UNIVERSITY
OF WOLLONGONG
AUSTRALIA

University of Wollongong
Research Online

Department of Computing Science Working Paper Series

Faculty of Engineering and Information Sciences

1981

A hybrid real time performance analyser

P. J. McKerrow
University of Wollongong, phillip@uow.edu.au

Recommended Citation

McKerrow, P. J., A hybrid real time performance analyser, Department of Computing Science, University of Wollongong, Working Paper 81-5, 1981, 24p.
http://ro.uow.edu.au/compsciwp/16

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

A HYBRID REAL-TIME PERFORMANCE ANALYSER

Phillip John McKerrow

Department of Computing Science,
The University of Wollongong,
Post Office Box 1144,
Wollongong, N.S.W 2500
Australia.

ABSTRACT

Hybrid monitors attempt to take advantage of the complementary nature of hardware and software tools. Research into ways of dividing the monitoring functions between the hardware and software sections of a tool and co-ordinating their interaction has guided the design of an easy-to-use hybrid performance analyser. The proposed tool can be added to an existing system or implemented as part of an integrated design of a new system.

The real power of the tool is in its ability to reduce and analyse the data being read. A hierarchy of performance measurement strategies is discussed. Execution time, execution path, execution frequency and stimulus information can all be obtained without having to refer to the code of the modules, considerably simplifying the measurement process.

Real-time analysis provides the information needed for model development and verification, module spectral analysis, bottle-neck analysis and adaptive system control.

Keywords: Hybrid Performance Analyser, Performance Evaluation, Hardware Monitoring, Software Monitoring, System Design, Operating System Design, Adaptive Control, Data Collection, Data Analysis, Module, Module Number, Task, Event, Execution Time, Execution Path, Logic State Analyser, Performance Evaluation Models, Cache Memory, Spectral Analysis, Real-time Analysis, Bottle-neck Analysis.

1. INTRODUCTION

Performance evaluation can be split into three broad areas: measurement of system parameters, evaluation of the data and modifications to improve performance. The goal of the evaluator is to improve system performance by either identifying and correcting bottle-necks or by selecting optimum system constants. Some workers [4, 7, 30] have suggested that the ultimate goal of performance evaluation research is to be able to measure and adaptively control performance indices in real-time in order to maintain optimum performance under varying system load conditions.

The state of the art of control theory, and technology, is such that practicing control engineers generally feel that if a parameter, in an industrial plant, can be measured or accurately modelled, then it can be controlled. Computing power is no longer a limitation in developing highly sophisticated control algorithms and systems. In order to develop the necessary control algorithms, extensive measurement, modelling and evaluation must be done.

When evaluating and controlling performance the analyst first has to decide what to measure and then how to measure it. A high performance monitoring tool designed to provide accurate real-time information should:

(i) be simple to use, with an ergonomic human interface,

(ii) be flexible enough to provide both microscopic (instruction level) and macroscopic (process level) measurements,

(iii) provide data reduction, analysis and display in real-time,

(iv) not require an intimate knowledge of the system before any useful results can be obtained,

(v) eliminate the need to spend hours digging through code,

(vi) cause minimal performance degradation due to interference,

(vii) not be expensive.

The design of such a tool requires an integrated approach to system design. If, at system design time, the indices to be measured and the method of measurement can be specified then the necessary hardware and software can be included in the computer and operating system. This would provide considerable improvement over the current method of adding tools to an existing system, often in an ad-hoc way.

2. HUMAN ENGINEERING

Increasingly computer users are faced with black boxes built without regard for the user's need for information about the system operation. In many ways, computers complement human intelligence but if computers fail to provide information that is easy for the user to assimilate, confidence in the system is diminished.

Often one types in a copy command on a dual floppy-disc system and wonders if the copy is going the correct way. Lights indicating reading and writing operations would give the user increased confidence and more rapid feedback. Computers are excellent at handling numbers and tables; humans at recognizing patterns and pictures. The removal of speakers and front panels from computers have made them less friendly.

The operating system used in a real-time control project [2] was modified to illuminate an individual light on the console for each process, when that process was active. As many of the processes were cyclic, due to the nature of the external machines, a regular pattern could be observed on the lights when the system was operating correctly. This enabled the operation of a complex control system to be analysed visually. On a number of occasions, control system problems were diagnosed, by an engineer, using information about the light pattern relayed over the phone by electricians who subsequently fixed the faults.

Often by concentrating on the technical aspects of a problem, we overlook human engineering and consequently miss an elegantly simple solution. Looking at the performance measurement problem from a human engineering and control system point of view led to the proposal discussed in this paper.

3. MEASUREMENT TECHNIQUES AND PROBLEMS

The main sources of measurement data are existing accounting software, hardware monitors attached to the system and software monitors. Normally, accounting data does not contain sufficient information and a hardware of software monitor is required.

Conceptually, a measuring instrument consists of four parts (figure 1). The sensor's task is to sense the magnitude of the quantity being measured and thus it includes the software or hardware probes. In the transformer section the measured data is manipulated and reduced to the desired form, e.g. a state analyser's ability to distinguish desired states from a continuous data stream. After data reduction the measured information can be displayed directly or it can be analysed and the results of the analysis displayed.

3.1. Hardware Monitors

Hardware tools consist of additional hardware attached to the computer's backplane via test probes. Attaching the probes requires a detailed knowledge of the backplane and a clear specification of which signals are to be monitored. Consequently they tend to make maintenance people nervous. They can detect events occurring at microscopic levels with high accuracy, e.g. individual instruction fetches. Thus, potentially their scope is broad since most of the interesting points in the system can usually be reached. The effect of the increasing scale of chip integration can be offset if probe points are included at design time.

Hardware monitors are often used to check the accuracy of software monitors. Their main advantage over software monitors is they do not interfere with the system being measured. They are inherently limited to measuring information that can be interpreted at the hardware level without knowledge of operating system activities. Consequently, information like cpu service time and file activity by process cannot easily be obtained. Distribution estimates, e.g. mean channel service time, can be obtained for the system as a whole but not for specific work load classes. For these reasons hardware monitors are often supplemented by software monitors or accounting data.

Logic state analysers used as hardware monitors [1] provide very accurate microscopic measurement, e.g. instruction execution path and time. They can also be used for certain types of macroscopic measurement, e.g. process execution time or path, but again they lack operating system specific information. It is difficult to analyse some processes (e.g. the scheduler) without intimate knowledge of the current system load.

3.2. Software Monitors

In event-driven software monitors significant events are defined (e.g. process switch) and the operating system is modified to record information about the event. Small measurement sessions can be recorded in memory but most require disc file or tape storage. Monitoring detail is limited by the number of events recorded. Consequently, if after data analysis, it is found that a significant event has not been recording then at best the session has to be rerun, and at worst the operating system has to be modified to record the event.

An event-driven monitor has greater flexibility than a sample driven monitor but is likely to interfere with the system more. Depending upon the particular system and monitor, a software monitor may consume 20% or more of system resources (particularly cpu and channel time) and thus produce very questionable results. If this overhead can be kept to 5%, by appropriate event definition and probe implementation, software

monitor results are reasonably accurate [3].

In addition to interfering with system performance, software monitors have two other significant problems. First, the amount of data produced by an event trace may require excessive post-data reduction before meaningful results can be obtained. Secondly, software monitors must be specifically designed for the operating system and machine architecture. If not designed into the system they can be difficult to add.

However, software monitors can get at operating-system-specific information (e.g. system queues) which hardware monitors can not. Also they can readily associate physical activity with logical entities (e.g. disc access with file name). In many ways, software and hardware tools compliment one another. Hybrid tools have been developed to try to utilize the advantages of both.

3.3. Hybrid Monitors

Hybrid tools require the addition of both extra hardware and software to the system to be measured. They consist of external hardware tools which receive data collected by a software or firmware tool running in the system being measured [14, 27]. The hardware monitoring device is no longer invisible to the operating system, but it is treated as an "intelligent peripheral device" which can be used by a software monitor [16, 23]. Some even allow sections of the hardware tool, e.g. event counters, to be allocated to users under the control of software. A device of this type has been used to derive an on-line histogram of subroutine utilization and other similar tasks [17]. Another type of hybrid monitor uses part of the existing hardware (a channel processor) as the monitor [18].

Burroughs have included a monitor micro instruction in the firmware of some of their computers [13, 19]. This instruction writes a bit pattern specified by the programmer to pins accessible to the probes of a hardware tool. Nutt [16] indicates that the system/monitor interface technique used by Hughes and Cronshaw [21] and by Ruud [22] best illustrates the trend toward hybrid monitoring.

A hybrid monitor (figure 2) consists of a hardware monitor plus a data channel between the monitor and the computer being evaluated, which can be used for transferring information between the two. While using the hardware probes to monitor an event the data channel can be used to obtain information about the stimulus of the event, thus overcoming the inherent limitation of hardware monitors. Alternatively, an event driven software monitor can interrupt the external monitor which picks up the required event data from memory, via the data channel thus reducing the overhead of the software monitor. In addition, software can request a measurement to be made by the hardware. The addition of

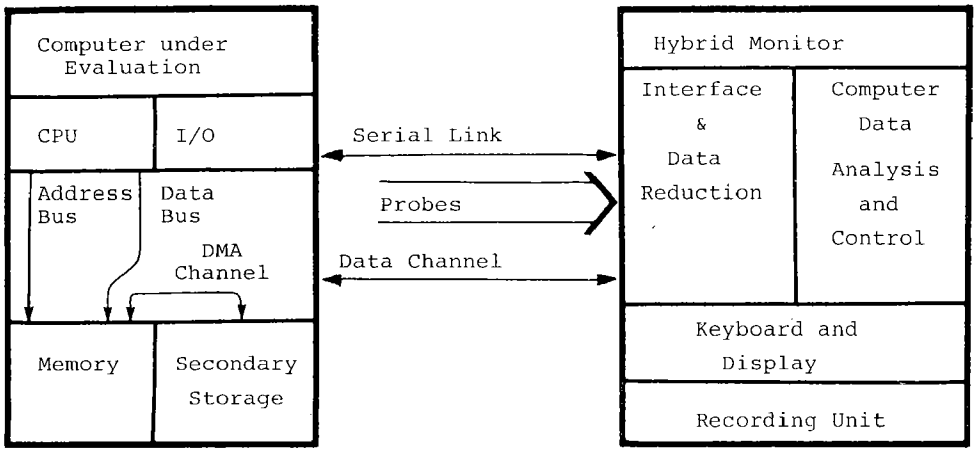

Figure 2. Basic Components of a Hybrid Monitor.

computing power into the monitor allows considerable real-time data reduction and analysis to be carried out and increases the flexibility of the monitor.

Thus hybrid monitors are an attempt to take advantage of the complementary nature of hardware and software tools. Ferrari [4] comments that the simultaneous usage of co-operating tools of the same or different nature, their coordination, and the partitioning of their functions and jurisdictions create problems which are still open to research. The proposal developed in this paper is an attempt to solve some of these problems.

4. COMPUTER AND O/S DESIGN FOR MONITORING

The majority of monitoring tools are designed to monitor existing systems. Consequently, serious problems, due to the constraints imposed by the organization of the system being monitored, are often encountered. These problems include:

(i) Events of interest are not accessible.

(ii) Excessive interference.

(iii) Difficulty in verifying collected data.

(iv) Modifications to the system to provide the required data may be difficult, risky or expensive.

These problems could be minimised if a comprehensive set of measurement facilities were included during system design. This practice has not been popular so far among computer manufacturers even though such tools would then be available for use during system implementation and debugging. Research projects into this have included the instrumentation of Multics [6] and of PRIME [5]. One of the main problems faced by the designers was to predict what events would be of interest for measurement purposes once the system was implemented. As a result, both projects adopted a mixture of ad-hoc fixed tools and general purpose tools.

To be effective in the long term, performance evaluation has to be considered when the system, both hardware and software, is designed. We should no longer design hardware, software and instrumentation separately and then patch until they work. An integrated approach is necessary. Before such an integrated approach is achieved, performance evaluators will have to be able to specify more clearly, at design time, what they want to measure and why. Current research into modelling methods is providing much of this information. When performance evaluation theory has developed to the stage where what is to be measured and why it is to be measured and evaluated, can be specified at system design time, then the manufacturers are more likely to include instrumentation in their systems.

5. HYBRID ANALYSER DESIGN - DATA COLLECTION

Before commencing to evaluate the performance of a specific computer system, the evaluator must first define what is meant by "performance" of the system. This includes establishing what the user, or whoever requested the evaluation, wants out of the system and what the system was designed to do in the first place. Performance indices can then be defined for the system and the measurements needed to give values to these indices specified.

Thus performance requirements and performance indices will vary from system to system and from user to user within the same system, e.g. the systems programmer and the payroll clerk have completely different requirements. However, even though the actual variables measured depend upon the context, the type of measurement is similar.

Measurements of interest to all include:

(i) Execution time.

(ii) Execution path.

(iii) Frequency of execution.

(iv) Bottleneck identification within a task.

(v) Stimulus information related to the above.

The above hold true whether context is defined as a user program, a segment of an operating system, a whole operating system, a disc server or a model of the system. A proposed hybrid real-time performance analyser designed to make these measurements is discussed in the following sections. Implementation effort is currently directed toward verifying the basic theory.

5.1. Hardware Probes

The hardware to be added to the system under measurement is a simple digital output port (figure 3) and some connectors. The output port consists of a sixteen bit module-number probe-register and a thirty-two bit stimulus probe-register; each with a light emitting diode display and a connector. The stimulus probe-register can be used to store a single thirty-two bit variable, two sixteen bit variables or four eight bit variables. The light emitting diodes are for visual analysis and user confidence only.

Software can store information in these registers pertaining to program operation at any time. When either probe-register is updated a clock pulse is sent to the analyser. Software overhead would be less in a hardware system that used memory mapped input/output than in a system that had special input/output

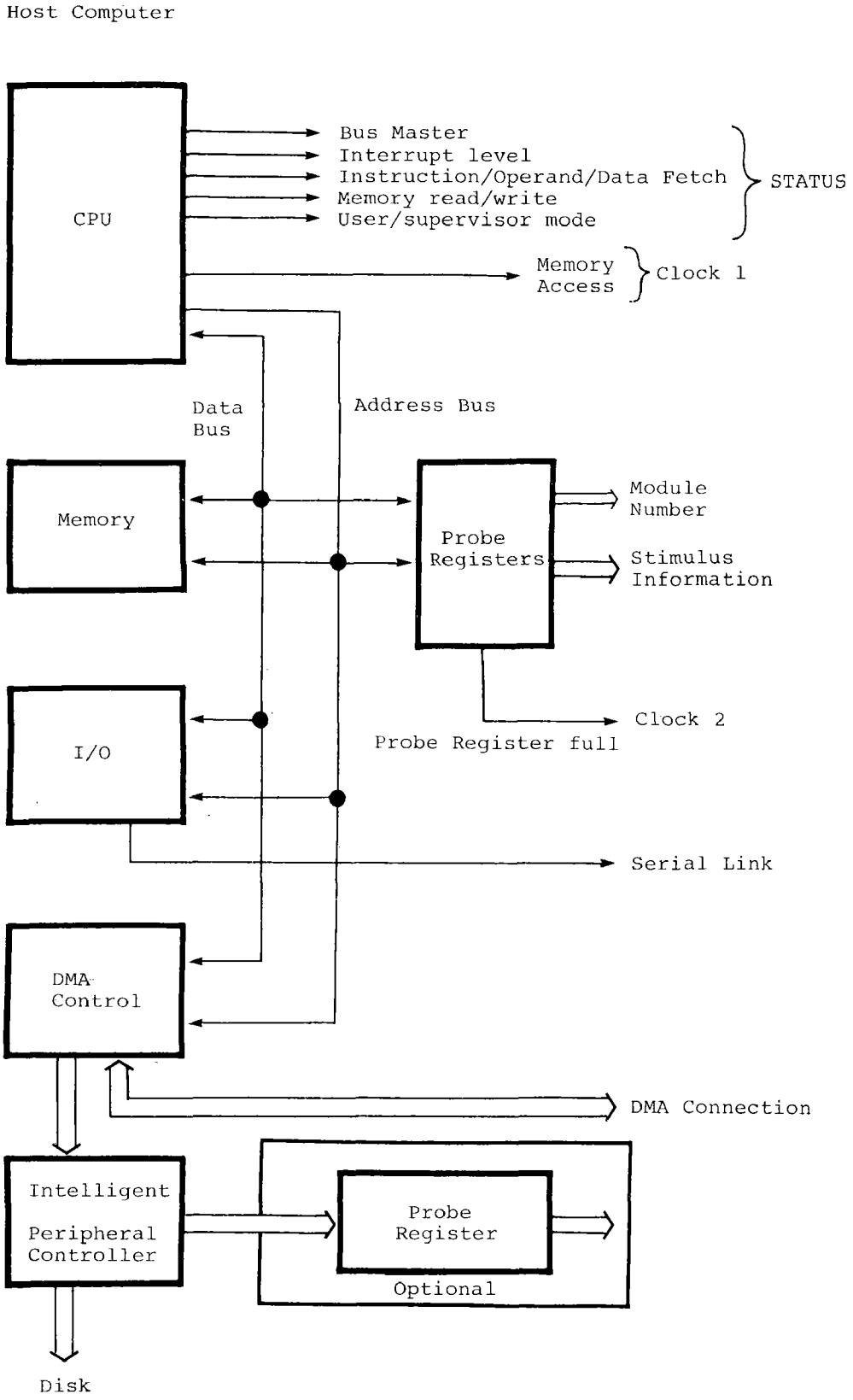

Figure 3. Hardware Probes

instructions, especially if they could only be executed in supervisor mode.

Other hardware includes connectors for a serial communications port, a direct memory access channel (only necessary in some situations) and state analyser signals including the address bus and processor status lines, e.g. user/supervisor mode, interrupt level, type of memory access, etc.

5.2. Software Probes

Every module of code (including processes, traps, interrupt handlers, subroutines) is given a unique module number. A module is a contiguous piece of code that is executed in response to an event. An event is any action that initiates a significant change in system state. Events include interrupts, software traps, process switches, subroutine calls and, at a higher level, user commands, but do not include programme branches. The first statement in a module writes the module number to the output port. The number remains there until that module completes execution and the next modules overwrites it. When a subroutine is called the calling process saves the number of the calling module so that on return from the subroutine it can be written to the output port. Thus the number in the module-number probe-register always corresponds to a unique module and is there only while that module is executing (or not executing in the case of idle).

In the case of a re-entrant module, such as a terminal driver, additional numbers can be written to the stimulus register indicating, in this case, terminal number and calling module number. In the case of a disc driver, module information such as disc number, channel number, file number, number of records transferred or calling module number could be written out. Operating system specific information, e.g. number of active tasks, can be written out when operating system modules, e.g. the scheduler, are executing.

A system module map must be available to the operator, preferably in a form that can be transfered to the analyser for display to the analyst and for use by the analysis programmes. Entries in the map include the name, function, module number and stimulus variables that are written out for the each module.

For many evaluation studies it may be necessary to update the stimulus probe-register several times during the module. The sequence in which information is written to the stimulus probe-register is defined in the module map. Sequence order variation between different module execution paths can be compensated for by assigning several sequential module numbers to the module. In effect a module can be broken up into sub-modules , each with a unique module number, if the situation requires. The flexibility of the software tool can be increased if stimulus information is obtained using pointers rather than directly. This allows either

a table of pointers from which one, defined by a global index, is selected or pointers can be modified either by software from a terminal or by the external analyser over the direct memory access connection.

5.3. Implementation

The hardware and software needed in the computer system are easily included in an integrated system. A structured approach to operating system design should simplify the inclusion of the necessary software. The overhead created by the inclusion of the software should be minimal and in fact can be measured by the analysis tool. If the overhead is considered too high it could be reduced by using an automatic checkpoint insertion system similar to that used in the Informer [23] where the software is only installed for the measurement period. Modifications to an existing operating system may be difficult particularly if it has grown like Topsy. Partial implementation to allow study of specific modules can easily be achieved either by modifying process start and termination supervisor calls or by providing a small output routine for user programmes. In either case a module map will be needed.

In a new design the additional hardware could be easily included. Existing hardware could be modified by adding a digital output port (preferably memory mapped) to the system. Other probe points should also be wired to connectors to make attaching and detaching the analyser easier. A further advantage of the registers being memory mapped is that the analyser can test the probes out by writing to the registers over the direct memory access channel and then reading the registers via the probes. The address bus probes could also be checked.

5.4. Analyser

The signals available for analysis are:

(i) Probe register contents - module number etc.

(ii) Address bus

(iii) CPU status signals

- Processor mode, user/supervisor,
- Memory read/write,
- Memory read type, instruction fetch, operand fetch, data fetch,

- Bus master, CPU/DMA,
- Interrupt mask/level,

(iv) Serial link.

(v) DMA connection.

(vi) Clock signals

- Memory access,
- Probe registers full.

These signals must be collected, reduced, analysed, displayed and recorded in real-time. The type of tool (figure 4) needed is an extension of the logic state analyser [1, 24] but with greater intelligence and flexibility. Thirty-two bit micro-computers have the power needed to make this possible at relatively low cost. It must be easily attached to the computer to obtain the above signals, using quick-connect connectors, and must not interfere with the host hardware. The typical probe width is 98 bits consisting of:

(i) probe register 48 bits,

(ii) address bus 32 bits,

(iii) CPU status 16 bits,

(iv) and 8 bits for clock generation.

State analyser front end technology typically samples at 25 MHz, sufficient for reading individual memory cycles of current mini and micro-computers.

The ability of the state analyser [1] to selectively trigger and selectively trace specified sequences of states needs to be expanded to allow at least twenty states to be specified in each sequence. It should be possible to specify a trigger sequence and a trace sequence for each clock. The resultant time-stamped trace will be a mixture of both traces. A flag will indicate which sequence the sample belongs to. Also the ability to trace a number of unknown states before and after a specified state would be very useful for determining which programme called the module under study.

In addition the analyser should be double buffered so that it can monitor continuously. Current state analysers stop reading when their buffer is full. Even in their pseudo-continuous mode, input data is ignored until the buffer is transferred to a mass storage device. Continuous monitoring allows the analyser to analyse, display and record one trace while continuing to trace, consequently no data is lost.

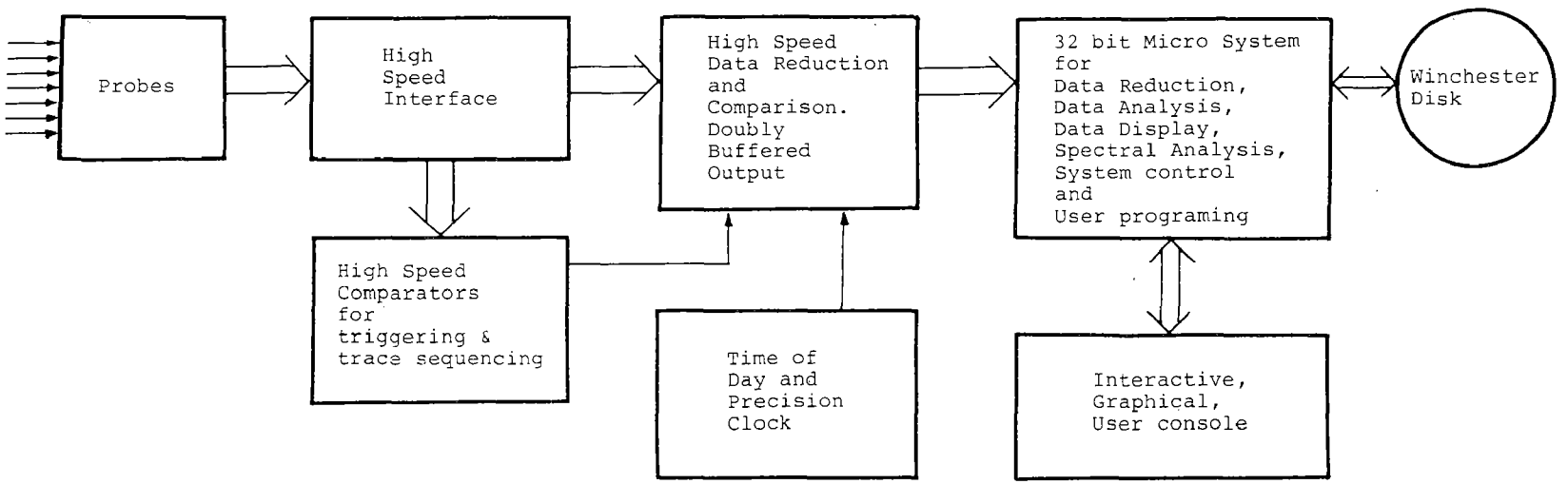

Figure 4. Hybrid Real-Time Performance Analyser.

A current limitation of logic state analysers is their inability to modify the clock signal, by logically combining signals to form a composite clock. Also the ability to combine the clock signal with an analysis input to produce a composite clock is desirable, for example combining the memory access clock with the instruction fetch signal to produce an instruction fetch clock.

6. HYBRID ANALYSER DESIGN - DATA ANALYSIS

The real power of the tool is determined by the analysis it can perform on the collected data. As each module is executed its exact execution time and frequency of execution can be measured, simply by tracing the contents of the probe registers. Normal entry and exit from modules is confirmed by monitoring entry and exit addresses obtained using the address-bus probe. Thus, significant information can be obtained just by reference to the module map and simple macro-level tracing. The analyst does not have to consult the "mythical" system link map, which will change the next time the system is linked, to find module starting addresses. In fact the analyser can find them for him directly, if they are needed.

The execution time for any module will vary depending upon the path taken through it and thus some form of data reduction is needed.

6.1. Models

Kumar and Davidson [25] have argued that a hierarchy of performance models ranging from analytical models to detailed simulation models, is a very useful tool in the design of computer systems. The development and validation of these models, both structural and functional, require the measurement of actual system values.

(i) Service times and visit ratios for analytical queueing models [26], can be calculated from data produced by taking the mean of the execution times of the appropriate modules and their frequency of usage during the period under study. Multiprogramming level can be obtained either as a probe output from the scheduler or by averaging the number of user programmes executed during the measurement period. Job class is an obvious probe output for user programmes. These calculations can be provided both as a standard instrument function or implemented by user written software running in the analyser.

(ii) A simulation model represents the behaviour of a system in the time domain. As there is a conceptual similarity between simulation and measurements the results obtained by measuring module execution time and

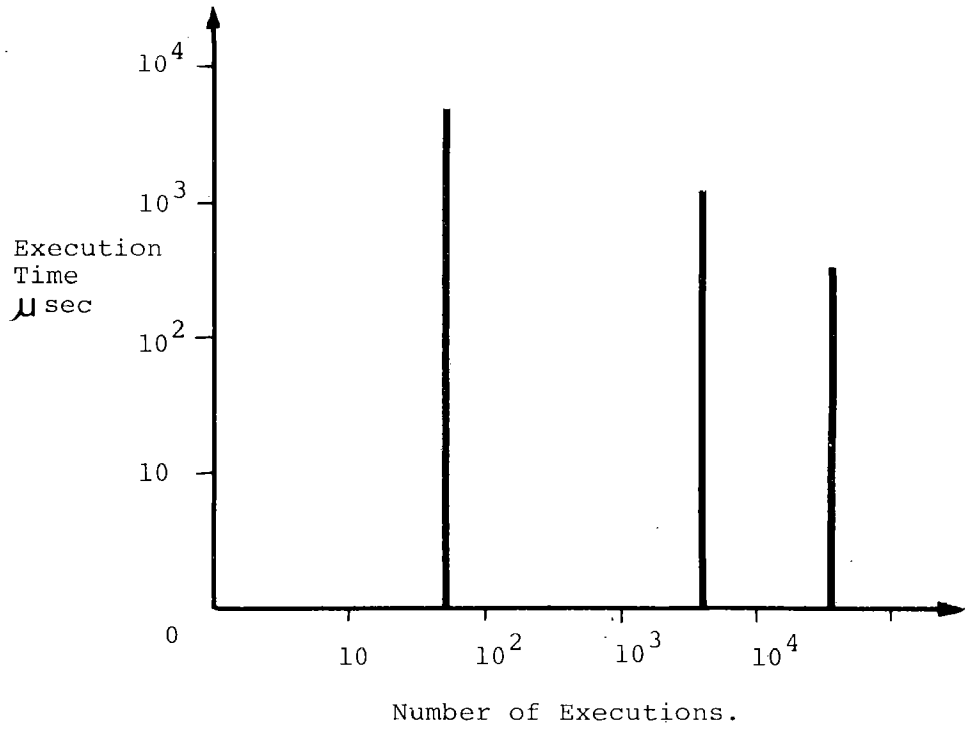

Figure 5. Program Spectrum.
e.g. Time of day clock measured over a one hour period.

Average frequency of Execution

$$= \frac{\text{Total Number of Executions}}{\text{Length of Measurement Period}}$$

sequence can be used for both development and validation of simulation models.

6.2. Module Spectral Analysis

A frequency spectrum is produced by plotting the amplitude of an electrical signal versus the frequency of the sameesignal on a graph. A module spectrum (figure 5) can be produced by plotting the time taken to execute the module verses the number of module executions. As there is a discrete number of paths through a module the spectrum consists of a vertical line for each execution path. Continuous monitoring of a particular module allows a history of execution to be accumulated from which a module spectrum can be plotted. This could be done at the end of a measuring period or displayed on a video monitor, in real-time, so that the progressive development of the spectrum can be observed.

Once the spectrum is obtained the analyser could be armed to trigger on the occurrence of a particular execution time and to trace the path taken through the module. Comparison of this to the code will identify the exact execution path. Analysis of the other data stored in the probe registers should provide sufficient stimulus information to characterise the system at the time. Data to be stored in the stimulus probe-registers is an important consideration in module design.

Module spectral analysis indicates areas of poor performance that require further investigation. Tracing a desired path through the module allows detailed analysis of the problem. Thus the tool is useful for finding and analysing bottlenecks. Module spectral analysis is one example of a general purpose graphics utility which will accept any two input variables and plot them either in real-time or at the end of the measurement period.

A higher level example is the concept of task spectral analysis where both the modules and the time between modules of a sub-system are monitored. The analysis of disk file handling falls into this category. A task is a collection of modules that interact to perform a desired operation. Examples include disk file transfers, interactive editing, process switching and batch processing.

The highest level is spectral analysis of the whole system, facilitating such measurements as the detection of the most commonly used modules and resource usage by job class or multiprogramming level.

6.3. Module Execution Path

Connection of the address bus to the analyser allows the execution paths through any module to be traced and measured.

One obvious advantage this tool has over a logic state analyser is that the majority of measurements can be made without having to dig through the code. Module entry and exit addresses can be determined simply by reading the address bus when the module-number probe-register is updated.

Setting the analyser to record all instructions executed during a module allows the analyser to accumulate an empirical flow diagram of the module. Comparison of execution paths quickly pin points branches and loops. Thus a complete flow diagram of the module, which shows not only execution paths and address relative to the start of the module but execution times and loop frequencies as well, can be constructed and displayed.

Symbol table information, obtained by the operator over the serial link to the host, can then be used to place labels on the flow diagram. For high level languages it may be possible to place line numbers on the diagram. For high level languages that produce assembler mnemonics as an intermediate compilation stage both labels and line numbers could be placed on the diagram.

At this stage the code can be consulted to fill in final details and to analyse time consuming sections etc. It is this feature which makes the analyser much easier to use than other tracing tools. The operator can obtain the majority of the information needed without having to refer to link maps and assembly code.

Discontinuities occur in module execution as a result of servicing external interrupts, waiting for input/output transfers, supervisior calls and time slicing. These can be detected and their effect eliminated by monitoring the module number while tracing the instruction path.

At a higher level, monitoring the modules in a system or sub-system enables typical task execution paths to be established and task flow diagrams to be displayed. From this task bottlenecks can be pinpointed and analysed. The presence of stimulus information at the probes allows the effect of job class on task operation to be analysed.

6.4. Performance Measurement Hierarchy

A top down approach to performance measurement is both desirable and achievable. Execution time, execution path, execution frequency and stimulus information can all be measured at system level, task level and module level. Understanding individual modules in an operating system is relatively easy but understanding the complex web of interactions between modules can be a mind boggling exercise. Establishment of typical task execution paths should ease this problem.

6.5. Come From

It is often desirable to determine which processes call a particular module and how often, i.e. how did the computer get here. This tool can be set up to trigger on entry to a module and record either the number of the previous module or a specified number of instructions prior to the module entry point.

6.6. High Speed Data Transfer

The direct memory access connection allows the analyser to read specific memory locations, e.g. contents of loop counters, process table numbers, etc. Also the analyser would be able to modify memory locations in the host computer e.g. tuning constants in an adaptive control loop.

Information such as operating system configuration, module map and symbol tables could be transferred over this link or over the low speed serial link. An obvious extension to this is to make the analyser an intelligent peripheral so that measurements can be requested from the host computer. In large computer systems this would be economical but in many installations a portable tool would be desirable.

6.7. Multics Style Instrumentation

Multics was developed as a research project whose intent was to create an operating system centred around the ability to share information in a controlled way which could support a wide variety of computational jobs [6]. Instrumentation, integrated into the system at design time, proved very useful in detecting performance problems and identifying the cause. The measurements made by the Multics hardware tools can all be done by the analyser as defined.

Some of the software tools included in Multics performed the tasks described below:

(i) A general measuring package recorded the time spent executing selectable supervisor modules and their frequency of execution.

(ii) A segment utilization meter sampled, every ten milli-seconds, the segment number of the segment which was executing and stored the result in a table. This provided a simple way of detecting how time spent in the system was distributed among the various components.

(iii) The number of missing pages and segments encountered during execution in a segment was recorded on a per segment basis.

(iv) The number of procedure calls was counted.

(v) A graphics display monitor [28] on a separate computer connected by a channel, included a variety of standard displays which were used to observe the traffic controller's queues, the use of primary memory and arrays produced by other tools. During system initialization, Multics built a table containing pointers to interesting data bases for use by this hybrid tool.

(vi) The effect of the systems multiprogramming effort on an individual user was traced.

(vii) Feedback was provided to the user about the resource utilization of the command just typed.

All the monitoring tasks performed above can be carried out using the monitor as described except the last which is more properly a user service.

6.8. Accounting

One of the problems with software added to a system for measurement is interference with the system being monitored. In this case the tool can be used to measure the interference. Also the interference can be compensated for by reducing the accounting programme to provide only user specific accounting data as all other information can be obtained more easily and more accurately by the analyser.

7. Flexibility

In many circumstances the analyst is interested in the occurence of events and the relationship between these events. All events result in the execution of a module. A page fault causes a handler routine to be executed. The onset of thrashing can be detected by monitoring the frequency of execution of the swapping modules.

User programmability and a suite of general purpose graphics display programmes allows the analyst to tailor the analysis to his particular application. Considerable research is being conducted into the design and improvement of algorithms. The addition of a special high speed supervisor call would enable user programmers to pass information to the probe-registers for accurate analysis of algorithm performance.

8. THE PROBLEM OF CACHE MEMORY

Cache memory creates a significant problem for state analysis. The analyser must be connected to the address bus after relocation from virtual to physical address space and before cache memory otherwise the addressing information will not reflect programme execution. Also the memory-read status signals must represent processor requests not cache requests. On some minicomputers, the cache memory is an integral part of the processor and the address bus is only accessible after cache. Thus it does not contain the address of the current instruction during a cache hit. On a cache miss several instructions (which may not be executed) after the current one are read into cache, and again the addressing signals to main memory may not reflect programme execution.

Adding an additional status signal which indicates whether the current memory reference is a cache hit or miss allows cache performance for instruction fetches, operand fetches and data fetches to be measured for individual modules. Once again, the need to define performance measurement requirements at design time and build them into an integrated hardware/software design is emphasized.

9. CONCLUSION

Throughout this paper a theory of performance measurement has been developed. Performance requirements vary from system to system and user to user within the same system. Even though the actual variables measured depend upon the context, the type of measurement is the same in all contexts. The basic measurements are:

(i) Execution time.

(ii) Execution path.

(iii) Frequency of execution.

(iv) Bottleneck identification.

(v) Stimulus information related to the above.

A hierarchy of performance analysis strategies is provided. The system under study can be broken down into a number of tasks. A task is a collection of modules that interact to perform a desired operation. A module is a contiguous piece of code that is executed in response to an event. An event is any action that initiates a significant change in system state. A module map defines the information contained in the probe-registers during the execution of each module.

An easy-to-use, real-time, hybrid performance analyser which

will provide the above information has been described. It can be added to an existing system or implemented as part of an integrated design of a new system.

The real power of the tool is in its ability to reduce, analyse and display the data being read. Execution time, execution path, execution frequency, and stimulus information can all be obtained without having to refer to the code of the modules, considerably simplifying the measurement process. A flow diagram of a module or a task can be built up and displayed. Access to symbol tables allows the labels and source code line numbers to be included on the display. At this stage, when the module is fully characterised and understood, the analyst consults the code to fill in final details or make corrections.

Real-time data reduction and analysis provides the information needed for model development, model verification, module spectral analysis, task spectral analysis, bottleneck analysis and adaptive system control.

Bibliography

1. P.J. McKerrow, Evaluation of Interrupt Handling Routines with a Logic State Analyser, Submitted to Performance Evaluation, North Holland.

2. P.J. McKerrow, Control of Coating Mass on a Continuous Hot Dip Galvanizing Line, Master of Engineering Thesis, The University of Wollongong, 1978.

3. C.H. Sauer & K.M. Chandy, Computer Systems Performance Modelling, Prentice-Hall, 1981.

4. D. Ferrari, Computer Systems Performance Evaluation, Prentice-Hall, 1978.

5. D. Ferrari, Architecture and Instrumentation in a Modular Interactive System, Computer 6, 11, November 1973, pages 25-29.

6. J.H. Saltzer & J.W. Gintell, The Instrumentation of Multics, Communications of the ACM, Volume 13, Number 8, August 1970, pages 495-500.

7. G. Boulaye et.al., A Computer Measurement and Control System, Proc 3rd International Symposium, Measuring, Modelling and Evaluating Computer Systems, October 1977, North Holland.

8. C.D. Warner, Hardware Monitors: The State of the Art, Infotech Reports, Vol 18, 1974, pages 623-631.

9. Carlson, Hardware Monitoring for System Tuning, Infotech Reports System Tuning, 1977, pages 255-274.

10. W.C. Lynch, Operating System Performance, Communications of the ACM, Volume 15, Number 7, July 1972, pages 579-585.

11. Update on Hardware Monitors, EDP Performance Review, Volume 2, Number 10, October 1974, 8 pages.

12. D. Ferrari, & M. Spadoni, Lecture Notes of the Second Summer School on Computer Systems Performance Evaluation, SOGESTA, Urbino, Italy, June 1980, North Holland.

13. W.T. Wilner, Design of the Burroughs B1700, AFIPS Conf. Proc. 41 (FJCC), 1972, pp 489-497.

14. L. Svobodova, Online System Performance Measurements with Software and Hybrid Monitors, Operating Systems Review, 7, 4, October 1973, pages 45-53.

15. A.H. Agajanian, A Bibliography on System Performance Evaluation, Computer, Volume 8, No. 11, November 1975, pp 63-74.

16. G.J. Nutt, Tutorial: Computer System Monitors, Computer, Volume 8, No. 11, November 1975, pp 51-61.

17. A.G. Nemeth & P.D. Rovner, "User Program Measurement in a Time-Shared Environment", Communications of the ACM, Vol 14, No. 10, pp 661-666, October 1971.

18. D.F. Stevens, "System Evaluation of the Control Data 6600", Proceedings of the IFIP Congress, pp 34-38, 1968.

19. W.M. Denny, The Burroughs B1800 Microprogrammed Measurement System: A Hybrid Hardware/Software Approach, Proc of the 10th Annual Microprogramming Workshop called MICRO 10, ACM, New York, 1977.

20. G. Estrin, D. Hopkins, B. Coggan and S.D. Crocker, SNUPER COMPUTER - A Computer in Instrumentation Automation. AFIPS Conf. Proc 30 (SJCC), pp 645-656 1967.

21. J. Hughes and D. Cronshaw, On Using a Hardware Monitor as an Intelligent Peripheral, Performance Evaluation Review, 2, 4, pages 3-19, December 1973.

22. R.J. Ruud, The CPM-X, A Systems Approach to Performance Measurement, Proceedings of the FJCC, Vol. 41, Part II, pp 949-957, 1972.

23. P. Deutch and C.A. Grant, A Flexible Measurement Tool for Software Systems, Information Processing 71, Proc IFIP Congress 71, North Holland, 1971.

24. R.W. Comerford, Measurement-Computer Era Arrives, Electronics, September 8, 1981, pages 96-100.

25. B. Kumar & E.S. Davidson, Computer System Design Using a Hierarchical Approach to Performance Evaluation, Communications of the ACM, Vol 23, No. 9, September 1980, pp 511-521.

26. C.A. Rose, A Measurement Procedure for Queueing Network Models of Computer Systems, Computing Surveys, Vol 10, No. 3, September 1978.

27. P.R. Sebastian, Hybrid Events Monitoring Instrument, Proc 1974 SIGMETRICS Symposium, ACM, New York, pp 127-139.

28. J.M. Grochow, Real-time Graphic Display of Time-Sharing System Operating Characteristics, Proc AFIPS 1969 FJCC, Vol 35, AFIPS Press, pp 379-386.

29. D.E. Morgan, W. Banks, D.P. Goodspeed, R. Kolanko, A Computer Network Monitoring System, IEEE Transactions on Software Engineering, Vol SE-1, No. 3, September 1975, pp 299-311.

30. A. Geck, Performance Improvement by Feedback Control of the Operating Sytem, Performance of Computer Systems, North Holland, 1976, pages 459-471 Ed. M. Arato.