actual processing power increase and thus saturation level predictions can be founded on firm data.

The ARTS III multiprocessing failsafe/soft systems are not exceptions to these considerations. Large data bases are commonly required by several tasks which may be run in parallel, the Executive code can be executed by several processors simultaneously while searching for tasks to perform, and the Executive is, needfully, far more complex than that of the basic ARTS III systems. Inefficiencies and delays may conceivably exist in searching the lattice and pop-up tables, in queueing of ESR and I/O requests, in data base organization, and memory access schemes.

To add further ambiguity to the problem of performance efficiency and saturation level prediction for ARTS III, new ATC functions will be periodically added to these systems; such as conflict prediction and resolution, metering and spacing, data link operations, radar reinforced tracking, multi-sensor radar processing, DABS, etc. With each addition or change to the system, performance and workload saturation levels will change.

Many of these problems can be addressed through extensive software simulation, but the reasonable and accuracy of simulation should be verified by actual measurement, whenever possible. If this is not done, a great deal of effort may be expended altering things which are not substantial and the results which are achieved may be significantly less than optimum. Without measurement, simulations are a collection of those characteristics of the system which the analyst considers important and key elements may be ignored. Without measurement, many inputs to the simulation are based on heuristics, statistical approximations, educated guesswork, or a great deal of research and data reduction; all of which leave more room for error and doubt than does direct measurement. Without measurement the effect of actual modification cannot be properly assessed.

The failsafe strategy of the ARTS III system includes periodic recording of critical data on on-line mass storage devices. Initially, disk devices will be used. This facility provides still another area where overall efficiency (and subsequently system life) can be affected. The organization and format of data sets on direct access devices determines how much "head" movement will take place and how much rotational delay will be incurred. These factors in turn determine how long disk requests will be queued and how long tasks must wait for use of the device. The utilization characteristics of disk devices and their effect on overall system efficiency is another measurement area where a great deal of attention has been focused in the past few years.

The critical data recording philosophy of the ARTS III multiprocessing systems is one where the tasks shall decide (within reason) which data to record and how frequently it should be recorded.[16] It is easy to imagine how, with ever increasing functional tasks, queueing for disk use could become a problem in future ARTS III,systems. The recording of numerous copies of the same basic data may also occur. If periodic measurement of these systems is performed as they evolve, may of these problems can be avoided or their effect minimized.

Some particular measurements of interest in the ARTS Multiprocessor systems are as follows

1. Time spent processing (searching, updating) Executive Tables (LDT, PTP, and Pop-Up)
2. Scheduling queue time for individual tasks (eligible but not executing)
3. PTP queue length distributions
4. Cycle completion times
5. Interrupt handling (time, frequency)
6. Interrupt Queue time
7. Processor queue times for shared table use
8. Percentage busy time for each processor
9. Program stretchout due to memory interference for each task
10. Time spent executing Pop-Up tasks
11. Distribution of pop-up tasks across cycles
12. Distribution of tasks among processors
13. Time spent recording critical recovery data
14. Memory references for recovery data
15. Access characteristics of disk
16. Frequency of recording recovery data (total, by task)
17. Task queue time for disk use
18. Rate of update to each recovery data base
19. Effect of CMA on overall performance (slows memory access from 750 to 950 nanosec.)
20. Memory queue effect on display refresh rates.

5.0 MEASUREMENT TOOLS AND TECHNIQUES

We have discussed the utility of direct computer measurement and suggested various areas in the ARTS III program where it can be benefically applied. Let us now look at the tools and techniques available for extracting measurement data.

In general, monitors can be classified as Software Monitors, Hardware Monitors or Hybrid Monitors. Within each of these three categories there are various techniques or approaches which may be employed depending on the measurement objectives, and the environmental constraints. Of course, a measurement and evaluation effort may combine tools and techniques from all three categories in any reasonable fashion to achieve its goals.

5.1 SOFTWARE MONITORS

Software monitors consist of code imbedded in the system for the purpose of recording interesting data.[17] Most Software techniques intercept, in some way, the normal flow of programmed procedures to obtain the required information.[18] The amount of code imbedded in the system depends a great deal on the amount and type of information sought.

Software monitors can be somewhat arbitrarily separated into two categories, Event Monitors and Statistical Samplers. While it is possible to obtain, in some cases, similar information with either technique, there are significant differences in ease of implementation, extent of information obtainable, credibility of results, computer resource requirements, and volume of measurement output. In order to properly assess these differences, and to put them in perspective relative to their importance in ARTS III measurement, several prototype measurement packages were developed for an H-832 computer available at TSC utilizing the different monitoring techniques.

The purpose of the prototype development was to compare the monitoring techniques from the vantage point of first-hand implementation experience. For purposes of comparison the Event Monitor

was limited to gethering data which could be readily analogized with output from the Sampling Monitors. We chose, therefore, to monitor and time the entry and exit from each program element of a large simulation program. This same simulation program was analyzed utilizing sampling techniques with both fixed and random sampling intervals.

The selection of a single large application program as the measurement "test-bed" for this study provided several expedients.

1. It reduced the necessary tinkering with the existing Operating System. This saved time, effort, and some frustration.
2. It provided a controlled (and controllable) environment whose
   a. Characteristics were familiar.
   b. Performance was known, repeatable, and not subject to fluctuation. This was necessary to establish confidence in the operation of the measurement package and in the measurement results.
   c. Source code was available for examination. This was useful in debugging the measurement package.
3. It was sufficiently large, complex, and unwieldy to represent some of the practical and logistical problems of a large system and thereby presented a reasonable set of challenges to the implementation of measurement packages.

It is worthy of note that although this experiment was conducted on an application task, the procedures are applicable and extensible to an entire system.

5.2 THE TECHNIQUES

5.2.1 Event Monitoring

Event monitoring, as the name suggests, is the process of noting the occurrence of specific events of particular interest

to the analysis at hand. The events usually are such things as the entering and/or exiting of program modules, the incidence of pertinent interrupts, updating of critical queues, start and end of I/O operations, accession to a particular device or data base, schedule initiation, etc. The monitoring package may compute elapsed time for events, time-stamp them for later analysis, keep a running count, or dynamically adjust percentage statistics. The design of the monitor is very much dependent on the nature of the analysis, availability of computer resources, and environmental constraints.

5.2.1.1 Ease of Implementation - When event monitoring is implemented through software means, it entails modification of existing application program or system software in some fashion. "Hooks" are planted at strategic points in the software; these hooks cause control to pass to the monitoring package which then takes appropriate action such as incrementing counters, reading a time-of-day clock, and gathering additional data from system tables. The monitor record is then composed, formatted, and recorded.

Hooks may take any number of forms from innocuous branch instructions, thru sophisticated executive service requests, to diabolical, intentionally planted, interrupt-causing pseudo errors. For the most part, hooks are accompanied by codes which indicate the nature of the specific event to the monitor. Implantation of hooks, if not part of the original system design, usually requires re-assembly or re-compilation of those elements to be monitored. The dynamic planting of interrupt-causing conditions is an exception. In any case, successful implementation of this technique usually requires a thorough, in depth knowledge of the structure of the object system software and the architecture of the computer. It is, therefore, normally performed by senior systems personnel.

The method used in the H-832 prototype Event Monitoring measurement package to acquire "hooks" was that of planting coded, interrupt-causing, pseudo-error conditions. An advantage of this method is that re-assembly or re-compilation is not necessary, and the measurement process is therefore relatively painless to the object program (mer). In addition, hooks may be easily removed

so that the program will run again normally, with no juggling of actual program code. Essentially, the measurement process is transparent to the monitored program. This consideration is not to be taken lightly when dealing with large, unwieldy systems. This method differs from that used in the Executive Scheduler Tracing package implemented as part of the ARTS III prototype*. Both techniques are transparent to the measured process. However, the technique used in the Executive Schedule tracing packages will measure elapsed time at the "subprogram" level only, while a finer granularity and flexibility is possible with the planted hook procedure --- at the cost of greater implementation complexity.

In the H-832 Event Monitoring prototype the entry point instruction in each subroutine is replaced with an instruction containing an illegal op-code and a coded address field. This replacement is performed by the event monitor immediately after program loading and prior to program execution. (It can be designed to take place at any time after program loading.) The original entry point instructions are retained by the monitor in an ordered list. The interrupt procedure of the host computer is then modified to give control to the event monitor program at the occurrence of an illegal op-code interrupt.

When, in the course of program execution, a subroutine is entered, an illegal op-code interrupt will be generated by the hardware and the monitor will receive control. At this point, the monitor first determines whether the interrupt was caused by a planted hook or an actual error. If the interrupt was not a planted hook, control is passed to the normal interrupt handling routine. Otherwise, a monitor record is composed containing the program code (obtained from the hook), the clock time, and the address of the entry point. Obviously, any other system data of interest can be gathered at this time, as well. The formatted record is then transferred to an output buffer for subsequent output. The event monitor makes provision to gain control again when

*Several prototype measurement tools have been developed for the Basic ARTS systems, by H. Glynn and F. Woolfall of TSC.

the subroutine exits, the original entry point instruction is executed via an XEQ* type instruction, and control is returned to the object program.

The method which was used by the H-832 Event Monitor to regain control at subroutine exit time makes use of several facts.

1. The simulation program (object program) was written entirely in a higher-level language. This usually means that a standard set of register-use conventions is employed by the compiler. In this specific case, return from subroutines is always accomplished via an indirect jump instruction utilizing the contents of register 15.

2. The Operating System does not support the execution of co-routines, recursion, or any other form of operation which would permit a subroutine return sequence to belong to any subroutine other than the last one entered.

Once these facts were established, regaining control at subroutines exit time was assured by altering the contents of register 15 before returning control to the subroutine, and saving the original contents of register 15 in a simple stack. The address at the top of the stack (last one entered) is then used to return control to the calling routine after the monitor has taken appropriate action to note the subroutine's exiting. If parallel executing of subroutines were supported, a means of coding the subroutine return sequence to identify the returning subroutine would be necessary.

The method employed in the Event Monitor prototype for planting hooks at subroutine entry points can be used to plant hooks at other places in the object program (or system) to count or time other events. The basic requirement is that the instructions to be replaced by hooks have "global" (as opposed to local) labels, or are near such globally labelled instructions, so that their references can be resolved for the event monitor code. It is also desireable to avoid replacing BRANCH, JUMP, SKIP, XEQ or similar

*An XEQ type instruction allows an instruction (the object of the XEQ instruction) to be executed out of sequence.

instructions which would cause special concern as the object of the Monitor's XEQ mechanism. A particular pitfall to be aware of in planting hooks in this manner is the possibility that other instructions in the object code, preceding or following the instructions to be replaced, are somehow dependent on the original contents of the replaced instruction. For example, a register may be set indirectly with the contents of the address field of the replaced instruction to be used in an indexing operation. This, of course, would wreak havoc with program execution. This type of error is extremely difficult to locate when debugging. It is easy to see from even this cursory discussion that Event Monitoring demands an intelligent grasp of the fundamental operational and programming character of the system under study and a great deal of care in implementation.

5.2.1.2 Credibility - The credibility of results from an Event Monitor experiment is affected by the resolution and accuracy of the hardware clock, the distortion of activity introduced by measurement artifact, the length of the monitoring period, and repetition of the experiment.

If considerable accuracy is required in absolute elapsed time measurements as is often the case when probing the "primitive" level of operating systems, then a high resolution timer is very important. The calendar clock used in the instrumentation of "Multics" counts microseconds. That rate is the same order of magnitude as the instruction rate of the host computer, therefore the timing of ten instruction subroutines is meaningful.[19] The Gemini real-time operating system instrumentation had available a ten microsecond timer which could be read or reset to zero under program control.[20] The Statistics Gathering System of the Apollo real-time operating system also used a clock with ten microsecond resolution to record accurate timing statistics.[21] With a timer whose resolution is good enough to time individual instructions or small groups of instructions, the credibility of a single event time becomes excellent. However, when events take place at the microsecond level but the clock ticks-off in milliseconds, there

will most assuredly be many zero elapsed times recorded, although no occurrence of monitored events will be missed. Several different events may also be stamped with identical times even though they did not occur simultaneously. In measurement which requires extreme timing accuracy, this is clearly an undesirable situation. On the other hand, the fact that an individual time measurement may be inaccurate is not necessarily a problem in many cases. In such instances, the elapsed time may be approximated from a statistical sample and a one millisecond or possibly even in 16.67 millisecond (60 cycle) timer is adequate. This is true, for example, in developing a time density distribution where the events to be times will occur many times during the monitor run and average or total elapsed times can be computed within a reasonable tolerence from the overall results. The accuracy of the elapsed time approximation, of course, is dependent on the total time measured for one or more events and the independence of the clock from the events being measured.[11] To illustrate this point, the event-monitor prototype was used to develop a time density profile of the simulation program in execution. The experiment was conducted utilizing a one millisecond time source and then repeated utilizing a 125 microsecond time source. An examination of the raw data obtained with the one millisecond time source revealed several zero elapsed times, while the 125 microsecond time source was sufficiently fine to time the smallest subroutine in the simulation. This was established both manually and by examination of the raw data from the 125 microsecond monitor run. The results are shown in Figures 5-1 and 5-1a (the number of subroutines in the simulation program suggested splitting the results into two graphs). It should be noted that the percentage scale of the two graphs differ. Where the largest concentration of total elapsed time is experienced (Figure 5-1a), it can be seen that use of the one millisecond timer produced overall results almost identical to that of the 125 microsecond timer. Reasonable approximation was also achieved for those subroutines whose running time accounted for less than four percent of the total.

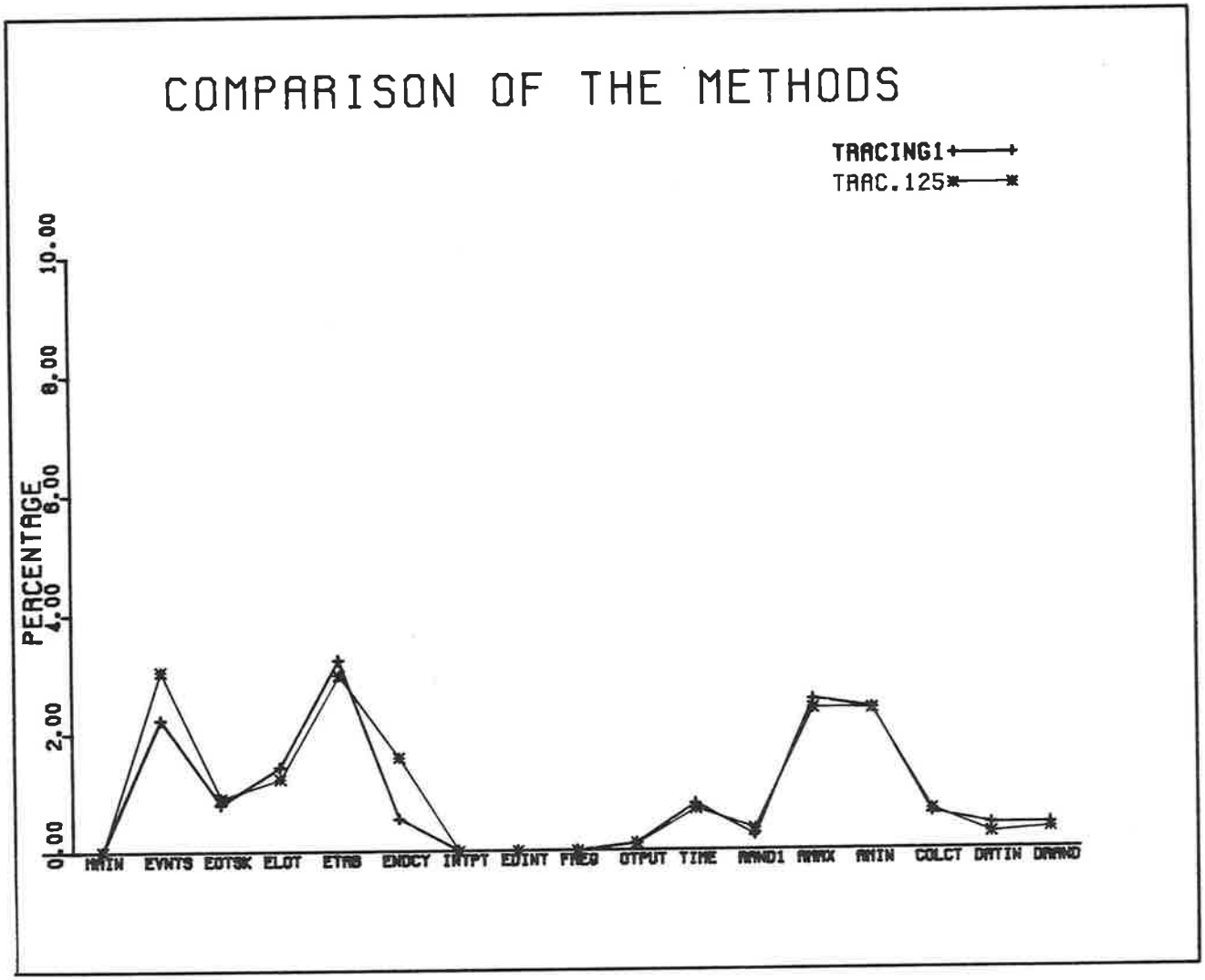


Figure 5-1. Tracing Results (Event Monitor) 1 Millisecond Timer vs. 125 Microsecond Timer

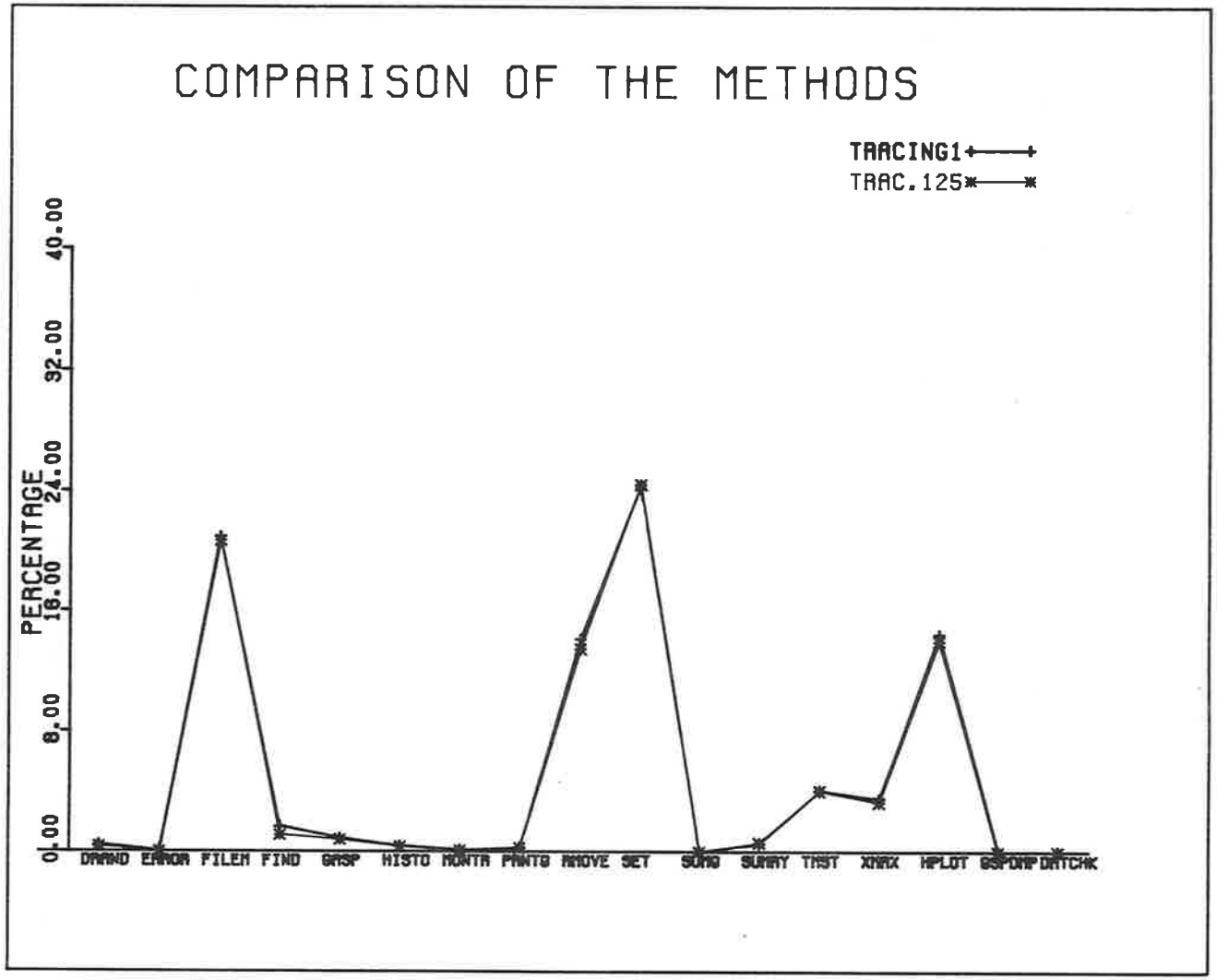


Figure 5-1a. Tracing Results (Event Monitor) 1 Millisecond Timer vs. 125 Microsecond Timer

When individual elapsed time accuracy is required but a high resolution timer is not available, the precision of elapsed time measurements can be improved over that directly available via the hardware clock through software techniques. This is done, however, at the expense of degradation in the total running time of the measured process. Such a technique is described by Bussell and Koster,[22] and is dubbed "the vernier clock algorithm". The procedure has five basic requirements: 1) an accurate hardware clock with 2) minimal jitter, and 3) a very short program loop whose time is 4) accurately known and 5) which can detect (be initiated) when the clock is incremented. Briefly, the technique utilizes the program loop 1) to synchronize event timing initiation with clock incrementation and 2) as a vernier to determine how much of the final interval (between clock incrementations) was spend actually executing the timed process. The algorithm introduces delays of several microseconds, or more, both at the beginning and the end of each timed process in order to determine the higher resolution elapsed time. The length of the delays is a function of the resolution of the available timer. The precision of the measured time is the time required to execute one instruction on the host computer. Elapsed time must be calculated on-the-spot, rather than during post processing data reduction. Effective use of this technique requires that calibration methods be developed to experimentally determine the precise time of the vernier loop, the precise resolution of the hardware clock, and the precise overhead for interrupt processing (used to signal the vernier loop when the clock increments). This can be a rather cumbersome process, but does provide a useful substitute for a high resolution timer under the proper circumstances.

Distortion due to measurement artifact varies considerably according to the methods employed in collecting the data. The mere existence of software measurement instrumentation implies that the normal course of the measured process will be disturbed periodically. In the very least, an instrumentation overhead factor will be introduced. Typical overhead for measurement has been reported to be in the neighborhood of 1% to 10%. This

overhead, however, can run much higher, especially in event monitoring, if not carefully controlled.

Separating the time spent monitoring from the time spent in the monitored process can be difficult. For example, if the clock is continuously running and is used to "time-stamp" the occurrence of an event, the time used by the monitor to collect the data may be included as part of a later calculation of elapsed time between two events. This monitor time can be significant, and not necessarily constant. Steps should be taken, wherever feasible, to minimize monitor time or to properly account for it. This particular problem was reasonably resolved in the H-832 prototype by exploiting the fact that the clock could be stopped and subsequently resumed without resetting (with some programming acrobatics). Therefore, the clock was halted as soon as the monitor program was initiated and resumed just before control was returned to the object program. The remainder of the clock interval is lost, however, each time the clock is stopped, so an error factor still exists. It is worthy of note that this action was taken after certain peculiarities in the original output, obtained with a continuously running clock, were noticed and investigated. It is always tempting to assume that this type of error will be insignificant and thereby not worth the effort to minimize.

In addition to the problems of clock resolution and distortion of results, credibility is affected by the length of the measurement period and repetition of the experiment because a system reacts to workload and environmental changes, and a program may take different paths for different input data. If the monitoring period is too brief, significant events may be missed entirely; and if the experiment is not repeated, the results may be biased by an atypical situation. The danger of drawing erroneous conclusions due to either condition is all too apparent.

5.2.1.3 Volume of Measurement Data - Since the application of event monitoring techniques is largely in the microscopic analysis of internal system events, and since such events usually occur with great frequency, it can be expected that event monitoring will

produce a large volume of output data. Typical of the amount of data accumulated during event monitoring is that reported by Schwetman and Brown.[23] "During observation period of normal production which lasted 4,037 seconds, in excess of 1.8 million events were recorded." The H-832 event monitor prototype package produced 6060 records while merely tracing the entries and exits of the subroutines used during execution of a simulation program. The CPU time logged for the simulation program (exclusive of I/O and monitor time), was approximately 3 seconds. Obviously, the processing of such a large volume of data presents quite a problem in itself. The amount of thought and effort to be expended on this task should not be dismissed as inconsequential.

Analysis of data usually requires summarization and extraction of features of interest. If the analysis is performed as a post processing task rather than dynamically during the monitoring period, many uses can be made of the same data as the need for different points of view arises, and real-time measurement overhead can be minimized.

5.2.1.4 Extent of Information - Event Monitoring has the advantage of permitting a fine grain, microscopic view of activity, and therefore, is well suited to the study of the complex interactions and subtle behavior of systems.

It is possible to obtain queue time and wait time as well as holding time for system resources. The sequences of events as well as their time distributions can be determined. The software traffic patterns of the system can be studied and the hierarchy of events revealed. This information allows the definition of the structure and flow-of-control throughout the system which is extremely useful in debugging as well as system "tuning" efforts. It is possible to ascertain cause and effect relationships or merely to count the incidence of a single event. Areas of system activity are accessible through monitoring which may be "masked" to other techniques. The elapsed time of events can be directly measured rather than estimated. This makes timing fluctuations and other anomalies, which may be significant, readily apparent even at the raw data level.

The natural chronology inherent in event monitor data can be used to advantage in the manipulation of that data in a variety of ways to answer different questions, or to study a problem from an alternate point of view. To illustrate the latter point, consider the raw output of the H-832 event monitor prototype (Figure 5-2). As stated earlier, the events monitored consisted of each entry to, and exit from, every subroutine exercised during the course of execution of a simulation program. Each monitor record contains the subroutine identification, a time-stamp, and an address. On entry to the subroutine, the address field of the monitor record contains the entry point address and on exit from the subroutine, the address field contains the address to which control was returned (in the "calling" routine). A sample of basically raw data, formatted for human consumption, is shown in Figure 5-2. Being from an early test run, the time-stamps in this figure reflect the monitor overhead, including monitor I/O as well as system I/O on behalf of the program, and should not be taken seriously. The clock resolution was 16.67 milliseconds (all numbers are hexidecimal). This is not the data which was used in the final analysis.

The ultimate purpose of the experiment was to produce a time density distribution for the simulation program. The final results are shown as a printer-plot in Figure 5-3, as well as in graphical form in Figures 5-1 and 5-1a. One can see, however, that a wealth of additional information can be gleaned from the output.

A perusal of the unprocessed raw data reveals the hierarchy of events during program execution. Subroutine JUDY (the "main" routine) called subr. GASP, which in turn called subr. DATIN which in turn called DRAND, which called RAND1. Then RAND1 returned control to DRAND which returned control to DATIN which then called SET (which returned control to DATIN) followed by FILEM which itself called SET; and so forth. A closer look at the addresses reveals that the subroutine sequence FILEM, SET, XMAX, XMAX, SET, FILEM is executed as part of a loop in subroutine DATIN rather than from different places throughout the program. (The return addresses are always the same.) This type of information is particularly useful when attempting to understand how a program goes about performing its function. Such information may be sought

```
PGM CODE = 00000005  SUBR = JUDY    ADDR = 00003280  TIME STAMP = 00000000
PGM CODE = 00000015  SUBR = GASP    ADDR = 00005656  TIME STAMP = 00000812
PGM CODE = 00000010  SUBR = DATIN   ADDR = 00004DAC  TIME STAMP = 000008B8
PGM CODE = 00000011  SUBR = DRAND   ADDR = 00005228  TIME STAMP = 00003478
PGM CODE = 0000000C  SUBR = RAND1   ADDR = 00004C20  TIME STAMP = 00003540
PGM CODE = 0000000C  SUBR = RAND1   ADDR = 00005238  TIME STAMP = 000035E8
PGM CODE = 00000011  SUBR = DRAND   ADDR = 00004F7E  TIME STAMP = 0000368E
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 00003860
PGM CODE = 00000019  SUBR = SET     ADDR = 00004FE0  TIME STAMP = 00003928
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00003AFC
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 00003BC4
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 00003C6A
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 00003D10
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 00003DD8
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 00003E80
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00004052
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 00004 0F8
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 000041A0
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 00004268
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 0000430E
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 000043B4
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00004588
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 0000462E
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 000046F6
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 0000479C
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 00004844
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 000048EA
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00004ABC
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 00004B84
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 00004C2C
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 00004CD2
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 00004D9A
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 00004E40
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00005014
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 000050BA
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 00005160
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 00005228
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 000052D0
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 00005376
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00005548
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 000055F0
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 000056B8
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 0000575E
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 00005804
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 000058AC
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00005414  TIME STAMP = 00005A7E
PGM CODE = 00000019  SUBR = SET     ADDR = 0000605C  TIME STAMP = 00005B46
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 00006BA2  TIME STAMP = 00005BEC
PGM CODE = 0000001D  SUBR = XMAX    ADDR = 000061E0  TIME STAMP = 00005C94
PGM CODE = 00000019  SUBR = SET     ADDR = 0000548C  TIME STAMP = 00005D5C
PGM CODE = 00000013  SUBR = FILEM   ADDR = 00004FEE  TIME STAMP = 00005E02
```

Figure 5.2. Raw Event Monitor Output

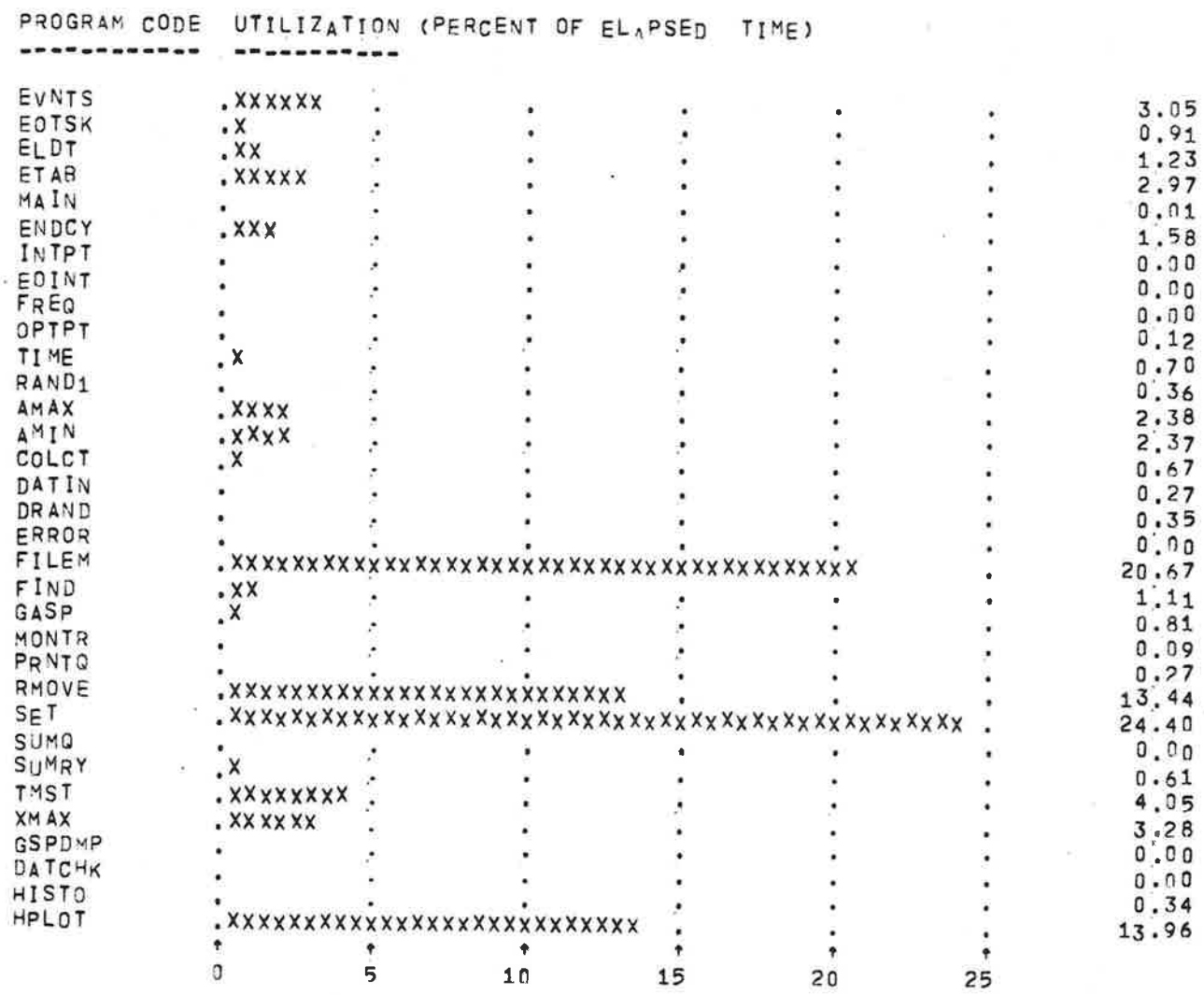


Figure 5-3. Printer Plot of Event Monitor Time Density Distribution

merely for educational purposes or, more often, for purposes of debugging. The raw output can also be useful in determining to some extent the overhead introduced by the monitoring instrumentation, if one is familiar enough with the monitored process to identify points at which an exit from one subroutine is immediately followed by entry to another, etc.

This data can be manipulated to tell other stories. Consider Figure 5-4. These are the processed results of an event monitor run using a 125 microsecond timer. The numbers in the "time" columns represent the number of 125 microsecond pulses of the clock. Here we have counted the number of times each subroutine was executed, accumulated the total time for all executions, computed the mean execution time, and indicated the minimum and maximum elapsed time spent in each separate subroutine. We can see that several of the subroutines are frequently used. They are EVNTS, AMAX, AMIN, FILEM, RMOVE, SET, TMST and XMAX. The bulk of time, however, seems to be distributed slightly differently with ETAB and HPLOT entering the picture. Figure 5-1 and 5-1a depict this same time density distribution graphically. An interesting point revealed by Figure 5-4 is the large difference between the minimum and maximum elapsed times for the subroutines EVNTS, ENDCY, SET and HPLOT. This variation, combined with a relatively low total elapsed time for routines EVNTS and ENDCY accounts for the deviation in Figure 5-1, between the total time accumulated with the one millisecond and 125 microsecond time sources, and also demonstrates why timer resolution matters. The elapsed time variations are not cause for alarm in this particular case, but such conditions might well be a startling relevation in other analyses.

The Subprogram Call Analysis chart (Figures 5-5 and 5-5a) is another useful output from the same data. The chart is divided into two parts for publication but should be viewed as one chart. The numbers heading the columns represent the same numbered subroutines listed in the leftmost column. There are 33 subroutines, thus 33 rows and 33 columns. The "calling" routines are listed vertically and the "called" routines horizontally. For example, we can see that subroutine EVNTS (#1) calls subroutine EOTSK (#2)

PROGRAM CODE	NO. OF EXECUTIONS	TOTAL TIME	MEAN EXECUTION TIME	MIN
EVNTS	156	1228	7.872	4
EOTSK	44	367	8.341	4
ELDT	44	497	11.295	7
ETAB	60	1195	19.917	8
MAIN	1	4	4.000	4
ENDCY	8	635	79.375	9
INTPT	0	0	0.000	0
EOINT	0	0	0.000	0
FREQ	0	0	0.000	0
OPTPT	1	50	50.000	50
TIME	44	283	6.432	3
RAND1	34	146	4.294	4
AMAX	274	960	3.504	3
AMIN	274	954	3.482	3
COLCT	42	269	6.405	5
DATIN	1	108	108.000	108
DRAND	34	143	4.206	3
ERROR	0	0	0.000	0
FILEM	342	8331	24.360	18
FIND	40	446	11.150	9
GASP	1	328	328.000	328
MONTR	1	37	37.000	37
PRNTQ	6	108	18.000	9
RMOVE	342	5418	15.842	15
SET	685	9833	14.355	10
SUMQ	0	0	0.000	0
SUMRY	1	246	246.000	246
TMST	232	1631	7.030	6
XMAX	342	1323	3.868	3
GSPDMP	0	0	0.000	0
DATCHK	0	0	0.000	0
HISTO	17	136	8.000	8
HPLOT	4	5626	1406.500	1016

TOTAL ELAPSED TIME=40302 PULSES
THERE WERE 6060 OBSERVATIONS

Figure 5-4. Processed Results of Event Monitor (125 Microsecond Timer)

***SUBPROGRAM CALL ANALYSIS ***

CALLED SUBPROGRAM

CALLING PROGRAM		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
EVNTS	1	0	44	44	60	0	8	0	0	0	0	0	0	0	0	0	0	0
FOTSK	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ELDT	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ETAR	4	0	0	0	0	0	0	0	0	0	0	0	440	0	0	42	0	0
MAIN	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ENDCY	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
INTPT	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EOINT	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FREQ	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OPTPT	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TIME	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
RAND1	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMAX	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMIN	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COLCT	15	0	0	0	0	0	0	0	0	0	0	0	42	42	0	0	0	0
DATIN	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
DRAND	17	0	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0
ERROR	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FILEM	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FIND	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GASP	21	156	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
MONTR	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PRNTQ	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RMOVE	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SET	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUMQ	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUMRY	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TMST	28	0	0	0	0	0	0	0	0	0	0	0	0	232	232	0	0	0
XMAX	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GSPDMP	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DATCHK	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HISTO	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HPLOT	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-5. Subprogram "Call" Analysis

		18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	TOTL
EVNTS	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	156
EOTSK	2	0	44	0	0	0	0	0	0	0	0	40	0	0	0	0	0	84
ELDT	3	0	84	40	0	0	0	40	0	0	0	39	0	0	0	0	0	203
ETAB	4	0	146	0	0	0	0	146	0	0	0	121	0	0	0	17	0	912
MAIN	5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
ENDCY	6	0	57	0	0	0	0	0	0	0	0	32	0	0	0	0	0	89
INTPT	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EOINT	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FREQ	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OPTPT	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4
TIME	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
RAND1	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMAX	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMIN	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COLCT	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84
DATIN	16	0	11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	13
DRAND	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
ERROR	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FILEM	19	0	0	0	0	0	0	0	342	0	0	0	0	0	0	0	0	342
FIND	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GASP	21	0	0	0	0	1	0	156	0	0	1	0	0	0	0	0	0	316
MONTR	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PRNTQ	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RMOVE	24	0	0	0	0	0	0	0	342	0	0	0	0	0	0	0	0	342
SET	25	0	0	0	0	0	0	0	0	0	0	0	342	0	0	0	0	342
SUMQ	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUMRY	27	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	6
TMST	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	464
XMAX	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GSPDMP	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DATCHK	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HISTO	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HPLOT	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-5a. Subprogram "CALL" Analysis (Cont'd)

44 times because the number 44 is found opposite EVNTS in the column headed 2. Similarly, EVNTS calls subroutine ETAB (#4) 60 times, and ETAB calls RAND1 (#12) 440 times The total number of subroutines that each routine has called is listed as the last number opposite that routines' name in Figure 5-5a. For EVNTS, the total is 156. The chart enables one to see at a glance which routines are used by which. This can be valuable if a change is to be made in the overall program and it is necessary to know what routines may be affected. It may also lead to reorganization. For example, if a subroutine is called frequently by one routine, it may be more efficient to make that subroutine a part of the calling routine instead of a separate subroutine. This chart indicated that AMIN (#14) is called 232 times from subroutine TMST and never called by any other subroutine; and XMAX (#29) is called exclusively by SET, 342 times. Once again, this situation is acceptable in our case because these subroutines are part of a general simulation package, and different input data will very likely change the picture somewhat. If this were a special purpose program, however, it would be wise to look into the possibility of reorganization.

The utility of event-monitor data is manifold.

5.2.1.5 Resource Requirements - Computer system resources are required to conduct measurement via software techniques. These resources are central processor time, memory space, I/O channel time, peripheral devices, and program readable clocks. When event-driven techniques are employed, the central processor is usurped by the event monitor each time an event of interest occurs. The processor is occupied by the monitor for the amount of time needed to record the event and to gather whatever other data is desired. If multiple, frequently occurring events are monitored and related data must be gathered from available tables, a significant amount of processor time will be spend on the monitoring task resulting in delays to all other processes in the system. The nature of event-monitoring, then, suggests prudence in selecting a reasonable set of "variables of interest" in order to minimize the processor time overhead.

The number of events being monitored, and the frequency of occurrence of those events also impacts on the memory space and I/O channel time requirements. Memory space is necessary for the monitor program code, but equally importantly, for the data buffers. It is necessary to allocate sufficient buffer space to accommodate all the new event records which will be accumulated while the previous records are being processed or dumped to an external device. Failure to properly allot memory space will result in lost data or additional delays. Obviously, if events occur with great rapidity and each event is accompanied by incidental information, the buffer space must be large. The amount of channel time required is a function of the size of the buffer blocks transmitted as well as the frequency of output transfer.

It is usually necessary to dedicate a tape drive to the task of recording monitor data, thus tying up a peripheral device. For purposes of timing events, a program addressable clock is a necessity. Event-monitor timing usually requires a high resolution timer, as previously discussed.

5.2.2 Statistical Sampling

Statistical sampling is a method for determining how the computer time and other resources spent on a process are distributed. It is useful for finding the areas where performance bugs may exist and in pinpointing the areas where "tuning" can be applied to the greatest benefit.

Statistical sampling techniques are founded on a premise aptly stated by Cantrell and Ellison[2] as follows: "If an executing program is frequently interrupted according to some random or periodic time schedule which is known to be statistically independent of any natural execution pattern in the program, then the frequency with which the interrupt location falls within a particular instruction sequence is proportional to the total time spent by the program in executing that instruction sequence." In other words, it is possible to develop a time density distribution of a program in execution without precisely timing the entry and exit of every subroutine in that program. It is, in fact, possible to

locate time consuming areas within a program which has no subroutines. Commercial performance monitors utilizing sampling techniques have carried this philosophy one step further and have applied it to overall system activity.[24]

5.2.2.1 Ease of Implementation - The sampling technique can be applied on any computer which has a program interrupt capability and an interval timer, or other clock capable of generating an interrupt after a pre-defined interval of time has elapsed.

In general, the more complex and general purpose the Operating System, the easier it becomes to implement this measurement facility. That is, in terms of the built-in provisions for clock handling, interrupt routing, and the availability of operating system maintained tables containing hardware, system, and job status information. On the other hand, multiprogramming, multitasking, and multiprocessing introduce special considerations which do not exist in uni-programming environments. For example, in a multiprogramming situation, one may have to determine whether a program was executed at all since the last sample was taken.

Basically, a sampling monitor obtains control at the beginning of a monitor run and sets the interval timer (clock) with the selected sampling interval. The timer interrupt facility of the processor must be initialized at that time to communicate with the sampling monitor, and then the object program is initiated. Object program activity is suspended by the computer system's interrupt mechanism when the interval has elapsed and control is passed to the sampling monitor. The sampling monitor records the interrupt address (the instruction counter or program counter value at the time of the interrupt). The clock is then set with a new time interval and control is returned to the object program. The recorded addresses are later sorted or otherwise processed and the the resulting frequency distribution indicates where most of the object program time has been spent. It is not necessary to have any previous knowledge of the object program in order to implement a sampling monitor for this purpose.

If one is interested in overall system activity, however, things are a bit more difficult. The system tables of interest must then be probed and pertinent information assembled. This can require a fairly sophisticated level of know-how.

A major selling point, of statistial sampling as a measurement tool is its relative ease of implementation. It is not normally necessary to modify existing program or operating system code. However, certain "protection" schemes, virtual memory, or hardware relccation features can make addressability to system and program tables difficult enough to warrant implementing the sampling monitor as part of the operating system.

When random intersample intervals are used, the programming is slightly more complicated and the overhead slightly higher due to the random interval generation process.

5.2.2.2 Credibility - The credibility of statistical sampling results is affected by sample density, length of monitoring, randomness of observation, interrupt handling, and experiment repetition.

Relative to sample density Kolence[25] states "given the randomness of observation, the accuracy of the data obtained is then a function of the number of samples taken, not the frequency of sampling." Confidence curves have been developed to indicate the expected accuracy as a function of the number of samples. See Figure 5-6. A discussion of the theory and development of the confidence curve equation can be found in our previous report.[1]

It follows, that the length of the monitoring period must be suitable for obtaining a sufficient number of samples to achieve the desired level of accuracy. The resolution of the timer limits the number of samples obtainable in a given unit of time. For example, a time with a one millisecond pulse limits the number of sample which can be taken in a one second run to 1000. If random intervals are used, even fewer samples can be taken in the same time period, since many of the intersample intervals will be greater than one millisecond and none can be less than one millisecond. It can be seen, therefore, that this techniques is not

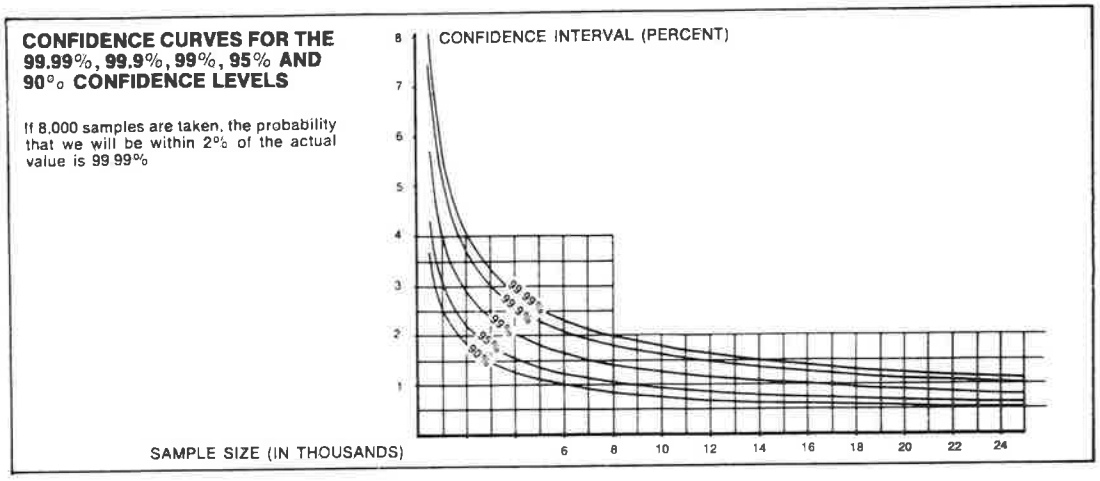


Figure 5-6. Expected Accuracy of Data as a Function of the Number of Samples

particularly appropriate to the examination of short running programs.

Sample density alone is not sufficient for accurate results. The statement by Cantrell and Ellison specifies that the sampling time schedule must be known to be "statistically independent of any natural execution pattern in the program", and Kolence also states "The key to a successful sampling technique is that a certain number of samples be collected randomly with respect to the variables being observed. That is, randomness in the sense that synchronization of sampling with the occurrence of some event in the system must be avoided." If synchronization occurs, events can be missed entirely. Experience has shown that on general purpose systems synchronization is actually rare with fixed interval sampling. Even so, using random length intersample intervals gives one more confidence in achieving the randomness necessary to assure the validity of the results. As mentioned previously, however, it is sometimes more difficult to obtain an adequate number of samples when using random intervals. In addition, computer program random number generators will begin to cycle after a period.* It is, therefore, necessary to carefully choose an algorithm which will not degenerate into a cycle quickly because synchronization with the cycle is also possible. Although, an argument could certainly be made that any randomness is better than none at all.

Designing the Random Sampling Process

The random sampling process is designed to be a Poisson process. This assumes that for a given time interval the number of samples that will occur in the interval is proportional to the length of the interval and the number of samples is independent of non-overlapping time periods.

The Poisson distribution states the if the mean intersample time is m, the probability of k samples in time t is

*See Knuth, "The Art of Computer Programming," Volume 2, "Seminumerical Algorithms" (Chapter 3) for an excellent discussion on computer random number generation.

$$\frac{e^{-t/m}(t/m)^k}{k!} .$$

Using this expression, the probability of no samples in time t is (setting k=0) $e^{-t/m}$. Thus, the probability of at least one sample in time t can be found by subtracting the probability of no samples from unity. The probability of at least one sample in time t can be restated as the probability that the next sample will occur within t time units and is given by

$$1-e^{-t/m} .$$

Calculating Random Intervals

For the sampling experiment, assuming we have a desired mean intersample time m, we want to have a different t between each sample.

Pr (at least 1 sample) = $1-e^{-tm}$ = a random number where $0. \leq$ random number < 1.

Rearranging the terms,

$$e^{-t/m} = 1\text{-random number.}$$

Since (1-random number) is also a random number, we can replace this term as

$$e^{-t/m} = \text{RNUM}$$

where RNUM is a random number. Finally, solving for t

$$-t/m = \log_e(\text{RNUM}) \text{ or}$$

$$t = - m \log_e(\text{RNUM})$$

Minimum Interval

Since the real time clock in the computer has a minimum time increment, we want to be certain that the intersample times are not less than the minimum interval. This can be accomplished by setting m, the mean intersample time, such that the probability

of having a sampling interval less than the minimum increment is close to zero. Or stated another way, we want to set m such that the probability that the sampling interval is greater than the minimum clock interval is close to one.

M can be set from the following:

We want Prob (sampling interval > minimum clock increment) = .99

The probability density function corresponding to the cumulative probability density function $(1-e^{-tm})$ is $(1/m)\ e^{-t/m}$, so we want to set m such that

$$\int_{\min}^{\infty} (1/m)\ e^{-t/m}\ dt = .99$$

where min is the minimum clock increment.

Solving for m

$$-e^{-t/m} \int_{t=\min}^{\infty} \geq .99$$

$$0 - (-e^{-\min/m}) \geq .99$$

$$-\min/m \geq \log_e (.99)$$

or

$$m \geq \min/\log_e (.99)$$

Sample Size

Assume that we want p, the percentage of time spent in a given range to be within 5 percentage points of the true value, P, the desired range is then $P\pm5$. Since p is assumed to be normally distributed about P, it will lie in the range $(P\pm2\sigma_s)$ with probability .95. Since

$\sigma_s = \sqrt{PQ/n}$ (where Q = 100-P), and n = number of samples

we may put $2\sigma_s = 5$,

$$2\sqrt{PQ/n} = 5 \text{ or } n = \frac{4PQ}{25} \qquad (1)$$

It is obvious at this point that n depends on a property of the process to be sampled, namely P. However, if a range of values can be estimated for P, then we can calculate the corresponding value of n over the range and select the largest n. If the range for P is estimated to be 45-65%, n is a maximum of 400 for P = 50%. The following table summarizes the appropriate values for n at the 95% confidence level under varying acceptable ranges for p.

ACCEPTABLE RANGE	SAMPLE SIZE REQUIRED
$P \pm 1$	10,000
$P \pm 2$	2,500
$P \pm 5$	400
$P \pm 10$	100

Setting the Mean Intersample Time

The selection of m can be done by dividing the length of the sampling period by the number of samples required. This number should then be compared with the m calculated under "minimum interval". If it is larger than the feasible minimum m, the sampling period might be lengthened so that the required number of samples could be obtained. If the sampling period cannot be lengthened (i.e., the program terminates in a finite length of time), then a calculation can be expected with the limited number of samples which can be made.

Re-arranging the terms of Equation (1)

$$\pm r = 2\sqrt{PQ/N}$$

The adjusted range ($P \pm r$) would be the maximum accuracy that could be obtained for the given confidence level of 95%.

When the sampling technique is applied to overall system measurement, the host system's interrupt handling procedures can have a subtle affect on the results. In large systems, some of the operating system (Executive) routines may be designed to run with all, or a class of, interrupts "disabled". This means that should an interrupt occur during execution of such a routine, it would not be processed until the routine has completed execution. The sampling process is directed by an interrupt, and therefore, could be disabled for certain routines. The affect is that those routines will never show up in the sampling results even though they may be significant time consumers. There is no way of knowing that this will occur unless one is familiar enough with the system in question to know whether the interrupt handling procedure makes this possible, and if so, which routines operate in this fashion.

Repetition of any measurement experiment is important in order to assure that the results are truly representative of the program or system activity.

In order to assess the credibility of the sampling technique under real-world conditions, the following experiment was conducted. The event monitor measurement package, previously described, was used to obtain time stamps on each entry to, and exit from, every subroutine exercised during the course of a simulation program execution. A 125 microsecond time source was used. This was adequate to time the smallest subroutine in the simulation, but the average instruction time on the host conputer was approximately four microseconds. This implies that a timing error will be introduced in those routines which call many other subroutines in a short period of time. A look at Figures 5-5 and 5-5a suggests that subroutines EVNTS, ETAB, FILEM, SET, and TMST are candidates for such error. The clock was stopped during all event monitor processing in order to eliminate distortion in the results due to measurement instrumentation. It was also necessary to stop during all simulation program initiated I/O in order to force the results to conform to those obtainable through sampling. (It is difficult, sometimes impossible, to link the I/O activity with the initiating

subroutine when sampling.) By stopping the clock during problem program I/O, we are timing only the execution of the actual simulation code which can be easily compared with sampling output. This, however, suggests that we will experience some timing error in those routines with heavy I/O operations, again due to clock resolution (when the clock is stopped, the remainder of the 125 microsecond interval is lost). The suspect routines in this case are DATIN and HPLOT. We are now prepared to consider the event monitor results as representing the actual operation and timing of the simulation program, within acceptable tolerences; and we know in which subroutines we may expect some error.

The same simulation program was subjected to measurement utilizing sampling techniques with both fixed and random intersample intervals. The sampling output consisted of the problem program interrupt addresses recorded at the expiration of the timer intervals. The interval timer available for sampling had a one millisecond pulse resolution. All measurement runs were repeated several times for confirmation.

Post processing programs were developed to generate time density distributions and other reports, from the output data of both the event monitor and sampling packages. Subroutine elapsed times were accumulated from the event monitor output and time percentages were computed from the totals. A frequency distribution was developed from the sampler output using the subroutine boundary addresses as limits, and time percentages were also computed. These particular measurements were not affected by disabled interrupts. There was roughly a 10 to 1 ratio of I/O time to actual simulation time. As a result, the absolute non-I/O elasped time of the simulation code is in the neighborhood of 3 seconds. This did allow sufficient time to collect enough sample data to have some confidence in the results, even though the portion we are interested in constitutes only 10% of the total activity.

Figures 5-7 and 5-7a are computer drawn graphs which show the random interval and fixed interval sampling results plotted against the baseline event monitor results. (Note that the scale

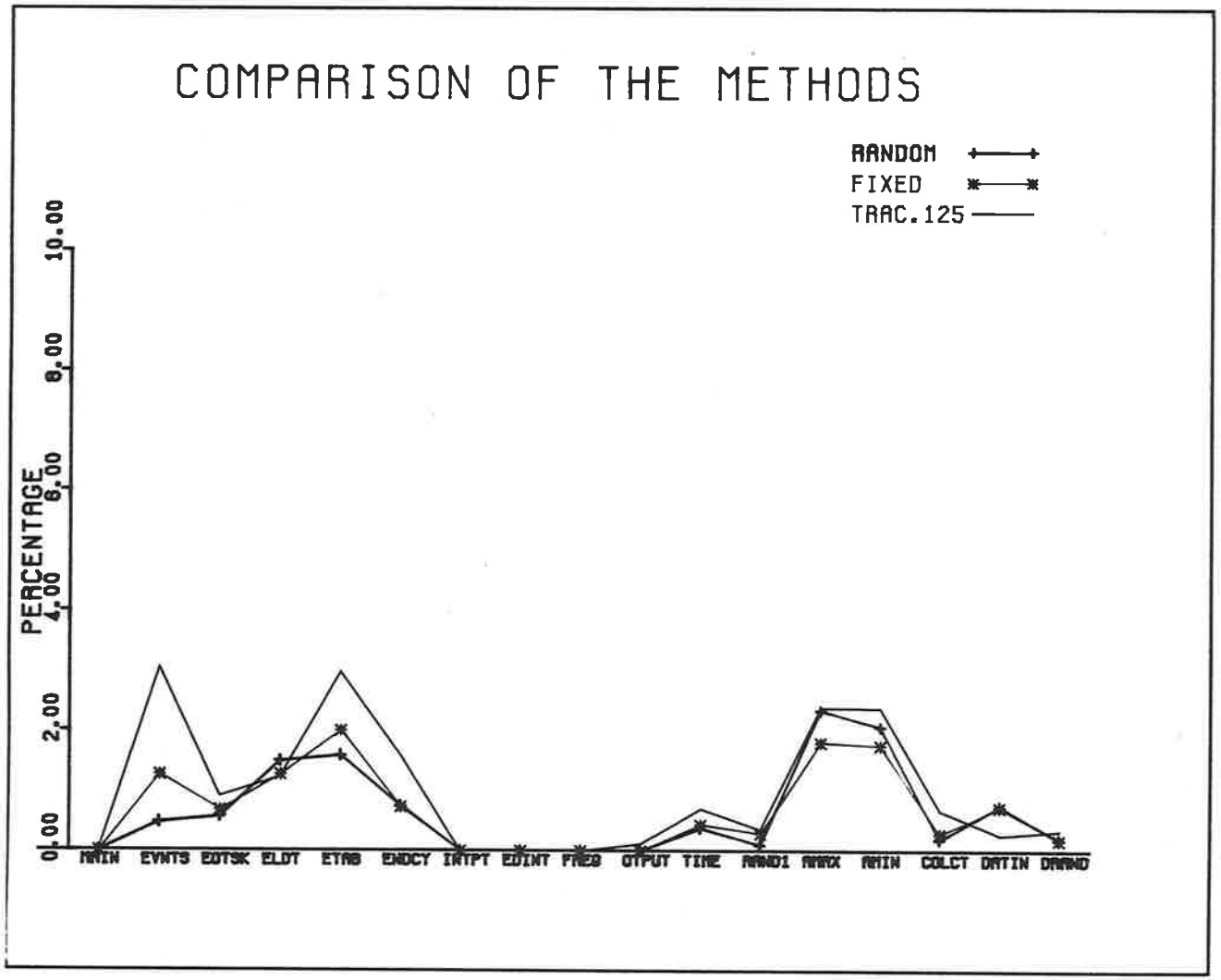


Figure 5-7. Event Monitor Results (Trace)
vs.
Random Interval Sampling
vs.
Fixed Interval Sampling

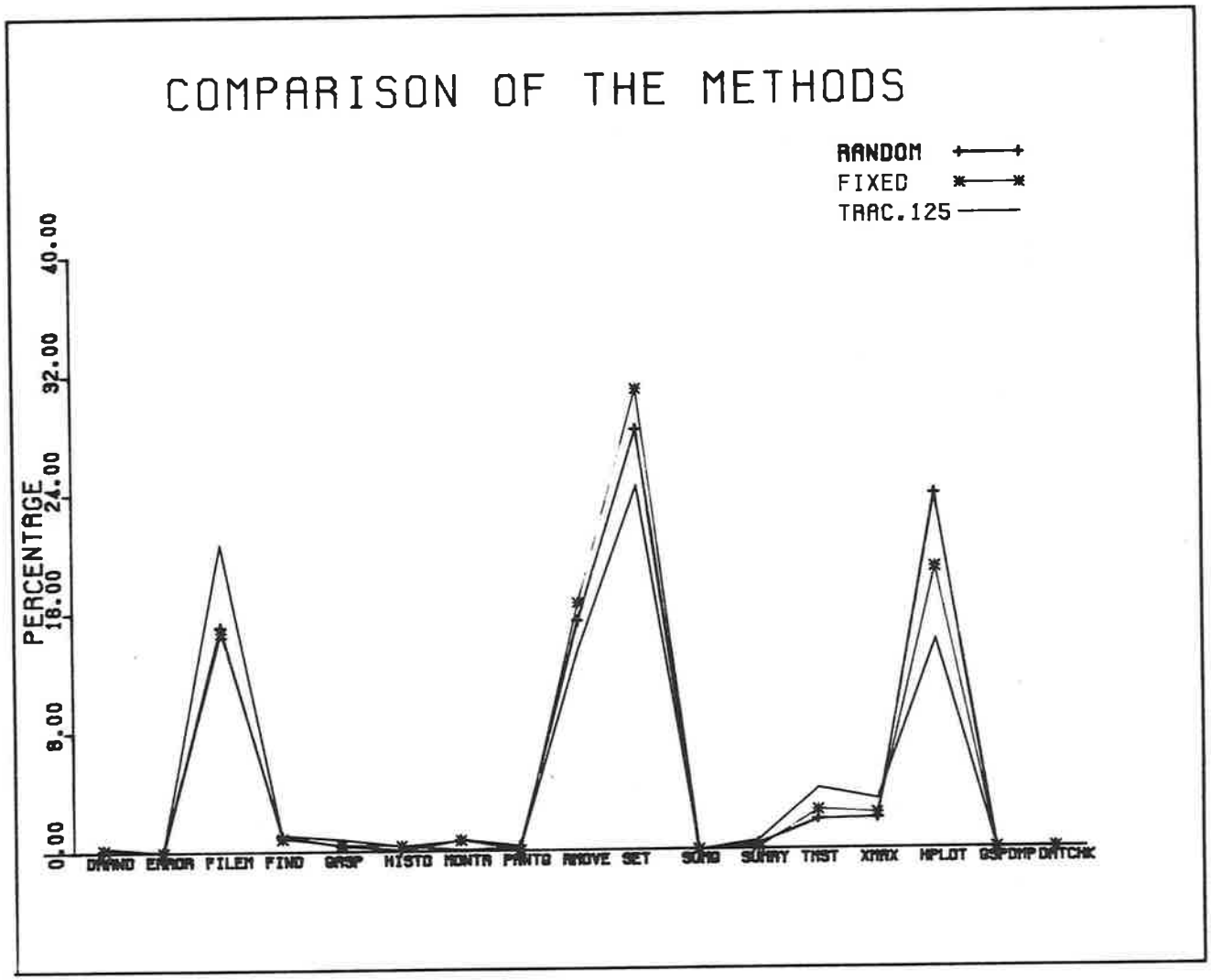


Figure 5-7a. Event Monitor Results (TRACE) vs Random Interval Sampling vs fixed Interval Sampling

is 10% on Figure 5-7 and 40% on Figure 5-7a.) In general, the sampling results are very good approximations. The areas where the largest discrepancies occur are in subroutines EVNTS, ETAB, DATIN, FILEM, SET, TMST, and HPLOT. This was expected; as the event monitor data for these routines should have some distortion introduced by the resolution of the clock and the stop-clock procedure. The differences between the fixed interval results and the random interval results is mostly due to the fact that fewer samples were obtainable with the random intersample interval, and therefore the level of confidence must be lower. There were approximately three times the number of samples taken with the fixed interval runs as there were with the random interval. The expected confidence level for the overall program time is 99.9% that the fixed interval results are within 1-1/2% of the true distribution, and 95.0% that the random interval results are within 1-1/2%. That confidence level applies to the total running time of the program. The simulation code accounts for only 10% of the total. We are essentially magnifying that 10%, and, therefore, should expect that the confidence level is slightly less than it otherwise would be.

As an illustration of the importance of sample density, consider Figures 5-8, 5-8a and 5-9, 5-9a. On Figures 5-8, 5-8a we have plotted the results of a sampling experiment using a 2 millisecond fixed interval and that of a sampling experiment using a 3 millisecond fixed interval against the baseline event monitor data. There were approximately 13,000 samples taken with the 2 millisecond interval and 8,600 samples with the 3 millisecond interval.

Figures 5-9, 5-9a represent the results of sampling runs made with only random intersample intervals. The difference in the runs is in the value of the "mean" used for the random interval generator. The mean values were two and three and the resulting sample densities were approximately 6,600 and 4,700 samples respectively.

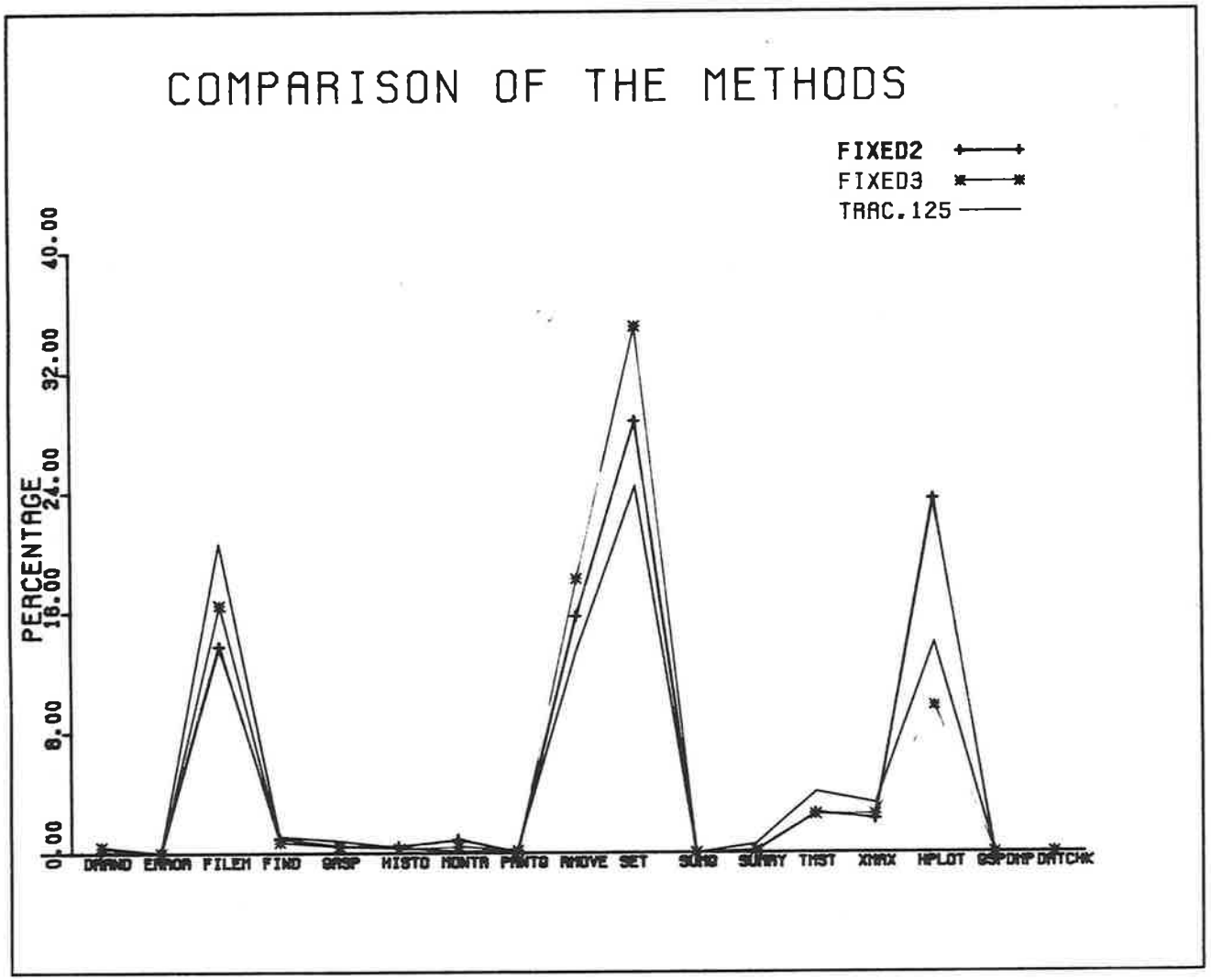


Figure 5-8. Event Monitor Results
vs.
Fixed Interval Sampling, (2 Millisec Interval)
vs.
Fixed Interval Sampling, (3 Millisec Interval)

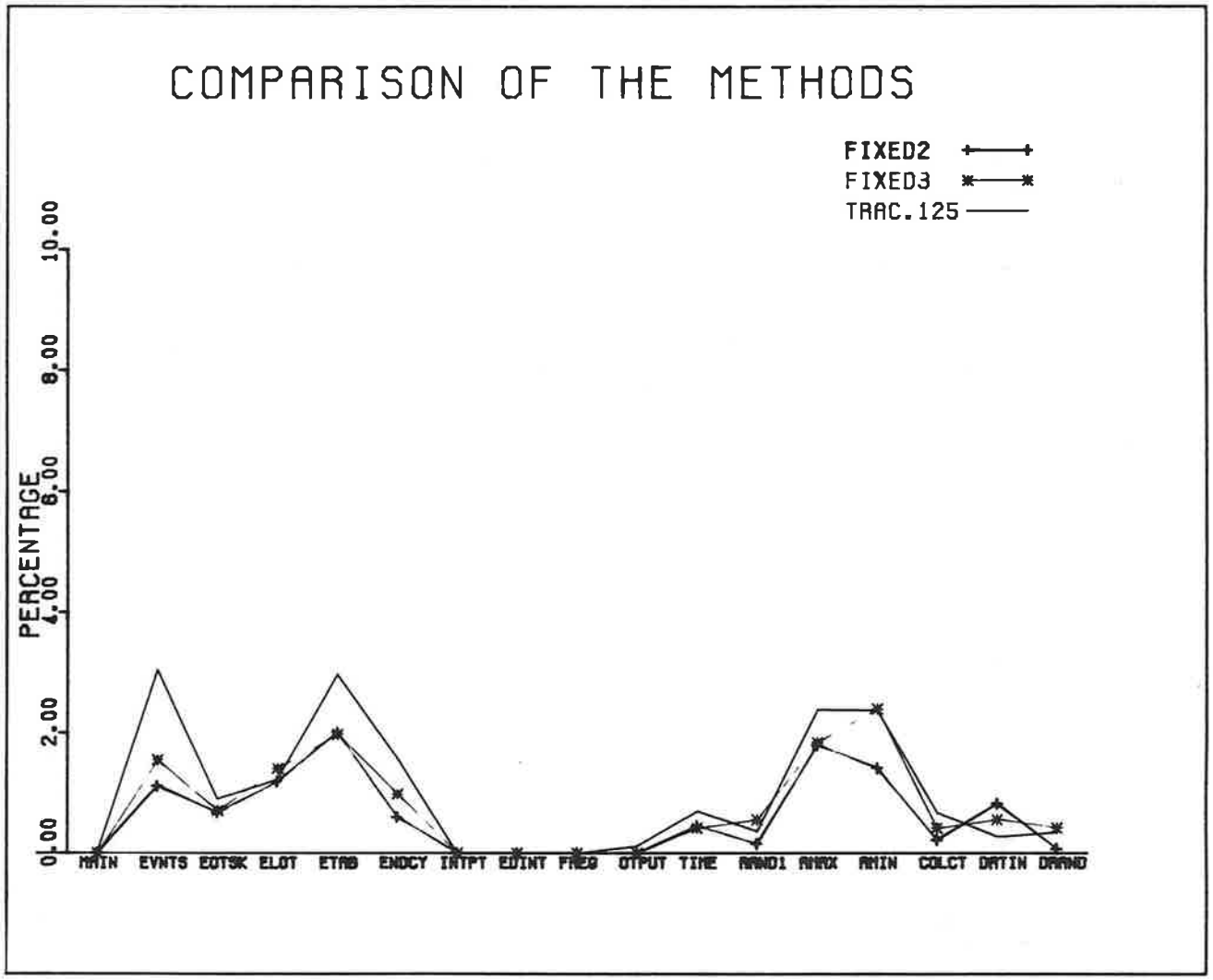


Figure 5-8a. Event Monitor Results
vs.
Fixed Interval Sampling, (2 millisec Interval)
vs.
Fixed Interval Sampling, (3 millisec Interval)

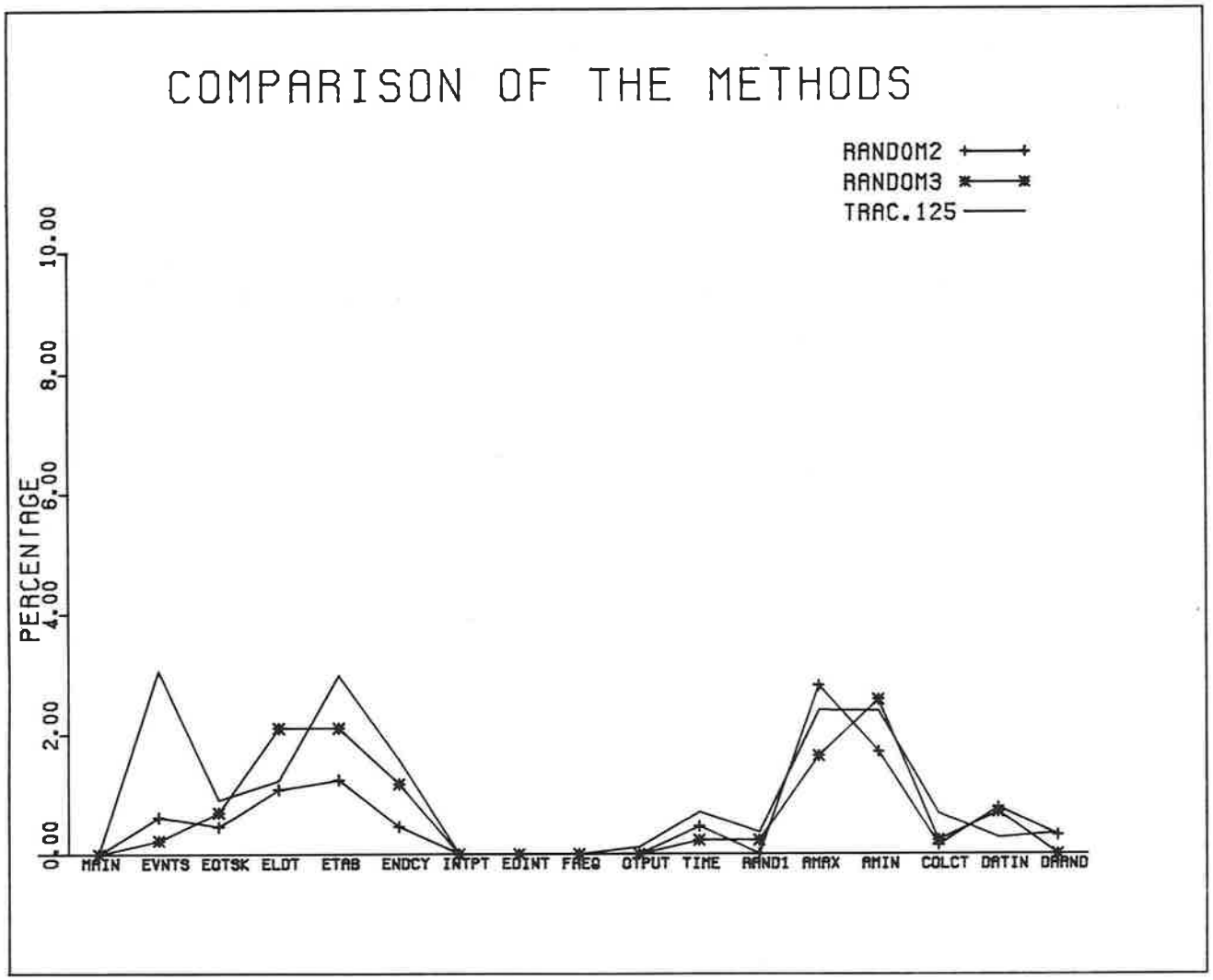


Figure 5-9. Event Monitor Results
vs.
Random Sampling, Mean Interval = 2 msec
vs.
Random Sampling, Mean Interval = 3 msec

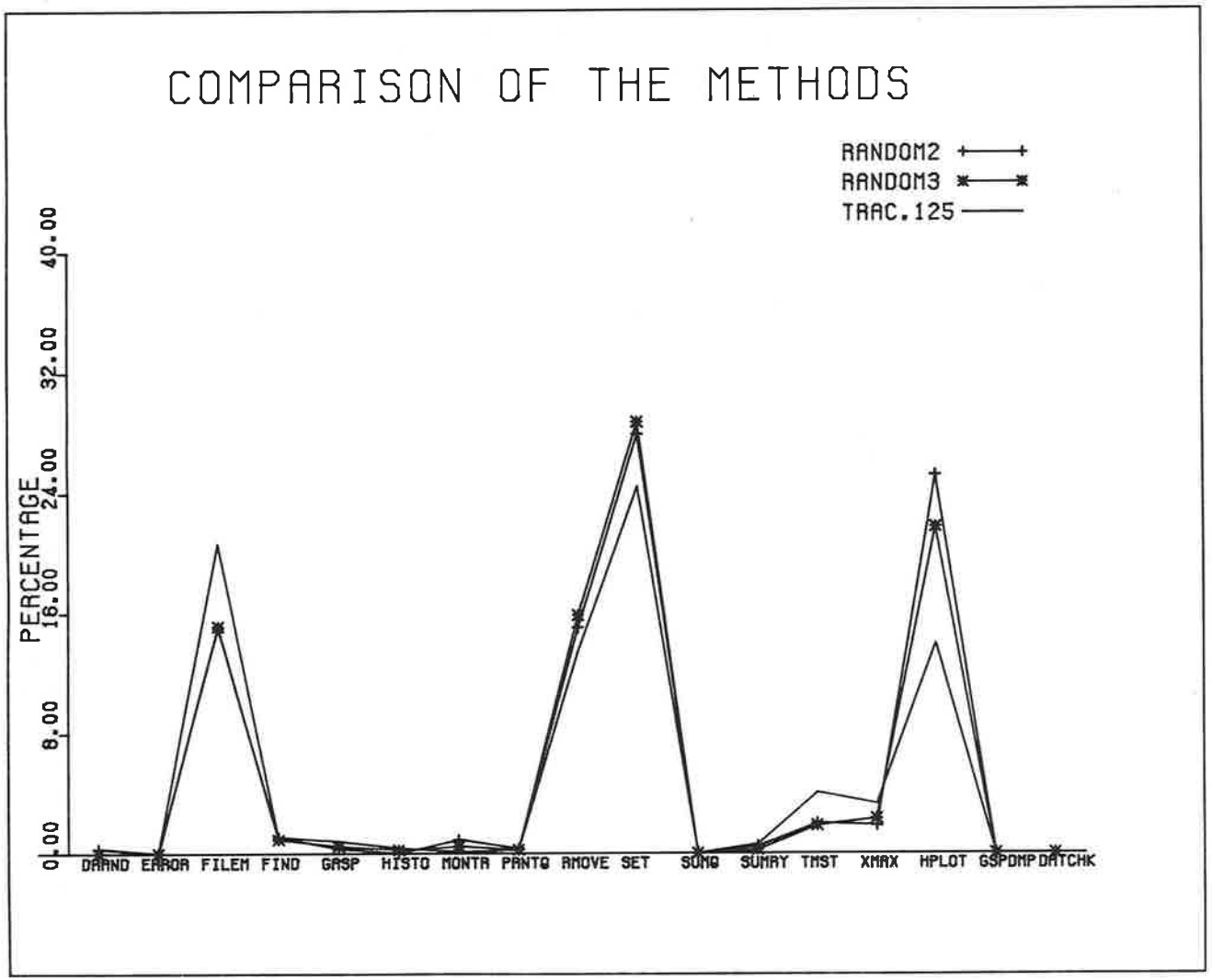


Figure 5-9a. Event Monitor Results
vs.
Random Sampling, Mean Interval = 2 msec
vs.
Random Sampling, Mean Interval = 3 msec

In summary, it appears that the sampling technique is quite credible for identifying where the concentrations of time exist in a program; and, when used with caution, can also be valuable for that task when applied to an entire system, provided that the caveat concerning randomness and sample density is heeded.

5.2.2.3 Volume of Measurement Data - The volume of sampling data output is usually measured in thousands as opposed to possibly hundreds of thousands with event driven techniques. Moreover, the analyst has the ability to keep the sample density within limits by manipulation of the length of the intersample interval. A manageable volume of measurement data is one of the desireable features of sampling techniques.

5.2.2.4 Extent of Information - The basic element of information available through statistical sampling of computer programs is an indication of the distribution of time. It is possible to resolve program activity down to the single instruction level. If the system on which the program is executed is data rich, it is sometimes possible to implement a traceback procedure which will correlate I/O time spent by the system on behalf of the program with the initiating subroutine or sequence of code. It may also be possible to probe operating system maintained status information tables to determine the activity of peripheral devices and channels, the size of queues, the location of disk heads, etc. In short, any information on the status of activity which is available to software can be sampled. It is possible, then, to determine percentage utilization of resources as well as distribution of time. Such information can be used to determine whether resources are overloaded, misused, not balanced, not used, etc. Changes in resource utilization as a result of system modifications can be studied with the aid of sampling measurement.

Sampling can be used to discover the fact that inefficiency exists. It does not, however, necessarily identify the reason for the inefficiency. This technique suffers from an inability to measure sequences of events, classes of events whose rate of

occurrence exceeds the sampling rate, absolute duration of events, and fluctuations in elapsed time.

5.2.2.5 Resource Requirements - In general, sampling requires use of all the same resources as event monitoring. However, it requires less of each. Less processor time is required because activity is sampled at intervals rather than monitored for every change. Less memory space is required for buffers and less channel time for data transfer because the volume of data is lower and the frequency of recording is usually moderate compared to that of event monitoring. A dedicated tape drive is usually required, although data can be accumulated elsewhere and processed or dumped at the termination of the monitoring period.

5.2.3 Practical Implementation Considerations

There are a number of practical problems which must be addressed when implementing software monitors regardless of the chosen measurement technique.

Initiation and termination of the monitoring facility must be dealt with. Initiation for obvious reasons; termination in order to complete processing, dump buffers, and to close-out (end-file) the output data set on the recording medium.

One approach is to implement the monitor as an integral part of the system with all necessary monitor code present, but dormant, until initiated through an operator action. A common method of initiation, in this case, would be the setting of a console switch during the system load procedure. The system initiation routine would be modified to interrogate the switch and to take the action necessary to cause monitor initiation, if desired. Termination could be controlled in a similar manner, through a different switch or by resetting the original switch. Another method would permit the operator to interrupt a system in operation with a keyboard request to begin monitor activity. This method would require that the monitor code be available to the system as a load module with all references to system tables, etc. previously resolved. Again, termination procedures would be signalled in a

like manner. In both cases, provision must be made within the system for communication with the outside world, either through console switch testing or keyboard interrupt (break key) techniques.

When monitoring only problem program code, the monitor is usually loaded as part of the problem program; either as a result of an actual reference to it from the problem program, or through system facilities which permit the monitor to link-to and control problem program execution. Termination for the former method is usually accomplished through a closing reference from the problem program code or by dynamic implantation of branch instructions by the monitor. Termination in the case of monitor link and control procedures is effected naturally as part of the facility.

The allocation and handling of monitor data buffers is also an important consideration. Monitor record data transfer to a peripheral device should be overlapped, with the execution of the monitored program or system, in order to minimize processing time overhead and total system degradation. If monitor output is not overlapped, the monitored process waits while each record or group of records is transferred to the assigned peripheral device. The speed of most peripherals is several orders of magnitude slower than processor speed so the overall effect can be devastating.

If monitor output is overlapped, buffer allocation and handling schemes must be developed. The size of the buffer(s) must be adequate to accommodate a reasonable amount of monitor data. Reasonable is defined as sufficient to minimize either data loss or wait time if the rate of monitor data recording exceeds the rate at which the buffer(s) can be emptied by output routines. The buffering scheme and size of buffers chosen also depends on the availability of memory space.

Double buffering is a good choice when memory is available and a high degree of overlap is desired. This scheme causes the output routines to process each buffer when it is full, while the monitor program continues to operate by placing records in an alternate buffer. Synchronization of input and output buffers is necessary to avoid data loss.

A circular buffer may be used to conserve memory space. When circular buffering is used, the monitor places data in the buffer, while at the same time the output routine is stepping through the buffer, transferring previous monitor records to the output device. When the end of the buffer is reached by the monitor, the next record overwrites the first record of the buffer, and the cycle is repeated. Consideration must be given to the possibility of the monitor overrunning the records being processed by the output routine; and to the possibility of the output routine catching-up with the monitor.

The capacity of the output medium has an effect on the measurement process. If the end of an output volume (tape or disk) is reached during the monitoring period, some action must be taken to close-out the volume, and terminate or continue with the measurement. This action can result in delays, loss of data, or both. Ideally, a second tape device can be "switched" in to receive data while the original tape is being closed-out and re-wound.

Data compaction techniques can be applied to minimize the buffer memory requirements and to maximize the volume of data obtainable before end of volume or end of data set conditions are encountered. Data compaction involves "coding" the data wherever feasible and packing the information as closely as possible into a word or group of words for later deciphering and processing. This means that each item of information takes up less space in memory, as well as the output medium. Blocking the records on the output medium reduces the number of interrecord gaps and therefore makes fuller use of available capacity. Data compaction, of course, adds to the overhead of the monitoring process.

Dynamic processing and periodic summarizing of the measurement data can be performed on-line to minimize output. To do so, however, adds a task to the system which of course requires processing time and other resources. Post processing of data has the disadvantage of dealing with potentially large volumes of data, but permits a given set of measurement data to be combined in various ways and analyzed from different points of view.

5.3 OTHER SOFTWARE MEASUREMENT TOOLS

In addition to Software Monitors, which consist of code embedded in an existing system, there are other software measurement tools which can be employed in computer systems evaluation. Among these are Self-Simulators, Benchmarks, and Synthetic Programs. These tools fulfill a purpose different from that of Software Monitors.

Benchmarks and Synthetic Programs are used to compare the relative suitability of several machines or machine configurations for a particular application or job environment; while self-simulators are used to collect in-depth information about the make-up and operational characteristics of particular program or task.

5.3.1 Self-Simulators

Self-Simulators are programs which interpretively execute another program and simultaneously gather statistics about its instruction mix and branching patterns. The simulation program simulates the host computer on itself. The key element for implementation of such a program is the availability on the host computer of an "EXECUTE' instruction which allows the execution of an instruction out of its normal sequence of code. Thus, the simulator steps through the program under study, examines each instruction, increments counters indicating which instruction type is being executed, indirectly "EXECUTES" the examined instruction and proceeds to the next instruction in sequence. Special processing is provided for instructions which would interrupt the normal sequence of events, such as "BRANCH" or "JUMP" instructions, and additional statistics may be gathered about such instructions; such as the "conditions" which caused this jump on conditional jump instructions, the location of the jump instruction, and the location to which it jumped.

It is important that the simulator occupy minimal space and be able to retain control in spite of error, interrupt or other conditions which may arise from the object program.[22]

The data collected through self-simulators is valuable in many program analyses including larger simulations in which the analyzed object program is but a part. Generally, this tool extends the natural running time of the object program by a large factor.

5.3.2 Benchmarks

A benchmark is an existing program which is coded in a specific language and executed on a machine being evaluated for use in a particular application or job mix environment. Benchmarks are developed primarily to compare one machine configuration (hardware and software) against another for possible acquisition, rather than for exhaustive performance analysis of machines already employed. A comprehensive series of benchmark runs can demonstrate differences in machine organization and evaluate the performance of I/O equipment and secondary storage as long as a variety of instructions are used by the test programs.[26]

In order to be effective, benchmarks must be carefully chosen to be representative of the application or job mix for which a machine is being selected. Performance predictions for problem areas which are not benchmarked can be risky extrapolations. It takes a large effort to program benchmarks and to prepare realistic data. Benchmark performance can be misleading in terms of capacity of a system unless complete benchmarks are selected to place a capacity load on that system.[27] Even large benchmark jobs must be recognized as models simulating the real world.[33]

5.3.3 Synthetic Programs*

Like a benchmark, a synthetic program is one that is coded and executed, but it differs in that it does not necessarily exist beforehand.[26] A synthetic job is a fabrication which strives to

*This approach has been taken by TSC to evaluate computer architectures for future ATC use. The synthetic programs have been imcorporated into a synthetic system which is a stylized version of the terminal ATC system and is known as the "ATC TEMPLATE".

represent an application (or job mix) by imitating the type(s) of data processing which characterize it. It is a stylized version of the function(s) for which a computer is being evaluated. It may be simply a well-behaved exerciser of system features, or a tool for comparing the speed of dissimilar systems. Requirements of such a job are: 1) that it can be stated as a machine indepentent procedure. 2) that it be meaningful over quite a wide range of computer systems, being neither too trivial for the larger ones nor too complex for the smaller ones. 3) that it should be long enough to be measured accurately, yet not so long that measuring becomes burdensome. Consequently, the procedures should be cycles with the running time directly proportional to the number of repitetions, 4) that it be simple enough to be readily re-programmed in different languages for different machines.

In order to keep the programs simple enough so that they can be readily re-programmed for various machines, details of the application being represented are intentionally supressed. Hence, performance of a system on a synthetic program cannot be used directly to predict the running time of a specific application accurately. The relative performance of two systems on a synthetic job, should, however, yield a reasonable first approximation to their relative performance on a specific job using the same system facilities.[28]

The major advantage of synthetic programs in the flexibility they provide, since jobs can be designed to include almost any desired measurement parameters.[26] They suffer, however, from some of the same problems as benchmarks in that it is difficult to realistically characterize the real-world applications and data, and a significant amount of programming effort is usually involved.

5.4 HARDWARE MONITORS

Hardware monitors are devices that use signal collection probes physically attached to defined acquisition points of computer circuitry. These probes feed counters or drive time accumulation registers making it possible to count the occurrence of events or time their duration. The values accumulated in the

counters and registers can be periodically copied to magnetic tape, or graphically represented on attached plotting equipment.

The capability and features of commercially available hardware monitors vary a great deal. For example, monitoring can be continuous, sampled at fixed intervals, and/or sampled at intervals stimulated and controlled by external signals. In many, individual signals can be logically combined (and/or/ex or/not) to filter the probe data. In some, a "Fanout" feature is provided to disperse the same signal to various monitors or counters.

The following features should be considered in selecting a hardware monitor. Each of them will affect the utility of the monitor for a particular measurement effort. Some monitors are modularly assembled so that capability can be incrementally increased. Others merely vary in capability from model to model or vendor to vendor.

1. Number of probes
2. Number of standard counters
3. Number of electronic counters
4. Resolution of standard and electronic counters
5. Number of logic panels
6. Type of decoders (hex, octal)
7. Real time clock availability
8. Availability of data tags or other mechanism to identify files or record groups on output medium.
9. Capacity of output medium
10. Bit width of counters and accumulators
11. Availability of Comparator facility
12. Bit width of comparator registers
13. Number of probes input to comparator logic
14. Ability to combine several monitor's probe data for output to a single medium. (Multiplexor)

15. Range of adjustable voltage threshold levels for different computer system logic levels
16. Changeability of probe tips to accommodate different computer system pin attachments
17. Ability to logically combine probe signals
18. Ability to Fanout signals
19. Selectable output frequencies
20. Other mechanism for periodic output (Buffer full, counter full)
21. Buffered output capability
22. Recording while outputting capability
23. Output recording speed
24. Maximum cable length for probes
25. Probe sensitivity to signal duration (typically 30 nsec) and repetition rate (10 - 15 MHz)
26. Minimum sampling interval selectables
27. External probe stimulation capability
28. Ability to optionally count multiple occurrences of event rather than every occurrence
29. Ease and Flexibility of monitoring initiation and termination
30. Availability of Data Reduction Software
31. Compatability of Monitor output medium with available computer data reduction facilities
32. Portability

Classically, hardware monitors are used to measure hardware features, many of which are difficult or impossible to measure by software means. Typical hardware monitor measurements include:

Processor active/idle time
Channel busy/idle time

Control unit busy/idle time
Processor/Channel overlap
Multprocessor overlap
Device busy/idle time
Device use overlap
Time in program state
Time is Supervisor state
Interface activity
Disk head seek time
Frequency of interrupts (by class)
Memory queue time by processor
Memory bank usage

With the addition of comparator and decoder logic, hardware monitors can be applied in a limited manner to many measurement tasks formerly considered the exclusive province of software monitors. These include program elapsed time, program execution frequency, and data base activity.

Attached to the memory address register a properly equipped hardware monitor can be used to trace program and subroutine activity, overlaying or paging. Attached also to the data register, it might be used to study the response time for terminal requests, or to time the various aspects of a transaction such as time to process, queue time, formatting time, or message turnaround time. Attached to the instruction register, utilization statistics can be gathered on the types of instructions most frequently employed by the system under study.

The use of hardware monitors for software related measurement is limited by the number of probes, complexity of the combinatorial logic, comparator and decoder capability, and machine architecture (paging-segmentation), as well as the analysts understanding of the subtle details of the software in question.

The use of hardware monitors for any measurement requires an intimate knowledge of the circuitry and architecture of the computer to which it is being applied. Hardware monitors provide the distinct advantage of not perturbing the system under study. There

is no deviation of normal program flow or timing (software overhead). Further, in a static real-time system such as ARTS, all programs and data are memory resident and therefore more easily accessible to a hardware monitor than in a dynamic General Purpose system.

Measurements can be made of hardware activity which would not necessarily be available to software monitors. On the other hand, a knowledge of computer electronics is required for installation, and if software related data is desired, an intimate knowledge of the subject software is also necessary along with particulars on the computer architecture.

Test procedures (hardware or software) must be devised to verify proper placement of probes. The computing system will be "unavailable" during probe attachment and checkout. The amount of information obtainable in one measurement run is usually more limited than that obtainable through software techniques; and hardware is considerably less portable and more costly to duplicate than software.

5.5 HYBRID MONITOR

Hybrid monitors are an attempt to combine the desireable features of both hardware and software monitors in one powerful measurement tool. Typically, a separate processing unit with peripherals is attached to the host computer system through a memory port or I/O channel.[19,29] Measurement software, with all its inherent flexibility runs on the attached computer and accesses pertinent measurement data through system tables in main memory. The data can be dynamically recorded, displayed, summarized or otherwise reduced by the attached system without interfering with the normal operation of the computer system under study. Special features might include the ability to stimulate an event or to obtain measurement assistance from the host system.

More recently, mini-computers have been employed to drive and control hardware monitors.[30] Such arrangements grew out of a need to obtain information directly, at a more primitive level,

in greater quantity, at higher bandwidths, and with more convenient and accessible output facilities than normally possible with ordinary hardware monitors. Basically, the mini computer console, display equipment, etc. are used by the analyst to select (interactively) through computer programs the events to be monitored via hardware probes attached to the host system. Programmable logic and registers which can be set and read by either the mini-cpu or the external environment (host computer) form the main communication link. This type of hybrid configuration also has the potential for interaction with software monitors operating in the host system. Thus providing a complete, flexible, and extremely powerful measurement facility with access to virtually every pertinent measurement variable.

The drawback in hybrid monitoring is the necessity for the presence of additional processing capability (mini computer) coupled with the fact that such hybrid facilities are not yet commercially prolific. While this approach represents the ultimate in computer system measurement technology, there may be an associated software development effort for the mini computer monitoring facility over and above the actual measurement software, and specialized interface hardware may have to be developed for a particular project. In addition, all the expertise required to implement effective, workable, hardware and software monitors is also necessary. The overall development time for a good hybrid measurement facility would generally be longer than for either a basic hardware of software monitoring activity.

5.6 FIRMWARE MONITORING

The current trend towards computer systems with alterable microprogram control logic makes available yet another measurement technique which we will refer to as firmware monitoring.

Microprogramming is a technique for implementing the control function of a digital computing system as sequences of control signals that are organized on a word basis and stored in a memory unit.[31] The microprogram memory is usually significantly faster than the main memory. If this memory is alterable then

microprogramming allows the modification of system architecture as observed at the machine language level. That is, instructions may be modified, or added to the repetoire, by providing micro-programs to properly handle them.

There are several ways in which the alterable microprogram feature can be used as a measurement tool. One approach is to add special instructions or "hooks" at strategic points in a program. These hooks are interpreted by the microprogram as NOPs (no operation) when measurements are not being taken. Otherwise, they may cause the microprogram to increment a special counter (or dedicated memory address) or to cause the clock cycles to be counted in order to time the duration of an event.

Another approach would not require special instructions in the program code, but would modify the microprograms used for the normal instruction set to perform additional measurement functions. For example, the source and target addresses of every BRANCH instruction executed might be recorded, thus providing a trace of program flow.

The fact that microprograms could be used to aid the measurement process means that such measurements would distort the measured system to a lesser degree, measurement overhead would be reduced, and measurements which would be difficult or impossible to obtain with straight software monitors would become tenable.

6.0 SUMMARY

6.1 THE ROLE OF MEASUREMENT IN THE ARTS III PROGRAM

We have suggested that direct computer measurement can play a useful role in the evolution and life of the ARTS III systems in a number of ways, they are:

1. Management Information - Improved Planning

   Direct Measurement of operational system characteristics would provide thorough and accurate data for use in simulation, system operational analyses, or other ATC related studies, whose results are ultimately used to determine the pattern of expansion in ARTS III systems.

2. Reconfiguration Evaluation

   As hardware elements and ATC functions are added to the ARTS III systems, performance characteristics of the systems will change. Direct measurement should be used to assess the impact of such modifications, to verify or refute anticipated results, and to obtain new data relative to overall operation. In this way the growth of the ARTS III systems can be guided in an orderly, cost-effective manner.

3. System Performance Evaluation

   In addition to monitoring the ARTS systems for purposes of management and planning, measurement activities can be directed towards the analysis and improvement of current computer system performance. The identification of system bottlenecks and their causes can point the way to software or hardware alterations (often minor) which can increase the performance and subsequently the life of a system. In general system "tuning" activities on large systems have been highly successful. Reported improvements of 30% are not uncommon. A General

Accounting Office report states that federal dp costs could be cut "several million dollars a year" if performance measurement tools were used more extensively.*

4. Future Program and System Design

   Major inefficiencies both hardware and software disclosed through proper measurement and evaluation can be avoided in future programs or systems, if not correctable in current ones. If performance bugs are not ferreted out in current systems, they may be carried over into future designs.

5. Program Development

   Measurement tools can be applied to task development activities to provide the programmer with the necessary data to debug and tune his individual programs prior to their integration with the system as a whole. This is not a substitute for total system performance evaluation however, because a computer system is not well represented by the sum of its parts. The performance of each part of a system will, generally, give little insight as to overall system performance.

An active measurement and evaluation effort, therefore, can provide the FAA with valuable insight, and the basic data necessary to make cogent decisions regarding the direction of the ARTS III systems. It can also provide the concrete information necessary to guide the progress and evaluate the results of contractor efforts, both in development and in simulation.

6.2 APPLICABLE MEASUREMENT TECHNOLOGY

There are a number of measurement tools and techniques which might be applied in a measurement activity: Software Monitors, Hardware Monitors, Firmware Monitors, Self-Simulators, Benchmarks, and Synthetic Programs. Each has positive and negative aspects

*From DATAMATION, January 1973, Page 91.

which must be weighed against the measurement objectives and environmental restriction.

Software monitors can be arbitrarily separated into two classes: Event Monitors and Statistical Samplers. Event monitors involve modification of existing program code and therefore require intimate knowledge of the task or system under observation and, very often, the basic architecture and operational philosophy of the host computer, as well. Overhead, resource requirements, and volume of output can be considerable, implementation can be tricky, and in certain cases the environment should be controlled and reproducible. Event monitors are necessary however, for in-depth performance analysis, and system tuning efforts. With Event monitors, sequences of events can be traced and algorithmic or performance bugs can be ferreted out. A high resolution timer in necessary for fine grain event-oriented measurement.

Statistical Samplers are usually easier to implement since they require little or no modification to existing code. Their processing time overhead is considerably less than most event monitors. A specific hardware requirement is the availability of a program addressable interval timer which will cause an interrupt at the end of a pre-set time interval. Knowledge of system logic and familiarity with system tables is required if overall system performance is to be studied. Statistical sampling can provide useful overview data such as task time density distributions and large resource usage percentages. It does not provide sufficient information to pinpoint cause and effect relationships or to analyze sequences of events. The interrupt protocol inherent in many systems can cause loss of information and subsequent distortion of results. A random, asynchronous sampling process and adequate sample density are of paramount importance for credible results.

Hardware monitors obtain measurement data through passive probes which do not interfere with system code or activity. They are primarily important for obtaining hardware oriented measurements which are difficult or impossible to obtain through software means. Certain measurement data can also be elicited with regard to software activity, such as elapsed time of program modules, utilization

statistics on key data sets, and turnaround time of man-machine interface messages. Hardware monitors are usually equipped with their own recording facilities. Measurement data acquisition during a given hardware monitoring period is limited by the number of probes, combinatorial and comparator logic restrictions, and the capacity of the output medium. Intimate knowledge of the host computer architecture and circuitry is required for implementation.

Hybrid monitors combine the passive, non-perturbative aspects of hardware monitoring with the flexibility and information gathering/processing strengths of software monitoring, thereby achieving a fairly comprehensive and desireable measurement tool. A separate computer (mini) with attached peripheral devices is required, and a significant amount of software development work may be involved.

Firmware monitoring is the exploitation of a microprogrammed computer's microprogram control logic as a measurement tool. It is possible only on those systems which employ microprogram control logic. It has the advantages of access to the computer's control registers and reduced execution time overhead.

Self-Simulators are software packages which interpretively execute program code while gathering data about program characteristics, such as branch analysis or instruction use statistics. They incur a great deal of overhead in execution time. They are not a suitable tool for a real-time environment, but are extremely useful as a developmental or non-real time analysis tool.

Benchmarks and Synthetic programs are programs designed to represent the data processing characteristics and workload of a particular application or installation job mix. They are used primarily to compare the suitability of various computers for the job environment which they represent.

Any combination of these measurement tools (except Firmware monitoring) can be applied to the tasks of measuring, evaluating and studying the ARTS III systems. The choice depends on the specific measurement objectives and the environment in which the measurement activity will take place. With any measurement

activity there is an accompanying and equally important data reduction effort to extract, combine, and organize the measurement data in a digestible and informative fashion. The use of computer generated graphs and charts is invaluable for aiding the analyst in grasping the significance of results. With Benchmarks and Synthetic Programs there is also a large effort involved in realistic input data preparation.

7.0 CONCLUSIONS & RECOMMENDATIONS

Measurement aids to program development and debugging are currently being developed by UNIVAC under the heading "General Purpose Data Recording and Timing for ARTS"; and an effort is underway at TSC to evaluate various computer architectures for ATC applications through the use of Synthetic Programs collectively known as the "ATC Template".

There are at least two other specific areas where direct computer measurement should be utilized within the overall ARTS III program: systems analysis of operational site performance and environmental characteristics; and continuous test-bed monitoring which includes the effects of hardware reconfiguration, the impact of new Executive and functional task software, as well as performance efficiency.

Simulations have been developed to aid in planning the hardware configurations, memory mapping and lattice table organizations for future ARTS III multiprocessor systems. Accurate and comprehensive data relative to the operational and environmental characteristics of the various sites which are candidates for multiprocessor operation is inportant in making such simulation models truly effective. This information should be obtained directly from the existing basic ARTS III sites, since much of the data would be concerned with ambient air traffic characteristics and other local considerations. Additionally, those sites should be subjected to measurement after multiprocessor installation to verify anticipated performance.

A certain amount of information can be obtained through the use of existing measurement aids in the form of a Data Extractor program and a task Timing program. These aids are rather primitive, however. The data extractor dumps raw buffers, and it is not possible to run it concurrent with the timing program. It is, therefore, not easy to obtain task timing as a function of air traffic load; although useful operational data for system operational analysis is available. These programs can be improved on,

or built upon to extract data selectively and more germain to performance analysis.

Measuring an operational system, however, gives rise to several problems. Firstly, it is difficult if not impossible to obtain data relative to certain hardware utilization in the ARTS systems via software means, particularly without extensive overhead. Secondly, the sites of immediate interest would most likely be the busiest ones with the least amount of processor time to devote to measurement overhead. Any software measurement package will incur an overhead which could become detrimental to the real-time integrity of the system during periods of increased air traffic or additional function processing. Thirdly, only one magnetic tape (used to load the system) is available at operational sites. Even if this tape were used to output measurement data, some data would be lost during tape changes.

There are several approaches which may be taken to alleviate these problems.

1. Hardware measurements, along with a subset of software measurements relative to program execution frequencies and elapsed times, can be extracted via hardware monitor. However, data on traffic volume, distribution of targets across sectors, correlation statistics, etc, would not be available unless they were provided via software monitor augmentation.

2. Sophisticated software monitors yielding some hardware measures could be developed and run only during low and medium load periods with data for high load periods extrapolated.

3. Software monitor overhead could be reduced through the use of sampling techniques rather than continuous monitoring of events.

4. Software overhead could be reduced through the use of counts and gross timing measures rather than fine grain timing of operational events.

5. Provision could be made to include time for continuous software monitoring overhead in plans for future processing capability requirements for operational sites.

6. A hybrid monitoring scheme could be developed for use at operational sites.

If "level 1 redundancy" plans were implemented, the redundant processor would be used only a small percentage of the time recording critical backup data and performing error monitoring. It would be possible, therefore, to run a software monitor on the second processor which would have access to all information in main memory and which would not interfer with normal processing on the primary processor. This arrangement would be Hybrid-like in that the second processor is effectively passively monitoring the activity of the main operational processor.

Of the alternatives listed above, Item 5 has been somewhat addressed by the continuous Data Recording and Analysis System under development by UNIVAC. This facility will extract certain operational data on multiprocessing systems and it includes the necessary additional recording hardware (disc). Computer performance is not included in the current design.

The evolution of the Failsafe/soft multiprocessing phase of the ARTS III program bears close watching. This new sophistication brings with it a new difficulty in sizing, assessing, and understanding the effects of changes in the system. The Executive logic is considerably different and more complex than that of Basic ARTS III Systems. Potential exists for performance degradation due to less than optimum implementation, processor and parallel task interference, bottlenecks, queue delays, and inefficiencies involved with the use of on-line direct access devices for recording critical backup data. In addition new functional tasks will be periodically added to the system such as Radar tracking, Conflict Prediction, and Metering and Spacing, each will alter the system performance characteristics and saturation thresholds. The addition of any new hardware (CPM, Parallel Processor, Data Link Minicomputer, or modification of existing

hardware will also have an impact which should be measured and understood.

In order to keep abreast of the multiprocessor expansion, a comprehensive measurement facility should be incorporated into the ARTS III test bed facility for the primary purpose of continuous total system performance evaluation. The test bed is the ideal place for such a facility since it can tolerate the measurement overhead without jeopardizing actual terminal control. New software and hardware features can be properly integrated, their impact assessed, and the system tuned for performance prior to operational use. Once operational data has been gathered from the operational sites, that data can be used to simulate or emulate those operational sites in the test bed. Each site could therefore, be equipped with an operational system tailored for performance as well as for configuration and local parameters.

The measurement facility should be capable of obtaining hardware oriented measurements such as channel and memory activity, channel and memory conflict queue times, percentage busy statistics for each processor, processor overlap percentages, disk access statistics, etc. Software related measurements should be possible at the microscopic level to indicate absolute elapsed time of tasks and subroutines, as well as table access and scheduling queue effects. Operational data such as track load, track sector distributions, display statistics, etc. should also be obtainable.

Ideally, this would be accomplished with a mini-computer/hardware monitor hybrid system with tape and printer peripherals; where the mini-computer has access to main memory and interfaces with the hardware monitor and with a software monitor running in the host ARTS system. A more realistic approach, however, would be the development of a sophisticated software event monitor with special data gathering options, augmented by an independent hardware monitor with its own magnetic tape recording facility.

Considerations relative to existing test bed hardware and software would include the necessity for a program readable clock

driven by a timing source on the order of ten microseconds,* availability of tape devices for software measurement output, and modifications to existing operational software to integrate the software monitor and software probe points.

*A 60 microsecond timing source has been installed on the MSP multiprocessor configuration and presumably will be transferred to NAFEC along with the rest of the hardware. This would be an improvement over the one millisecond clock currently available and may be adequate for much measurement.

APPENDIX A
A SIMULATION MODEL OF THE ENHANCED ARTS EXECUTIVE SCHEDULER

It is necessary to have a clear understanding of the functions of the ARTS Executive in order to develop an effective measurement strategy. One technique for gaining insight into a system while selecting critical system features for measurement is computer simulation. For this reason, a simulation model of the Enhanced ARTS Executive Scheduler operating in the Data Processing Subsystem was developed.

A-1. SCOPE OF THE MODEL

The simulation model represents the logic of the Enhanced ARTS Scheduler Module operating in a multi-processing environment. The ARTS Scheduler is part of the table-driven ARTS Executive. Task programs operating in this environment are either planned or popup tasks. Planned tasks are grouped into heirarchical sets described in the Lattice Description Tables (LDT). When a planned task is ready for execution it is moved from its LDT to the Planned Task Pointer (PTP) Table.

The execution order of the lattices is stored in the Cycle Table (CT). The time-to-execute for a cycle is set, (i.e., 125 ms.) and there is one lattice pointer entry in the CT for each cycle (See Figure A-1).

Scheduling of popup tasks is requested by planned tasks through an Executive Service Request (ESR). The scheduled pop-up task is placed in the popup list. Pop-up tasks are indirectly given higher priority than planned tasks since the pop-up queue is searched before the PTP.

The model has the capability to handle planned tasks organized in lattice structures, popup tasks, and interrupts. The resources of the simulated system are one to four processors and two tables; the Lattice Description Table (LDT) and the Planned Task Pointer (PTP) table. Execution time was the only task program characteristic which was represented in the simulation.

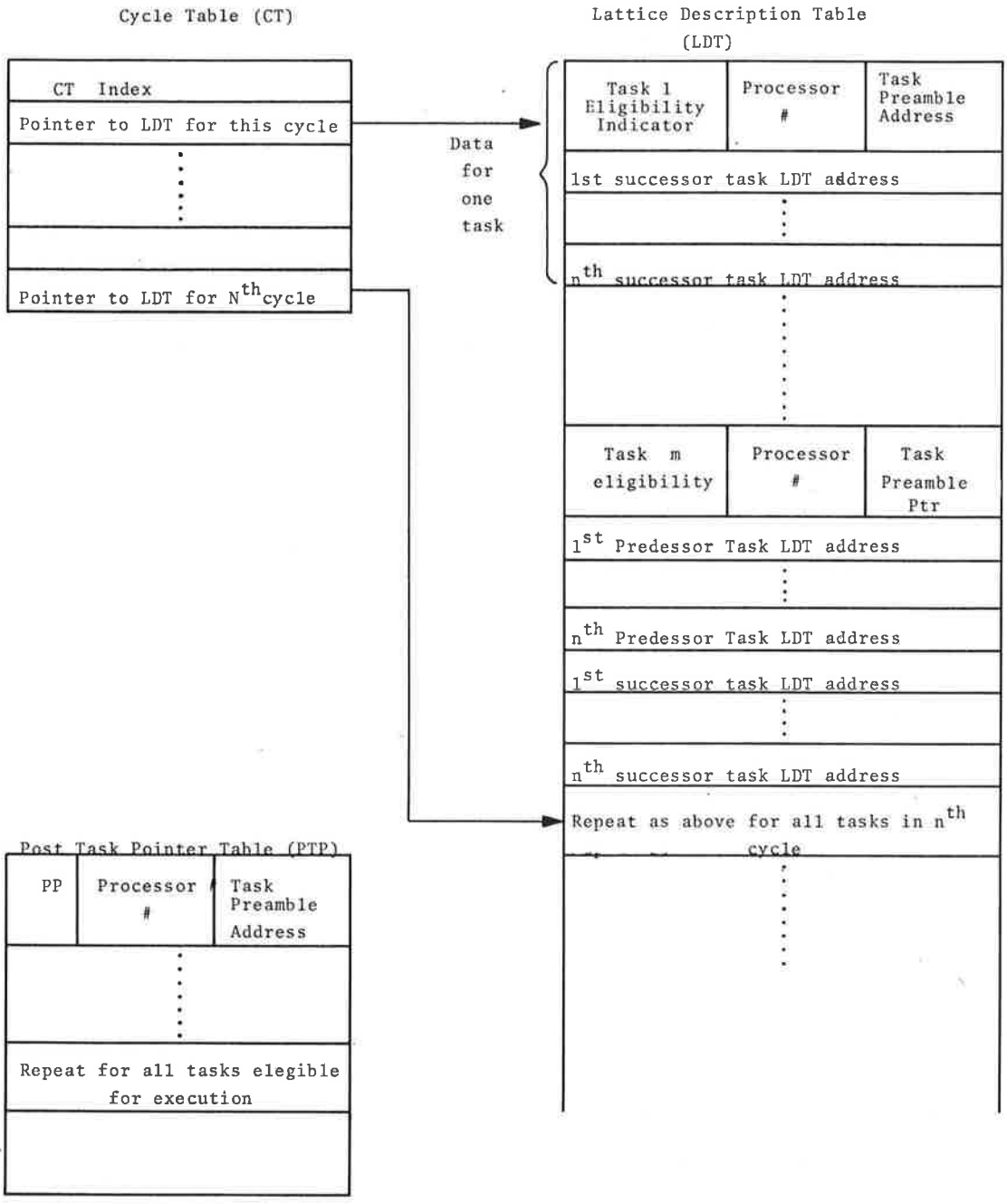


Figure A-1. ARTS Scheduler Tables

multiprogram design data available at the time the simulation was written, (2) the memory capacity of the Honeywell 832 (the machine used to run the simulation), (3) the lack of a high level simulation language.

The following "events" which occur in the operation of the ARTS Scheduler Module were modelled in the GASP framework.

1. Task completion
2. LDT pointer updating
3. PTP utilization
4. Cycle completion
5. Interrupt handling

Functions for calculation of task and interrupt execution times and inter-interrupt intervals are also included. A flowchart of the GASP model logic appears in Figure A-2.

A-3. MEASUREMENTS OF THE SIMULATED SYSTEM

The simulation program includes the capability for gathering statistics on the performance of the simulated system. The following summary statistics are provided:

Average and Max. number of processors waiting for LDT

Average and Max. number of processors waiting for PTP

Average and Max. number of interrupts waiting for processing

Average and Max. number tasks in PTP

Total time spent waiting for LDT and PTP for each processor

% busy time for each processor (non-idle time, includes table queue time)

% busy time for each table

Distribution of processor PTP wait times (histogram)

Distribution of processor LDT wait times (histogram)

For each task with a variable execution time: mean, std. dev., min., and max. and number of executions.

A system measurement package might gather these types of statistics. By analyzing the data gathered by the simulated system the critical features to be monitored by the measurement package can be identified, thus avoiding unnecessary data collection by a measurement tool. A sample simulation output is included.

A-4. DISCUSSION OF RESULTS

The ARTS simulation model was run using the eight different lattices (Figure A-3) provided by UNIVAC documentation and the timing data in Table A-1. Only planned tasks were included in the runs because no data was available for interrupt occurrences. Also, no popups were included because the UNIVAC data provided one popup task which executes every 10 to 12 seconds and the model was run to simulate 5 seconds (5000 milliseconds) of ARTS activity.

The model was run simulating a two processor system and then a three processor system. To assess the possibility of increasing the number of displays in the system, the time for task CPSE was changed to reflect the effect of having 5 displays rather than just one.

The results of four runs of the model are summarized below. Each run represents 10 cycles of activity

Two Processor System:

Resource	1 display % busy time	5 displays % busy time
Processor 1	9.72	9.72
Processor 2	34.34	39.97
PTP	11.0	11.0
LDT	8.9	8.9

Elasped time 5000 ms.

Three Processor System:

Processor 1	9.31	9.31
Processor 2	34.10	39.73
Processor 3	5.17	5.17

Lattice 6

same as 0.

Start time = 375.

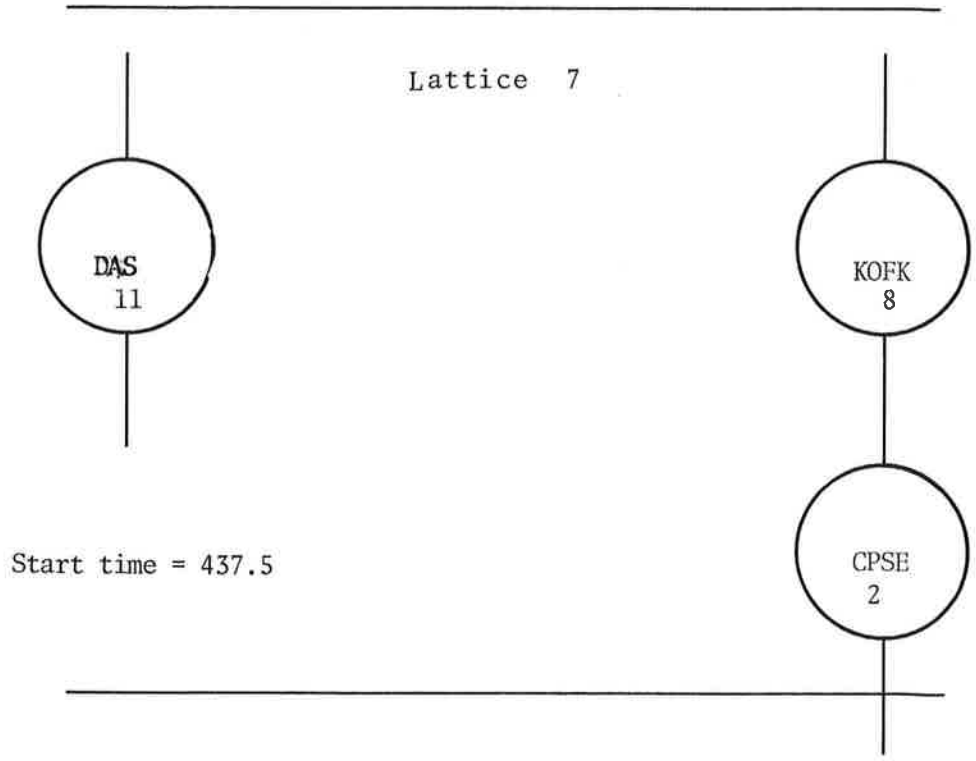
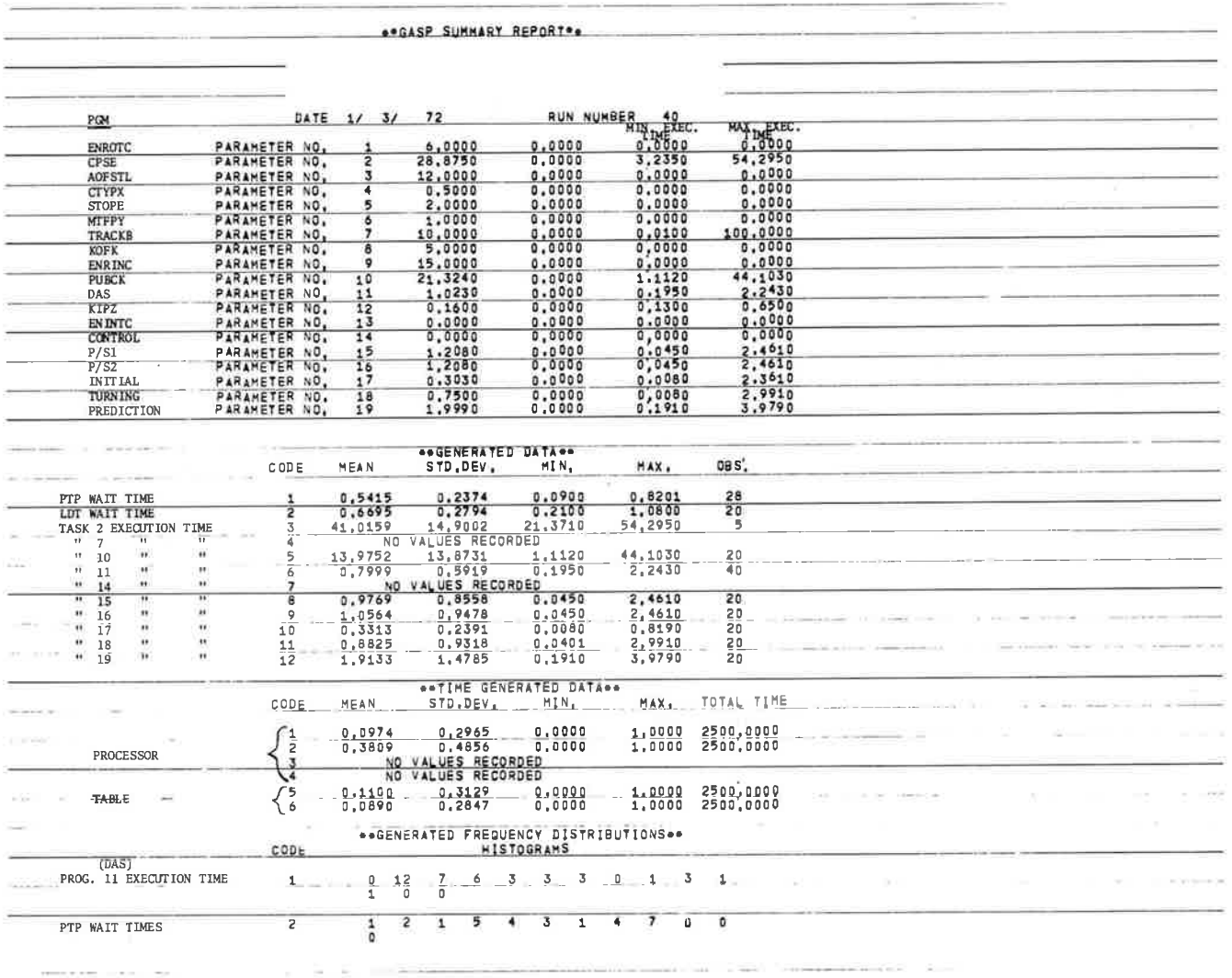


Figure A-3. Sample Lattices cont'd

(times are in ms.)

TABLE A-1 SUBPROGRAM TIMING DATA

Task name	Mean exec. time	Min. exec. time	Max. exec. time
ENROTC	6.		
CPSE	28.875	3.235	54.295
AOFSTL	12.0		
CTYPX	.5		
STOPE	2.0		
MTFPY	1.0		
TRACKB	10.0	.1	100.
KOFK	5.		
ENRINC	15.		
PUBCK	21.324	1.112	44.103
DAS	1.023	.195	2.243
KIPZ	.16	.13	.65
P/S1	1.2080	.045	2.461
P/S2	1.2080	.045	2.461
INITIAL	.303	.008	2.361
TURNING	.75	.008	2.991
PREDICTION	1.999	.191	3.9790



GASP SUMMARY REPORT

PGM — DATE 1/ 3/ 72 — RUN NUMBER 40

				MIN. EXEC. TIME	MAX. EXEC. TIME
ENROTC	PARAMETER NO.	1	6.0000	0.0000	0.0000
CPSE	PARAMETER NO.	2	28.8750	0.0000	3.2350	54.2950
AOFSTL	PARAMETER NO.	3	12.0000	0.0000	0.0000	0.0000
CTYPX	PARAMETER NO.	4	0.5000	0.0000	0.0000	0.0000
STOPE	PARAMETER NO.	5	2.0000	0.0000	0.0000	0.0000
MTFPY	PARAMETER NO.	6	1.0000	0.0000	0.0000	0.0000
TRACKB	PARAMETER NO.	7	10.0000	0.0000	0.0100	100.0000
KOFK	PARAMETER NO.	8	5.0000	0.0000	0.0000	0.0000
ENRINC	PARAMETER NO.	9	15.0000	0.0000	0.0000	0.0000
PUBCK	PARAMETER NO.	10	21.3240	0.0000	1.1120	44.1030
DAS	PARAMETER NO.	11	1.0230	0.0000	0.1950	2.2430
KIPZ	PARAMETER NO.	12	0.1600	0.0000	0.1300	0.6500
ENINTC	PARAMETER NO.	13	0.0000	0.0000	0.0000	0.0000
CONTROL	PARAMETER NO.	14	0.0000	0.0000	0.0000	0.0000
P/S1	PARAMETER NO.	15	1.2080	0.0000	0.0450	2.4610
P/S2	PARAMETER NO.	16	1.2080	0.0000	0.0450	2.4610
INITIAL	PARAMETER NO.	17	0.3030	0.0000	0.0080	2.3610
TURNING	PARAMETER NO.	18	0.7500	0.0000	0.0080	2.9910
PREDICTION	PARAMETER NO.	19	1.9990	0.0000	0.1910	3.9790

GENERATED DATA

	CODE	MEAN	STD.DEV.	MIN.	MAX.	OBS.
PTP WAIT TIME	1	0.5415	0.2374	0.0900	0.8201	28
LDI WAIT TIME	2	0.6695	0.2794	0.2100	1.0800	20
TASK 2 EXECUTION TIME	3	41.0159	14.9002	21.3710	54.2950	5
" 7 " "	4	NO VALUES RECORDED				
" 10 " "	5	13.9752	13.8731	1.1120	44.1030	20
" 11 " "	6	0.7999	0.5919	0.1950	2.2430	40
" 14 " "	7	NO VALUES RECORDED				
" 15 " "	8	0.9769	0.8558	0.0450	2.4610	20
" 16 " "	9	1.0564	0.9478	0.0450	2.4610	20
" 17 " "	10	0.3313	0.2391	0.0080	0.8190	20
" 18 " "	11	0.8825	0.9318	0.0401	2.9910	20
" 19 " "	12	1.9133	1.4785	0.1910	3.9790	20

TIME GENERATED DATA

	CODE	MEAN	STD.DEV.	MIN.	MAX.	TOTAL TIME
PROCESSOR	1	0.0974	0.2965	0.0000	1.0000	2500.0000
	2	0.3809	0.4856	0.0000	1.0000	2500.0000
	3	NO VALUES RECORDED				
	4	NO VALUES RECORDED				
TABLE	5	0.1100	0.3129	0.0000	1.0000	2500.0000
	6	0.0890	0.2847	0.0000	1.0000	2500.0000

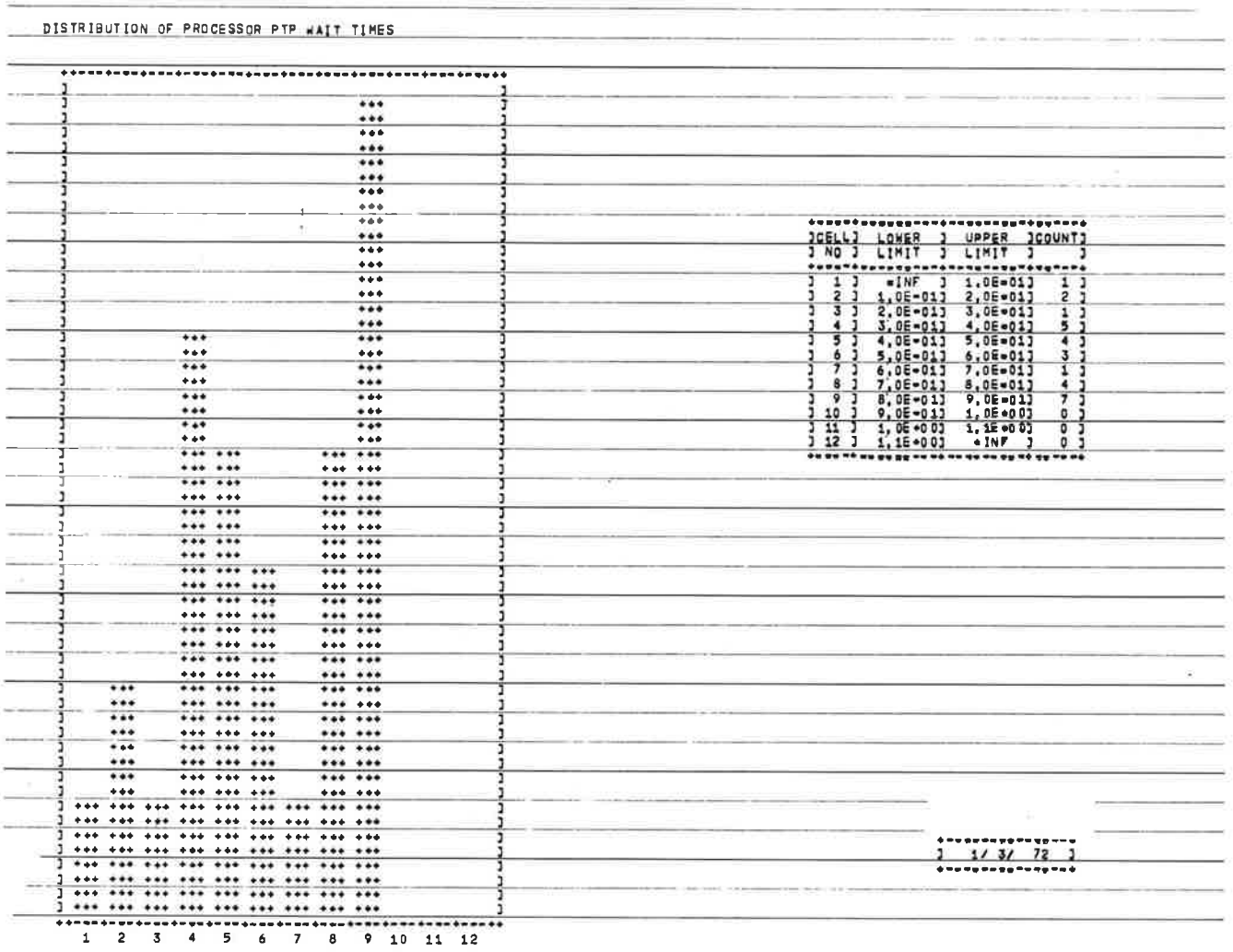
GENERATED FREQUENCY DISTRIBUTIONS

	CODE	HISTOGRAMS
(DAS) PROG. 11 EXECUTION TIME	1	0 12 7 6 3 3 3 0 1 3 1 1 0 0
PTP WAIT TIMES	2	1 2 1 5 4 3 1 4 7 0 0 0

Figure A-4. Computer Output - GASP Simulation of ARTS III Multiprocessor Scheduler (Annotated)

```
LDT WAIT TIMES          3        0  0  1  3  2  0  2  1  5  2  2
                                 2

                    FILE PRINTOUT, FILE NO.  1          EVENT FILE

                AVERAGE NUMBER IN FILE WAS     1.4671
                STD. DEV.                      0.6126
                MAXIMUM                             3

                    THE FILE IS EMPTY

                    FILE PRINTOUT, FILE NO.  2          PTP QUEUE

                AVERAGE NUMBER IN FILE WAS     0.0060
                STD. DEV.                      0.0771
                MAXIMUM                             1

                    THE FILE IS EMPTY

                    FILE PRINTOUT, FILE NO.  3          PTP

                AVERAGE NUMBER IN FILE WAS     0.0413
                STD. DEV.                      0.2201
                MAXIMUM                             3

                    THE FILE IS EMPTY

                    FILE PRINTOUT, FILE NO.  4          LDT

                AVERAGE NUMBER IN FILE WAS     0.5048
                STD. DEV.                      1.1621
                MAXIMUM                             8

                    THE FILE IS EMPTY

                    FILE PRINTOUT, FILE NO.  5          LDT QUEUE

                AVERAGE NUMBER IN FILE WAS     0.0053
                STD. DEV.                      0.0728
                MAXIMUM                             1

                    THE FILE IS EMPTY

                    FILE PRINTOUT, FILE NO.  6          INTERRUPT QUEUE

                AVERAGE NUMBER IN FILE WAS     0.0000
                STD. DEV.                      0.0000
                MAXIMUM                             0

                    THE FILE IS EMPTY
```

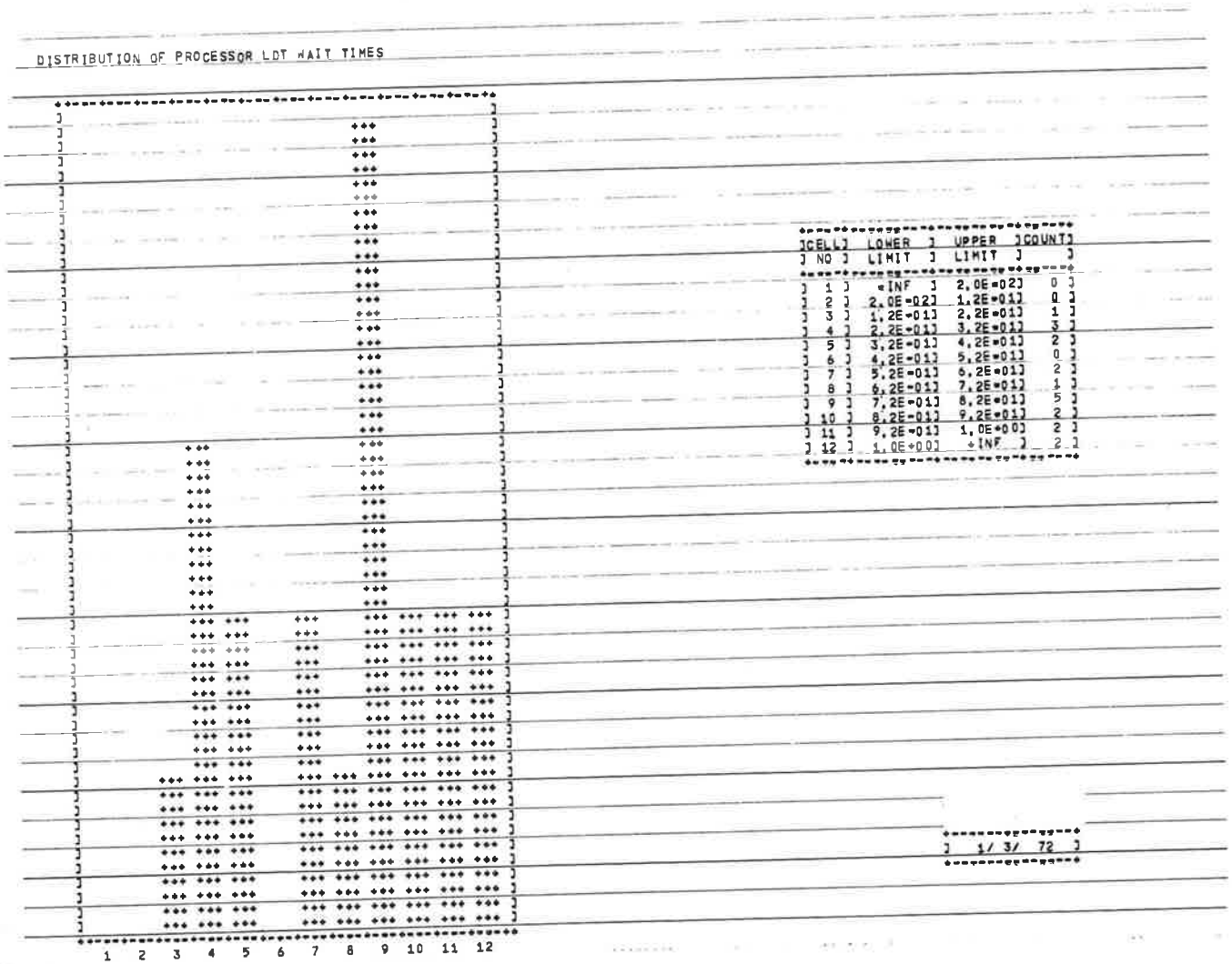
Figure A-4. Computer Output - GASP Simulation of ARTS III Multiprocessor Scheduler (Annotated). (Cont'd)



CELL NO	LOWER LIMIT	UPPER LIMIT	COUNT
1	-INF	1.0E-01	1
2	1.0E-01	2.0E-01	2
3	2.0E-01	3.0E-01	1
4	3.0E-01	4.0E-01	5
5	4.0E-01	5.0E-01	4
6	5.0E-01	6.0E-01	3
7	6.0E-01	7.0E-01	1
8	7.0E-01	8.0E-01	4
9	8.0E-01	9.0E-01	7
10	9.0E-01	1.0E+00	0
11	1.0E+00	1.1E+00	0
12	1.1E+00	+INF	0

1/ 3/ 72

Figure A-4. Computer Output - GASP Simulation of ARTS III Multiprocessor Scheduler (Annotated). (Cont'd)



CELL NO	LOWER LIMIT	UPPER LIMIT	COUNT
1	-INF	2.0E-02	0
2	2.0E-02	1.2E-01	0
3	1.2E-01	2.2E-01	1
4	2.2E-01	3.2E-01	3
5	3.2E-01	4.2E-01	2
6	4.2E-01	5.2E-01	0
7	5.2E-01	6.2E-01	2
8	6.2E-01	7.2E-01	1
9	7.2E-01	8.2E-01	5
10	8.2E-01	9.2E-01	2
11	9.2E-01	1.0E+00	2
12	1.0E+00	+INF	2

1/ 3/ 72

Figure A-4. Computer Output - GASP Simulation of ARTS III Multiprocessor Scheduler (Annotated). (Cont'd)

11. Stanley, W. I., "Measurement of System Operational Statistics", IBM Systems Journal, Vol. 8, No. 4, 1969.

12. Schwetmann, N.D. Jr., "A Study of Resource Utilization and Performance Evaluation of Large-Scale Computer Systems", Ph.D. dissertation, The University of Texas at Austin, July 1970, TSN-12.

13. UNIVAC Defense Systems Div., "Level 1 Redundancy Switching Dual Beacon Design Data", Final Rpt., Project No. 19180, May 1971.

14. UNIVAC Defense Systems Div., "Level 1 Redundancy Switching Single Beacon Design Data", Interim Rpt., Project. No. 19180, October 1970.

15. UNIVAC Defense Systems Division, "The Initial Multiprocessor Executive Concepts Design Study", Final Rpt., Project No. 19180, October 1970.

16. UNIVAC Defense Systems Div., "Multiprocessor Failsoft Executive Program Design Data", January 1972.

17. Browne, J. C., "Functional Concepts and Basic Data of Performance Measurement and Evaluation", First Texas Symposium on Computer Systems, June 29-30, 1972.

18. Drummond, Jr., M. E., "A Perspective on System Performance Evaluation", IBM Systems Journal, Volume 8, Number 4, 1969.

19. Saltzer and Gintell, "The Instrumentation of Multics", Comm. of the ACM, Vol. 13, No. 8, August 1970.

20. Meuller, J. H., "Aspects of the Gemini Real-Time Operating System", IBM Systems Journal, Vol. 6, No. 3, 1967.

21. Stanley and Hertel, "Statistics Gathering and Simulation for the Apollo Real-Time Operating System", IBM Systems Journal, Vol. 7, No. 2, 1968.

22. Bussell and Koster, "Instrumenting Computer Systems and Their Programs", Fall Joint Computer Conference, 1970.

23. Schwetmann, and Brown, "An Experimental Study of Computer System Performance", Proc. of the ACM, August 1972,

Vol. II, pp 693-703.

24. Holtwick, Gary M., "Designing a Commerical Performance Measurement System", ACM SIGOPS Workshop on System Performance Evaluation, 5-7, April 1971.

25. Kolence, "A Software View of Measurement Tools", DATAMATION January 1971, pp 32-38.

26. Lucas, Jr., Henry C., "Performance Evaluation and Monitoring", Computing Surveys, Vol. 3, No. 3, September 1971.

27. Johnson, R. R., "Needed a Measure for Measure", DATAMATION, December 15, 1970.

28. Buchholz, W., "A Synthetic Job for Measuring System Performance", IBM Systems Journal, Vol. 8, No. 4, 1969.

29. Esterin G., et al, "SNUPER COMPUTER - A Computer in Instrumentation Automation", Proc. Spring Joint Computer Conference, 1967.

30. Aschenbrenner, Richard A., et al, "The Neurotron Monitor System", Proc. Fall Joint Computer Conference, 1971.

31. Reigel, E. W., et al, "The Interpreter - A Microprogrammable Building Block System", Proc. Spring Joint Computer Conference, 1972.

32. UNIVAC Defense Systems Div., "The Initial Multiprocessor Executive Concepts Design Study", October 1970, Final Report, Project No. 19180.

33. UNIVAC Defense Systems Div., Corson, Miller, "Program Test Specification Plan for Multi-IOP Executive Program for use with Module-Processor, Input-Output Computer", Contract No. FA70WA-2289, 19 August 1971.