U.S. DEPARTMENT OF COMMERCE • National Bureau of Standards

NBSIR 87-3663

Institute for Computer Sciences and Technology

Recommended Instrumentation Approaches for a Shared-Memory Multiprocessor

George Nacht
Alan Mink

U. S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Advanced Systems Division
Gaithersburg, MD 20899

October 1987

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

Partially sponsored by the
Defense Advanced Research Projects Agency
under ARPA order number 5520,
July 23, 1985, and July 28, 1986.

Recommended Instrumentation Approaches for a Shared-Memory Multiprocessor

George Nacht
Alan Mink

Advanced Systems Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Work reported in this report was partially supported by the
Strategic Computing Initiative
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209
ARPA Order No. 5520, July 23, 1985,
as amended July 28, 1986.

U.S. Department of Commerce, C. William Verity, Jr., Secretary

National Bureau of Standards, Ernest Ambler, Director

October 1987

TABLE OF CONTENTS

Page

Recommended Instrumentation Approaches for a Shared-Memory Multiprocessor

George Nacht
Alan Mink

Two approaches for the design of performance measurement instrumentation for a shared memory, tightly coupled, MIMD multiprocessor are presented. The TRAce Measurement System (TRAMS) is a hybrid measurement tool used to obtain trace measurement information. The Resource Measurement System (REMS) is a non-intrusive hardware measurement tool used to obtain both trace measurement and resource utilization information. The TRAMS approach provides a hardware assist to the more traditional software approach of obtaining timestamps from the operating system at each event to be measured. This hardware assist reduces the artifact that is introduced in a test program and is a feasible and economical approach to providing measurement capabilities to a wide range of multiprocessors. Manufacturers could offer this type of measurement tool as a plug-in option. The REMS approach provides more detailed and extensive measurement information than does the TRAMS approach and introduces no artifact to the test program, but it does this at a significantly higher cost. When access to pertinent signals is restricted the applicability of such a hardware tool is limited.

Key words: Multiprocessor computers; parallel computers; performance measurement; computer performance.

1. INTRODUCTION

At the highest level of abstraction, performance measurement can be thought of as the information to be measured (raw samples) and the mechanism (tool) used to obtain that information.

The performance information to be measured can be placed into two orthogonal categories: (1) trace measurement, and (2) resource utilization. Trace measurement is concerned with the activities of the application or system processes, and provides information such as program execution time, execution path, response time, etc. Hardware, software, or hybrid (a combination of both) measurement tools can be used to obtain this trace information. Resource utilization is concerned with the detailed operation of the hardware, and provides information such as cache hit ratios, access delays, duty cycles, etc. Hardware measurement tools are required to obtain this class of information, due to the signal speeds involved and the fact that these signals are not

visible to software measurement tools. Roberts [ROB85] provides a more complete discussion on performance information to be measured.

Software and hybrid tools are intrusive in that some additional code is executed either in the measured program or in the operating system. This affects the timing characteristics of the measured program. Hardware tools are non-intrusive as they do not change the operating characteristics of the program. A common feature of these measurement tools is the triggering facility. A trigger is the mechanism which causes a measurement sample to be taken. Software and hybrid tools use either interrupts or embedded code to cause a trigger. As an example, a timer causes regular and frequent interrupts. Each interrupt causes a sample to be taken which consists of the current program location. These samples are later used to generate a program execution profile. Another example is the more standard software approach in which the execution time of a known segment of code is desired. Two events would be marked in the test program by the addition of extra code at the beginning and at the end of the code segment. This extra code requests a timestamp from the operating system and stores it along with information which identifies the requesting event. The time difference between the first and second events is the execution time of that segment. In the first example the trigger is the interrupt which was caused by a device external to the measured program. In the second example the trigger is the additional code, and is a function of the location of that code in the measured program. Hardware tools, being non-intrusive, must constantly monitor the signal lines to which they are attached, and when specific signal patterns occur, cause a trigger. Carpenter [CAR87] provides a more complete discussion on the entire range of measurement techniques and their associated trigger mechanisms.

These techniques are applicable to both uniprocessor and multiprocessor systems. For multiprocessor systems, the process of measurement is more complex due to the additional processors and the cooperating processes they are executing. Measurement tools which cause perturbations, or artifacts, in the timing characteristics of these cooperating processes can alter their cooperative execution and thus their performance. For this reason, hardware measurement tools which cause no perturbation or hybrid measurement tools which cause minimal perturbation are deemed the most desirable tools for multiprocessor systems. The threshold of perturbation that any given program can tolerate without altering its execution is program dependent. In general, the larger the granularity of each of the cooperating processes and the less frequently communication/synchronization occurs between them, the higher the perturbation tolerance.

Our model of the use of multiprocessors is one in which a single application program is broken into a number of cooperating processes. These processes are then distributed among the available processors to reduce the execution time. The performance concerns are focused not only on the application program, but also on the multiprocessor architecture and the implementation of that architecture. There is generally a performance hierarchy that starts at the application level. As performance is characterized at that level, there is a desire to delve into the underlying causes of the measured performance. Thus performance is measured from the application, through the operating system, down to the hardware operation. Because of this performance hierarchy, a measurement tool that is capable of acquiring performance information on both trace measurement and resource utilization is necessary.

Our present focus is on multiple-instruction-stream, multiple-data-stream (MIMD), tightly coupled, shared memory multiprocessor systems. We have previously reported on a hybrid measurement tool which acquires trace measurement information for such a multiprocessor [MIN86, MIN87]. A successful prototype was built and is currently functioning in our laboratory. A review of the design of this hybrid prototype is presented in this report. Based on this experience and the fact that non-intrusive trace measurements and resource utilization measurements are needed, a hardware measurement tool that is capable of accomplishing these measurements is nearing completion. This report discusses the design of that hardware prototype and the insight gained into its advantages and limitations for this class of multiprocessors.

2. HYBRID APPROACH

Our hybrid performance measurement system is a simple hardware assisted extension to the traditional software measurement approach. The traditional approach to performance measurement is to modify a program to be measured with the addition of supervisory calls to the operating system. These calls, which denote *events*, are inserted at the beginning and end of a segment of code whose execution time is desired. The calls obtain timestamps, which, along with information identifying the event, are stored in a buffer area in memory and are analyzed later at some convenient time. A major problem with this traditional approach is that it perturbs the original program by introducing artifacts. In particular, additional execution time is required both for the operating system to provide the timestamp, and for the additional code to store both the timestamp and the data describing the event. This time overhead is on the order of hundreds of instructions, or worse. In a uniprocessor running a non-time-critical application, such time perturbations merely increase execution time but do not otherwise effect the execution path of the program. In a multiprocessor and in a uniprocessor running real-time applications, time perturbations can greatly distort the execution path and, therefore, dramatically change the performance.

Our hybrid preformance measurement tool, called the TRAce Measurement System (TRAMS), is a memory-mapped I/O device which maintains its own clock for timestamping. The overall configuration of TRAMS is illustrated in Figure 1. From the programmer's point of view TRAMS is treated as a memory location. At selected *events* in the test program, writes to memory are performed, where the write address is that of a custom designed board called the Event Data Card (EDC). These writes are called *edc* measurement statements, and contain data specified by the user to identify an event, such as process ID, values of variables, or other program specific information. Since no supervisory calls are made to the operating system, only a few instruction times are required for the computer being measured to execute each *edc* statement. The time perturbation caused by the additional TRAMS measurement software which is added to the program is as little as a single memory write statement per timestamp. In our current implementation, the event data is fixed at 11 bits. The data, used as one field, is sufficient to identify 2048 unique events. The data, if divided into two fields of 3 and 8 bits each, can identify 8 separate processes and 256

unique events. The EDC is connected into the IEEE 796 bus of the system under test and, in addition, has a number of direct connections (probes) to each of the processors. These probes obtain information that is not available on the bus, e.g., processor identification. The EDC augments the 11-bit event data field with a 32-bit, one microsecond timestamp, a 4-bit processor identification and a 1-bit user/supervisor mode status, for a total of 48 bits.

Obtaining measurements with TRAMS requires familiarity with the application program to be measured. The programmer determines which segments of code are to be measured and demarcates those segments with *events*. The programmer then determines the event-identifying data that will be output to the EDC. The programmer inserts into the application program source code a single call to an initialization routine and also inserts *edc* measurement statements at each event. The initialization routine uses the operating system to fix the mapping between the fixed physical address of the EDC and a virtual address to be used by the application program. The *edc* measurement statements write data to the EDC which identifies each event. The EDC passes this event data to a First-In-First-Out (FIFO) buffer along with a timestamp, the identification of the processor and its mode (user/supervisor). The sample data in the FIFO is read by the analysis computer and stored for later processing. Since the analysis computer is separate from the system being measured, this can happen either on the fly while the application program is still executing or after it completes. It should be noted that the timestamp is a continuously running clock, i.e., wall clock time, and does not differentiate between user and supervisor time. To obtain such a separation would require the programmer to "instrument" the operating system in a similar manner as was done to the application program, e.g., place *edc* measurement statements at all entrances and exits of the operation system.

An analysis program takes the timestamped application-specific data, matches event pairs which demark code segments, computes the corresponding time intervals and produces a statistical summary for these measured code segments. We have primarily been using a fixed format for the event data, e.g., process ID and event number. Therefore, a table-driven analysis program has been developed which can be adapted to different measurement experiments by modifying the table describing the events without modifying the program.

Intervals and frequencies are two major categories of metrics. Intervals require paired events: the start of the interval, and the end. Frequencies may require one or two events, depending on the variation of the metric. In the simplest form, only one event is necessary to keep a simple count, e.g., the number of times per second that a loop is executed. In more complex variants, two or more events are necessary to determine ratios, e.g., three or four events may be required to determine the ratio of the execution time of a critical part of a loop to the execution time of the total loop.

The major design goal of the EDC was to produce, with a minimum of design effort, a simple, straightforward interface between the multiprocessor system being tested and the measurement sampling, storage and analysis equipment. The EDC is custom designed and consists of an IEEE 796 interface, an address recognizer, a data latch, a timestamp counter, a synchronization circuit to assemble the data from various sources and generate a trigger, an interface for hardware probes, and logic for intermediate processing of signals from the hardware probes. The availability of a

reliable 10 MHz clock signal on the IEEE 796 backplane makes it unnecessary to include a clock circuit. To minimize the implementation effort, we chose an available logic analyzer as the FIFO in our TRAMS, which had the added benefit of simplifying the EDC output function. The logic analyzer is attached directly to the appropriate signal sources on the EDC.

The data latch has sixteen outputs bits: eleven bits of software-determined data from the processor initiating the write, plus five bits of data from the hardware probe logic. On our current system under test certain hardware specific information from the processor is neither available on the backplane nor active when needed by the EDC logic to extract the processor ID and user/supervisor mode information. The probes, therefore, connect to the processor cards and store this extracted information until needed by the EDC.

In our current implementation, a logic analyzer with a 512-word buffer is used as the FIFO to store event data. One complete item of data, including the timestamp, is captured every time a trigger is generated. The logic analyzer used in this measurement chain suffers from limitations in capacity, speed, and functionality. Its 512-word FIFO buffer size is a severe limitation. Another problem with using a logic analyzer is that it is general purpose test equipment which is frequently being used elsewhere in our laboratory. This has resulted in annoying and inconvenient delays due to the reconnection and reprogramming requirements of the logic analyzer. The logic analyzer cannot report how many samples have been taken; therefore, the entire buffer must be transmitted each time. The effective transmission speed of event data from the logic analyzer to the analysis computer via the IEEE 488 interface is on the order of seconds for the entire 512-sample buffer contents. Yet another problem with the logic analyzer is that while it is transmitting its buffer contents to the analysis computer via the IEEE 488 interface it cannot capture new samples. It also cannot detect how many samples have been lost during this transmission. Based on these current limitations, we have redesigned the TRAMS to include a *much* larger on-board memory instead of the logic analyzer. This memory may be read at normal memory speeds either on the fly or later after the experiment has completed. This will eliminate the problems associated with the logic analyzer and the IEEE 488 interface.

The TRAMS concept is quite general and can be applied to various multiprocessor architectures. Our implementation of TRAMS is for a commercially available 32 bit, 10 MHz, tightly-coupled, shared-memory multiprocessor system. We have provided a general software measurement interface to the hardware device that captures and timestamps the event data. Rather than embedding the software measurement interface only in the operating system, we have provided both the applications programmer and the systems programmer access to this measurement tool. The information describing the event is defined by the user and is not predetermined. Each individual timestamped event is captured for offline processing rather than being preprocessed in fixed metrics. Because processes can migrate among the various processors, each event sample identifies the processor on which that event is currently executing. As a result we can reconstruct the specific sequence of events, as well as compute performance metrics.

3. HARDWARE APPROACH

In the hybrid approach described in the section above, events are denoted by the addition of *edc* measurement statements to the application program to be measured. These statements cause samples to be taken which identify event, time, processor and process. This contrasts to the hardware approach where events are defined by the state (value) of certain signals. These states are called *patterns* and are user defined prior to an experiment. Dedicated hardware searches for these patterns, and cause samples to be taken when they occur. As these are signal states, they may be used to define not only application program location (trace information) as in the hybrid approach, but also system component function (resource utilization). This hardware approach is discussed in the following sections.

3.1 OVERVIEW

The REsource Measurement System, REMS, consists of a number of Sample Units (SU) connected to an analysis computer via a bus, see Figure 2. An SU is the logic that captures, processes, and stores measurement samples. There is one SU for each CPU in the multiprocessor under test. Each SU connects directly to the hardware signals of a processor under test via a set of wires called probes. An SU consists of three major modules: a Pattern Matcher, a Preprocessor and a Sample Memory. The Pattern Matcher is the logic that constantly monitors the hardware signals of the processor and compares those signal patterns against a set of stored signal patterns. When a match occurs, a trigger is generated that causes a measurement sample to be taken. The data output from the Pattern Matcher is the pattern that was matched. The predominant signal patterns monitored by the Pattern Matcher are those of the CPU bus containing address and data. The Preprocessor performs a data reduction on resource utilization events to decrease the number of samples required and/or the number of bits in a sample. One type of Preprocessor monitors certain hardware signals and keeps cumulative counts of the occurrence of these signals. These signals usually occur frequently and are high speed signals, and the value of interest is the number of occurrences. The Preprocessor, therefore, relieves the measurement system from the burden of monitoring and triggering on these frequent signals and filling the sample memory with all of their occurrences, which would be reduced to cumulative counts during postprocessing. The data output of the Preprocessor is the current count values. *The Preprocessor and the Pattern Matcher of any single SU do not have to be attached to signals from the same processor.* In general the Pattern Matcher module is concerned with obtaining trace measurement performance information, while the Preprocessor is concerned with obtaining resource utilization performance information. The Sample Memory is a large local memory which stores every measurement sample. A sample from a single SU consists of the combined data output of the Pattern Matcher and the Preprocessor. When a trigger is generated at any of the SUs, each of the SUs takes a measurement sample. In this way the activities of each of the processors under test may be correlated when any of them reach a critical point in the test program.

3.2 PROBES

Each SU is connected to the hardware signals of a processor via a set of probes. The probes have a two-fold purpose: they reduce electrical loading on the signals to which they are attached and they contain combinatorial logic with which a sample clock and state information are extracted from various timing signals. The sample clock is used to strobe a signal pattern (trigger data) into and through the pipelined Pattern Matcher.

The most likely hardware signals of interest to the Pattern Matching module are the CPU bus signals along with the necessary control and timing signals. The current format of this 32 bit trigger data is 24 bits for bus information, 3 bits for bus qualifier, and 5 bits undefined. In the multiprocessor we currently use as a test bed, the 24 bits of bus information can be any 1 of 3 types: virtual address, physical address, or data. All three types of information are multiplexed onto the same signal lines, therefore, the information on the bus is timing (state) dependent. Combinatorial logic on the probes constantly monitors and interprets the bus timing and control signals and generates 3 bits of bus qualifier data. 2 bits are encoded to indicate the type of bus information. The remaining bit indicates the current processor mode (user/supervisor). Each time the information on the bus changes, the probes generate a sample clock to capture the new trigger data and a new 3 bit bus qualifier. As the bus qualifier is part of the pattern to be matched (trigger data), any combination of bus information types can be acquired during a single experiment. The remaining 5 undefined bits can be used to examine any additional signal lines of interest for a given experiment.

3.3 PATTERN MATCHER

The probes provide a sample clock and the trigger data to the Pattern Matcher. The Pattern Matcher latches the trigger data and compares it against a set of stored signal patterns. If a match occurs, a global trigger is generated that causes every SU to take a measurement sample. In this way measurement information from all of the CPUs is taken when any of them reach a critical event and the activities of any CPU may be correlated with each of the others. The output of the Pattern Matcher, which is used as part of the measurement sample, is the trigger data which caused the match.

The two central design issues of the Pattern Matcher were its operational speed (100 ns per trigger pattern) and the ability to specify and match an arbitrary set of patterns which may be changed for each experiment. Two approaches to designing a matcher are combinatorial logic and memory. Combinatorial logic was ruled out since it would only operate on patterns fixed at design time. Of the memory approaches, associative memory would seem to be a good candidate since it could simultaneously search every memory location for the trigger pattern. This is really a combination of logic and memory, where each memory element has its own comparator. The two main problems with associative memory are availability and *don't care* states. Associative memory is not commercially available in any useful size, and is, therefore, too expensive. More important is that associative memories cannot handle don't care

states in their memory contents (the pattern set). The don't care states could be eliminated from the pattern set by expanding the number of patterns to include all variations of each don't care state but this would have the potential of greatly increasing the required size of the associative memory.

A standard random access memory could be used as a pattern matcher. This technique is analogous to a sum-of-products combinatorial logic expression. This requires one memory location for each potential pattern, not just the currently interesting pattern set (which is much smaller). The trigger data pattern is used as the memory address, while the contents of each memory location contain a true/false indicator; true if the pattern that corresponds to this memory address is in the current trigger data pattern set, false otherwise. The benefits of this approach are that standard random access memories are widely available from commercial sources, are relatively inexpensive, are a well known design component, and can easily be rewritten to change the pattern set. The problem with this approach is that the trigger data is 32 bits long, requiring a memory of 2**32 (approximately four billion) locations. A memory of this size is not currently feasible for this purpose, mainly due to cost and space limitations, but since all the other aspects of this approach are so attractive, further effort was expended attempting to reduce the size of the required memory.

We are dealing with a sparsely populated state space. The trigger pattern set is generally very small (hundreds) compared to the entire set (billions) of potential patterns. Using this fact the pattern set can be re-coded into a smaller state space. A direct memory lookup technique can be used employing a smaller, thus more feasible, memory. First a mapping between the original trigger pattern set and the re-coded pattern set is needed. This mapping can be accomplished through a direct memory lookup technique, and it can be done in parallel on disjoint pieces of the trigger pattern. There is a direct analogy between this decomposition and the decomposition of a single stage sum-of-products combinatorial expression into a similar multi-stage sum-of-products combinatorial expression. In this case three stages are used.

Figure 3 shows a block diagram of the Pattern Matcher module which contains 2 trigger data latches and 3 levels of memory, one level for each sum-of-product stage. The first and second levels of memory are concerned with the mapping into the re-coded reduced state space. The third memory level does the actual pattern match of the re-coded state space. The incoming trigger data (from the probes) are latched by the sample clock (also from the probes) at the input to the Pattern Matcher. The 32 bits are divided into 3 disjoint segments of 11, 11 and 10 bits respectively. Each segment is used as an address input to one of the three 2K x 8 memories at LEVEL 1. By design of the recoding process, the 24-bit output from the three LEVEL 1 memories contain only 22 bits of valid data; 7, 8 and 7 bits respectively. These 22-bits are recombined into two disjoint segments of 11 bits each. Each segment is used as an address input to one of the two 2K x 8 memories at LEVEL 2. The 16-bit output (8 bits and 8 bits) of the LEVEL 2 memories is then used as a common address input to seven 64K x 1 memories at LEVEL 3. The output from one of the LEVEL 3 memories is the "Global Trigger", which causes all the SUs to take and store a measurement sample. The output of the other six LEVEL 3 memories represent "Selective Triggers" and are control signals directed to the Preprocessors.

Static RAMs, being faster than dynamic RAMs, were selected for the random access memories. All of the static RAMs used in the Pattern Matcher have an access

time of 45 ns. In order to meet the 100 ns speed requirement, the Pattern Matcher is designed in a pipelined manner. The trigger data is latched in two successive stages. The output from the first latch is the input to the LEVEL 1 memories and the input to the second latch. The output from the second latch is the data which is used as part of the measurement sample. The trigger data from the first latch is re-coded through 2 levels of memory and the output of LEVEL 2 is latched. The output from this latch is the input to the LEVEL 3 memories. The output of the LEVEL 3 memories are the triggers. Therefore, when a trigger is generated at the output of LEVEL 3, the data which caused that trigger is simultaneously available at the output of the second trigger data latch. Additional pipelining between LEVELS 1 and 2 could speed up the Pattern Matcher from 100 ns to 50 ns per data point if necessary.

3.3.1 PATTERN DOWNLOAD.

The three memory levels of the Pattern Matcher are mapped into the address space of the analysis computer. By memory writes across a VME bus, see Figure 2, the analysis computer can load a new pattern set into the Pattern Matcher. Each of the SUs is assigned a 256K byte (or 64K long word) address block within the VME address space. The analysis computer loads a new pattern set by writing to each longword address in the assigned address block of that Pattern Matcher. Since long words are used (4 bytes) for the write, it is understood that both of the lower two address bits are zero.

Some addressing tricks are used to conserve VME address space. The three LEVEL 1 memories (a total of 24 data bits) are addressed simultaneously in parallel. The two LEVEL 2 memories along with the seven LEVEL 3 memories (a total of 23 data bits) are also addressed simultaneously in parallel. Also these two sets of memories (LEVEL 1 and LEVELs 2 and 3) share the same 265K byte (64K long word) address space. The distinction as to which is being addressed is determined by a bit in the 32 bit data word rather than the address itself.

The pattern set represents events on which measurement samples are to be acquired. Each bit position of a pattern may be true, false or don't care. Although any given pattern can represent a virtual address, physical address, or data, along with other signals of interest, for the sake of simplicity the following discussion will assume that it represents a virtual address. A programmer must select the set of events within the program to be measured. An event is normally the beginning or the end point of a section of code which is to be measured and is specified by its virtual address. The virtual address information may be obtained from the compiler or linking loader outputs. The set of virtual addresses must be converted to a set of patterns. A 24 bit virtual address is converted to a 32 bit pattern by adding the proper 3 bit bus qualifier specification and (currently) zeros for the additional 5 undefined bits. The resultant pattern set must then go through a re-coding process to generate the contents for LEVEL 1 and LEVEL 2 memories of the Pattern Matcher. Then the reduced re-coded patterns are placed in the LEVEL 3 memories for a table lookup search process. A program, outlined below, has been written which does the re-coding and then downloads the codes into the memory of the Pattern Matcher. This program requires the user to select the desired events within the test program, discern the corresponding virtual addresses, transform them into patterns and input these patterns to this re-coding and download program.

An automated approach would make this preliminary step much more convenient. A possible implementation would be to add compiler commands that the programmer would insert at program locations to be measured such that at compile or link edit time a separate file would be created containing the list of virtual addresses to be matched. This list could then be used directly by the re-coding and download program.

The re-coding procedure begins by checking for any state space overlaps between patterns. Two or more patterns will overlap only when *don't cares* occur. These overlaps must be eliminated so that the resultant pattern set is disjoint. The *don't cares* that are causing the overlaps are eliminated by expanding each one to its two possible values, zero and one, all other *don't cares* may remain. The expanded pattern set is then separated into three disjoint sets, corresponding to the three LEVEL 1 memories. Each element of a set need not be unique. The 11-bit elements lie within the range of 0 to 2047 but they sparsely populate this range. The re-coding proceeds by assigning to each unique element a new, unique, 7-bit value in the range of 1 to 127. As an example of this mapping take an input set {24,3,6,43,6,15}. Note that the value of 6 appears as two elements of this input set. The mapping would be:

24 -> 1
3 -> 2
6 -> 3
43 -> 4
15 -> 5.

The new set is {1,2,3,4,3,5}. The value of zero is used to map many of the unpopulated states of the original state space into this single, re-coded state. The original set is the address for the LEVEL 1 memories while the re-coded value is the memory contents at that address. The re-coded value for each original pattern is the concatenation of the corresponding values from all three sets, that is, the combined output of all three LEVEL 1 memories. The original 32-bit pattern is now re-coded and consists of 22 bits.

The pattern from the LEVEL 1 memories is re-coded again in a similar fashion for the LEVEL 2 memories. The 22 bit pattern set is separated into two disjoint groups of 11 bits each, corresponding to the two LEVEL 2 memories. Each element in the set does not have to be unique. Each 11-bit element lies within the range of 0 to 2047 (11 bits) but sparsely populates this space. The re-coding proceeds by assigning to each unique element a new 8-bit unique value in the range of 1 to 255. The value of zero is again used to map a number of unpopulated states of the input set into this single, re-coded state. The first re-coded value is the address of the LEVEL 2 memories while this second re-coded value is the memory contents at that address. The second re-coded value for each original pattern is the concatenation of the corresponding value from both sets, that is, the combined output of both LEVEL 2 memories. The original 32-bit pattern set has been reduced to 22-bits after the first re-coding and reduced again to 16-bits after the second re-coding.

Using this 16 bit re-coded representation for the original 32-bit pattern set, a pattern search can now be implemented by a single-memory look-up technique using a 64K x 1 bit memory. A value of one is assigned to each address of the 64K x 1 bit memory corresponding to a re-coded trigger event pattern. A value of zero is assigned to all other addresses. When the re-coding procedure is completed, the values are

downloaded into the Pattern Matcher memories by writing into their memory mapped locations.

3.4 PREPROCESSOR

The predominant function of a preprocessor is to acquire resource utilization performance information. One fundamental type of measurement for resource utilization is accumulated counts. The main items being counted, which may occur at a rapid rate, are event occurrences and time. Some examples of typical resource utilization performance metrics that require such measurement counts are cache hit ratios, access latencies, duty cycles and elapsed time.

Although the pattern matcher could be programmed to detect and trigger on each occurrence of these events, doing so would incur intolerable data collection and analysis problems. The first problem would occur when the collection of each of the individual occurrences floods the sample memory. The second problem would occur when the analysis program has to sum each of the individual occurrences. Thus, the preprocessors, by accumulating these counts rather than reporting each individual occurrence, relieve the pattern matcher from taking these samples and filling the sample memory with large volumes of unprocessed data.

We currently have two classes of preprocessors. One type accumulates a single independent count whose value must be maintained and reported in full precision. An example of this type of preprocessor is a timestamp generator, which accumulates counts of time ticks in 32 bit precision and reports the current full 32 bit value each time a trigger occurs. The current clock rate is 4 MHz, one tick every 250 ns.

The other type of preprocessor is used to count pairs of related events whose values must be maintained in full precision, but need not be reported in full precision. An example of this type of preprocessor is a ratio counter, which accumulates a pair of counts to measure, for example, cache hit ratios. In this example, both counters accumulate 32-bit-precision counts; one of the counters accumulates memory accesses, while the other accumulates cache hits. Only a single value, the ratio, is of interest and its accuracy does not require the full 32 bit precision. Since division is a very time consuming operation, the decision was made to report both numbers rather than the ratio of the two. The division is done during postprocessing by the analysis programs. The ratio is reported as two floating point numbers with separate, truncated mantissas and a common exponent. The total number of bits required to represent this pair is dependent on the accuracy required of the ratio. We expect a 5% error to be acceptable. This requires a total of 16 bits (6 bit numerator, 5 bit denominator, and 5 bit exponent) reported for a pair each time a trigger occurs, thereby allowing two independent pairs for each 32 bit preprocessor sample output.

There are four types of measurements which the ratio counter Preprocessor can perform: duty cycle, average duration, ratio of events and cumulative count. For the purposes of this discussion, time is the cumulative count of some regularly occurring signal, such as a CPU clock or bus cycle clock. Duty cycle is the time a signal is

active divided by the total time. Duty cycle, for example, may be used to measure bus utilization which is the number of bus cycles the bus is busy divided by the total number of bus cycles. Average duration, such as average bus access latency, is the number of occurrences of an event divided by the cumulative time that event is active. Ratio of events is the quotient formed by a subset of events of type A divided by the entire set of events of type A. For example, hit ratio is the number of access hits divided by the number of accesses. Cumulative count is the total number of times an event has occurred. Each of these measurements can be taken over a specific segment of the test program if a Selective Trigger is generated to reset the counters of the Preprocessor at the beginning of the segment. The next section discusses the design of the ratio counter preprocessor since, unlike the timestamp generator, it is a non-standard concept.

3.4.1 RATIO COUNTERS.

The design of the ratio counter Preprocessor began with two major constraints, operational speed and sample size. The operational speed of the ratio counters had to be at least 10 MHz, 100 ns per count, and the data output of the Preprocessor was limited to 32 bits. The events being counted could occur at a rapid rate and 32 bit counters were required in order to accumulate counts over any reasonable experimental duration. Initial investigations centered on techniques to accumulate the ratio on the fly, performing the division as each new count occurred, but no practical techniques were found to accomplish this running ratio. Some effort was then expended in devising ways to maintain the running counts and perform the division in the Preprocessor only when a sample was requested. This was possible but it was deemed infeasible due to the complexities (cost and real estate) involved. The remaining design choice was to output both counts of the pair which totaled 64 bits. Thus a method was needed to condense the 64 bits into 32 bits. The only possible way to accomplish this condensation was to truncate the reported values and thus reduce the precision. Resource utilization measurements are normally reported as a percentage. For our purposes, we expect an accuracy of 5% to be sufficient. As will be shown later, the method of truncation which has been selected maintains a worst case accuracy within our acceptable limits.

The representation of the output that was selected is a variation on floating point representation. The pair of numbers are related so they can share a common exponent. The data precision of 32 bits requires a maximum exponent of 5 bits. These pair of numbers represent a ratio whose value will always be non-negative and less than or equal to one. For our purposes the mantissas can be truncated to 6 bits each, yielding a range of 0 to 63. In this format, denominator values between 0 and 63 are untruncated and therefore exact. Since the numerator value is always less than the denominator, it too would be untruncated and exact. Once the denominator value exceeds 63, its reported mantissa value is always truncated and left justified, therefore, its range is between 32 and 63. The numerator mantissa is also truncated and always less than or equal to the denominator mantissa. The worst case measurement error is 1/D, where D is the truncated denominator count and the minimum value of D is 32. This yields a maximum possible error of 3.125%, which is within our tolerable limits.

A pair of self normalizing counters was designed which accumulate two counts of

32 bits each. The output is the two related values represented as two truncated mantissas with a common exponent. A normalizing counter pair consists of three registers and a sliding incrementer, see Figure 4. The three registers are the 32 bit denominator register, the 32 bit numerator register, and the 5 bit scale factor register which contains the common exponent value. The numerator and denominator registers do not use a standard structure in which the least significant bit is fixed in the right-most bit position of the register and the most significant bit moves from right to left when a high order carry occurs. Instead, the denominator register uses a structure which keeps the most significant bit in the leftmost position of the register and the least significant bit moves from left to right by a shift right when a high order carry occurs. The numerator is always rightshifted along with the denominator and, therefore, they may share a common exponent. A sliding incrementer is necessary to support this register structure, since the position of the least significant bit is no longer fixed but moves one bit position to the right each time a high order carry occurs. Thus, on each high order carry out of the denominator register, the denominator register, the numerator register, and the sliding incrementer are all rightshifted and the scale factor (exponent) register is incremented. As a result of this design the ratio counter output is always directly available without the need of additional manipulation. The exponent is in the scale factor register and the mantissas are always in the 6 most significant bit positions of the numerator and denominator registers.

This technique allows the original 64 bits (the pair of numbers) to be represented in a total of 17 bits (6 denominator bits, 6 numerator bits, and 5 exponent bits). This is further reduced to 16 bits by not outputting the most significant bit of the denominator, since it is always set (when the exponent is non-zero). A ratio counter preprocessor output sample therefore consists of 16 bits, and, due to this condensation, two independent sets of ratio counters may be placed in a single preprocessor, for a total of 32 output bits and a maximum error of 3.125% of full scale. Higher precision ratios may be acquired by using higher precision mantissas. This limits the output of the preprocessor to a single ratio pair in the 32 bit sample. For example, 14 bit precision in the mantissas requires 32 bits (14 bit numerator, 13 bit denominator and 5 bit exponent) and yields a maximum error of 0.013% of full scale.

The measurements taken with the Preprocessors are of more interest if taken during specific periods of an experiment rather than over the entire duration of the experiment. In order to correlate resource utilization with segments of the test program, the Pattern Matchers generate *selective triggers* which are connected to individual Preprocessors through switches that are set prior to each experiment. The switches are used to select which of the selective triggers will be used to reset which pair of ratio counters to zero at the start of each measurement period. Like the global trigger from each SU that are all tied together in a logical OR, the corresponding selective triggers from each SU are similarly tied together in a logical OR. Thus, a global trigger occurring at any SU will cause a measurement sample to be taken and stored in the sample memory at every SU. Similarly, a selective trigger occurring at any SU will cause the counters of the Preprocessor attached to that selective trigger to be reset after a measurement sample, if any, is taken.

3.5 SAMPLE MEMORY

Each 64 bit SU measurement sample, which consists of the output from the Pattern Matcher and the Preprocessor, is stored in a large Sample Memory. This memory is capable of storing up to 256K measurement samples (2M bytes). The Sample Memory is also directly accessible by the analysis computer which extracts the samples for data reduction, storage, and analysis. Measurement samples can be generated at a rate of 10 MHz (100 ns apart), but the Sample Memory is not fast enough to accept data at this rate and it also may be busy servicing an access from the analysis computer. Therefore a FIFO is used to buffer the measurement samples to allow for the time delays necessary to write it into the Sample Memory. The global trigger causes a measurement sample to be taken. This sample is immediately shifted into the FIFO. The measurement sample stays there until the Sample Memory is available. It is then shifted out of the FIFO and into the Sample Memory. The Sample Memory word is only 32 bits wide. Therefore the two 32 bit words of the sample (32 bits each from the Pattern Matcher and the Preprocessor) are written sequentially into the Sample Memory under control of a state sequencer, see Figure 5.

The Sample Memory is dual ported, thus allowing simultaneous connection and interleaved access to both the SU and the analysis computer, see Figure 2. The analysis computer currently in use has a VME bus interface. Therefore, in order to allow easy interface between the Sample Memory and the analysis computer, one port of the Sample Memory was fixed as a VME bus interface. The readily available memories offer the option of the other bus as either VMX or VMX-32 (a variation of VMX). VMX-32 was selected because it is more efficient for our purpose. As can be seen in Figure 2, there is a separate VMX-32 bus segment for each SU, but only one VME bus connecting all of the Sample Memories and the SUs to the analysis computer. The VME bus is the path by which the pattern sets are downloaded from the analysis computer to the Pattern Matcher and the path by which the analysis computer reads both the measurement samples from the Sample Memory and the sample memory address counter, which indicates the number of samples taken.

A complete measurement sample is 384 bits wide, see Figure 6, which consists of 64 bits from each of the SUs. Each of the SUs' samples consists of 32 bits of trigger data and 32 bits of preprocessor data which are stored in successive addresses in the sample memory. Note that the 32 bits of preprocessor data from SU #6 is the timestamp for a complete measurement sample.

4. LIMITATIONS

In general a hardware measurement approach for shared memory multiprocessors is attractive because it is non-intrusive and can provide more detailed measurement information than either software or hybrid approaches, albeit at a relatively high cost.

However, the general applicability of a hardware measurement tool like the REMS to any processor is limited because of three classes of problems: processor architecture, signal accessibility and physical size of the measurement tool. Architectural problems deal with the way the software handles multiple processes and the way the hardware internally handles the data and instruction streams. Accessibility problems have to do with the way the measurement tool is integrated into or attached to the processor to be tested. Most of these problems have solutions but many must be resolved during the design phase of both the processor and the VLSI integrated circuits.

4.1 ACCESSIBILITY PROBLEMS

All hardware measurement tools must connect to various signals of the system under test. For any useful tool this is really two requirements. The first requirement is to have available the signals to be measured. Without these signals, measurement is not possible. The second is to have some means of connecting to these signals. Without a convenient means of connection, such as a plug or connector, measurement is limited to highly knowledgeable and skilled individuals who have access to the internal design of the machine, which may involve proprietary information.

4.1.1 SIGNAL ACCESS.

Among other things, hardware measurement tools need access to the virtual address signals in order to capture trace measurement information. Since all program locations referred to by the compiler, link editor, and debugger use the virtual address, physical addresses are, in general, too difficult to relate back to the program under test. Also the physical address of a program is not known prior to run time and in many environments is dynamic and may change a number of times during execution. Some of the newer VLSI microprocessor chips have the memory management unit and/or both data and instruction caches designed into the chip. Some examples of this trend include National Semiconductor's 32532, Motorola's 68030 and Intel's 80386. This means that neither the virtual address nor the cache "hit" and "miss" signals may be available for triggering or counting purposes. Even worse, a number of accesses (instructions and data) may occur that are satisfied by the on-board caches and will never be observed by the hardware measurement tool.

The best solution to this problem is for the chip designers and manufacturers to design-in the *hooks*, so that, as an option, these signals could be available for measurement purposes. The chips could be designed with a socket on top into which the measurement tool could plug. This socket would provide the virtual address and necessary timing signals. This may not be economically feasible, since there is significant price competition at the chip manufacturing level. Possibly an optional special purpose chip may be offered at additional cost to those interested in performance measurement, although the anticipated limited production run of these special chips may still prove uneconomical.

An alternative solution, which reduces the need for cooperation by the

manufacturers and chip designers, is to augment the strictly hardware approach of the REMS tool with some of the hybrid concepts used in the TRAMS [MIN86, MIN87]. Specifically, the use of some embedded code to trigger the REMS eliminates the need to access the virtual address. The cost, of course, would be some perturbation to the program under test.

This is only a partial solution, since resource utilization signals such as cache access, cache "hit", etc. would still be inaccessible. Using an on-chip generated cache "hit" or "miss" signal as an event input to the ratio counter preprocessor is the obvious way to measure cache hit ratios. It might be possible to track the state of the CPU and infer from this state information when a cache miss occurred, but this is not a trivial solution to the problem.

4.1.2 SIGNAL CONNECTIONS.

For any useful tool, other than a laboratory version, a convenient means to connect to the signals is necessary, such as a plug or connector. Without such a connector, measurement is limited to highly knowledgeable and skilled individuals who have access to the internal design of the machine which might involve proprietary information. All of the processors' signal lines are available at a connector on each CPU board of our current testbed multiprocessor. The REMS probes are designed with a matching plug which makes the physical and electrical connectivity a relatively simple matter. Only a few other signal lines are required, to which "flying" leads are attached.

This connector is not available on other manufacturers' CPU boards and on newer versions of the current testbed. Without this connector two choices remain: use *flying leads* for connection to each signal line or use a daughter board. The use of *flying leads* requires very detailed knowledge of the board design. The daughter board, on the other hand, connects into the socket on the CPU board where the processor is normally inserted. This daughter board has a socket for the processor so that normal CPU operation can be maintained and, in addition, would bring out the required signal lines of the processor. In both cases significant spacing is required between adjacent CPU boards. This may not be available and a CPU board may have to be put on an extender board. This has the negative effect of lengthening the signal propagation path causing delays in some signals and may change some critical timing, thereby introducing perturbations and possibly errors.

Manufacturers could lay out their CPU boards with measurement in mind and make a measurement connection available on the board. This could be in the form of a socket or possibly an edge board connector. This may not be feasible for two reasons, electrical and financial. The addition of an extra connector along with the additional signal path length may introduce unacceptable propagation delays and reflections (even if terminators are used). In addition, this may not be economically feasible since it would increase the cost of each board. Possibly an optional special purpose board may be offered at additional cost to those interested in performance measurement, although the anticipated limited production run of these special boards may still prove uneconomical. Certainly without a reasonable user base demonstrating demand, no manufacturer would even consider such a product.

4.2 ARCHITECTURAL PROBLEMS

Architectural problems have to do with the way the architecture functions. Solutions to these problems, when possible, require significant design modifications by both the manufacturer and chip designers. Due to current VLSI circuit technology, it is virtually impossible to provide access to every logic signal of a computer. It would be completely impractical for any manufacturer or chip designer to even consider making this access available. So there will be operations internal to CPU chips that affect the accuracy of measurement and yet will never be observable. In other cases the operation is observable, but it is impossible under normal performance circumstances to distinguish between a number of possible source processes, resulting in inaccuracy of measurement.

4.2.1 INTERNAL OPERATIONS.

Similar to the signal access problem in which signals internal to a chip are not available, hidden internal operations can erroneously indicate the occurrence of an event. Look-ahead and pipelining are used on the instruction stream in order to keep the instruction queue full. Instruction prefetch is used to implement this look-ahead. In some cases, data prefetch is used for similar reasons. This could cause inaccurate measurement in two ways. First, by detecting the address of data or of an instruction during prefetch, the time that will be sampled and recorded will be the time of the prefetch and not the time the event is actually used. In most cases the time skew will not be significant, although when very fine grain measurements are attempted these few cycles may be significant. Second, since prefetch is only a statistical anticipation of potential use and not an assured use, there is always the possibility that, although fetched, the instruction or data will not be used. But the measurement tool will erroneously log the fetch operation as if the event had occurred. The error introduced by this false trigger is much more serious than the introduction of a slight time skew as in the first error type.

There is no feasible internal redesign that will alleviate this problem. Additional complexity can be added to the measurement tool to monitor for sequences. But this in itself is at best a half solution. One would have to determine the maximal length sequence necessary to definitely detect an event occurrence for each architecture, since their prefetch instruction queue sizes vary. Furthermore, there may not be a maximal length sequence for data prefetch. The resulting complexity of such a sequence pattern matcher is far too large to be feasible. An alternative would be to modify REMS into a hybrid tool and allow software triggering similar to that used by TRAMS.

4.2.2 PROCESS DISTINCTION.

When acquiring trace measurement information, virtual addresses are the predominant signals on which to trigger. When there are multiple processes executing it is impossible to determine which process accessed the virtual address that caused the trigger. This is bad enough if the only processes executing on the machine are part of

the program under test, but will give totally erroneous results if there are other processes executing that are unrelated to the program under test.

There are two possible solutions, both of which involve modification to the REMS measurement tool and to the system software. The system software to be modified is the scheduler within the operating system. A system call to the operating system could be provided that would be inserted into each process to be measured. The effect of this supervisor call would be to have the scheduler of the operating system output a signal to the measurement tool each time a process to be measured is scheduled for execution. One solution would be for the scheduler to control a dedicated signal line. This signal would change state and would cause REMS to turn on and off all pertinent circuits. The other solution, which is preferred since it would identify the process, would have the scheduler write to a dedicated location. The data for this write would contain the process number identifying one of the processes of the program under test and zero otherwise. This is similar to the approaches used in the EGPA [FRO83, HER82] experimental pyramid multiprocessor and by Hughes [HUG80] in the Diamond analyzer. Both of these approaches would require modifications to the Pattern Matcher and the Preprocessor so they would become inactive when the scheduler output is zero. Again, both these solution cause REMS to become a hybrid measurement tool, since some program perturbation occurs due to the embedded code in the operating system.

4.3 PHYSICAL SIZE LIMITATIONS

As the number of CPUs under test increases, the number of SUs also increases thereby expanding the physical size of REMS and the enclosure necessary to house it. The current SU implementation consists of three circuit boards, each plugging into a VME slot in a card cage. A single card cage can house six SUs of the current implementation. An increase of one order of magnitude in the current multiprocessor size, from 6 processors to 60, would require 10 card cages to house the REMS.

The size of an SU can possibly be condensed to a single circuit board using VLSI. If this were the case then manufacturers would only have to offer their machines with a double sized enclosure. That is one having twice as many slots as processors so that a measurement board could be located next to each processor. But once again we return to the problem of connection between the measurement system and the CPU board under test.

5. SUMMARY

The designs of the Trace Measurement System (TRAMS) and of the Resource

Measurement System (REMS) prototype have been presented as examples of instrumentation approaches for a shared-memory multiprocessor. TRAMS is a hardware assisted extension to a traditional software approach to obtain multiprocessor trace measurements. Manufacturers could offer this type of hybrid tool as a plug-in option. REMS is a non-intrusive hardware measurement tool for MIMD, tightly coupled, shared memory multiprocessor architectures which is able to obtain both trace measurement and resource utilization measurements. Along with the design description of REMS, some of the design decisions and trade-offs have also been discussed. Also some of the limitations of using measurement tools like REMS on these architectures have been described, as have some potential solutions to overcome these limitations. If manufacturers would give more consideration to performance measurement tools in their product designs, even as an extra cost option, many of these limitations would be alleviated.

6. ACKNOWLEDGEMENT

The Authors wish to thank R. J. Carpenter for his extensive editorial comments in the preparation of this report and for raising some of the issues discussed herein.

7. REFERENCES

[CAR87] R. Carpenter, *Performance Measurement Instrumentation for Multiprocessor Computers*, NBSIR in preparation, National Bureau of Standards, Gaithersburg, MD, 1987.

[FRO83] H. Fromm, et. al., *Experiences with Performance Measurement and Modeling of a Processor Array*, IEEE Trans. on Computers, Vol. C-32, No. 1, Jan 83, pp 15-31.

[HER82] U. Hercksen, R. Klar, W. Kleinoder, F. Kneibl, *Measuring Simultaneous Events in a Multiprocessor System*, Proc. of 1982 ACM SIGMETRICS Conf. on Measurement and Modelling of Computer Systems, Aug. 1982, Seattle, Wash., pp 77-87.

[HUG80] J. Hughes, *Diamond: A Digital Analyzer And Monitoring Device*, Perf. Evaluation Review, Vol. 9, No. 2, 7th Intern'l Symp. on Computer Performance Modelling, Measurement, and Evaluation, Toronto, Ontario, Canada, May 1980, pp 27-34.

[MIN86] A. Mink, J. Roberts, J. Draper, and R. Carpenter, *Simple Multi-Processor Performance Measurement Techniques and Examples of Their Use*, NBSIR 86-3416, National Bureau of Standards, Gaithersburg, MD, July 1986.

[MIN87] A. Mink, J. Draper, J. Roberts, and R. Carpenter, *Hardware Assisted Multiprocessor Performance Measurements*, NBSIR 87-3585, National Bureau of Standards, Gaithersburg, MD, June 1987.

[ROB85] J. Roberts, *Performance Measurement Techniques for Multi-Processor Computers*, NBSIR 85-3296, National Bureau of Standards, Gaithersburg, MD, Feb 1986.

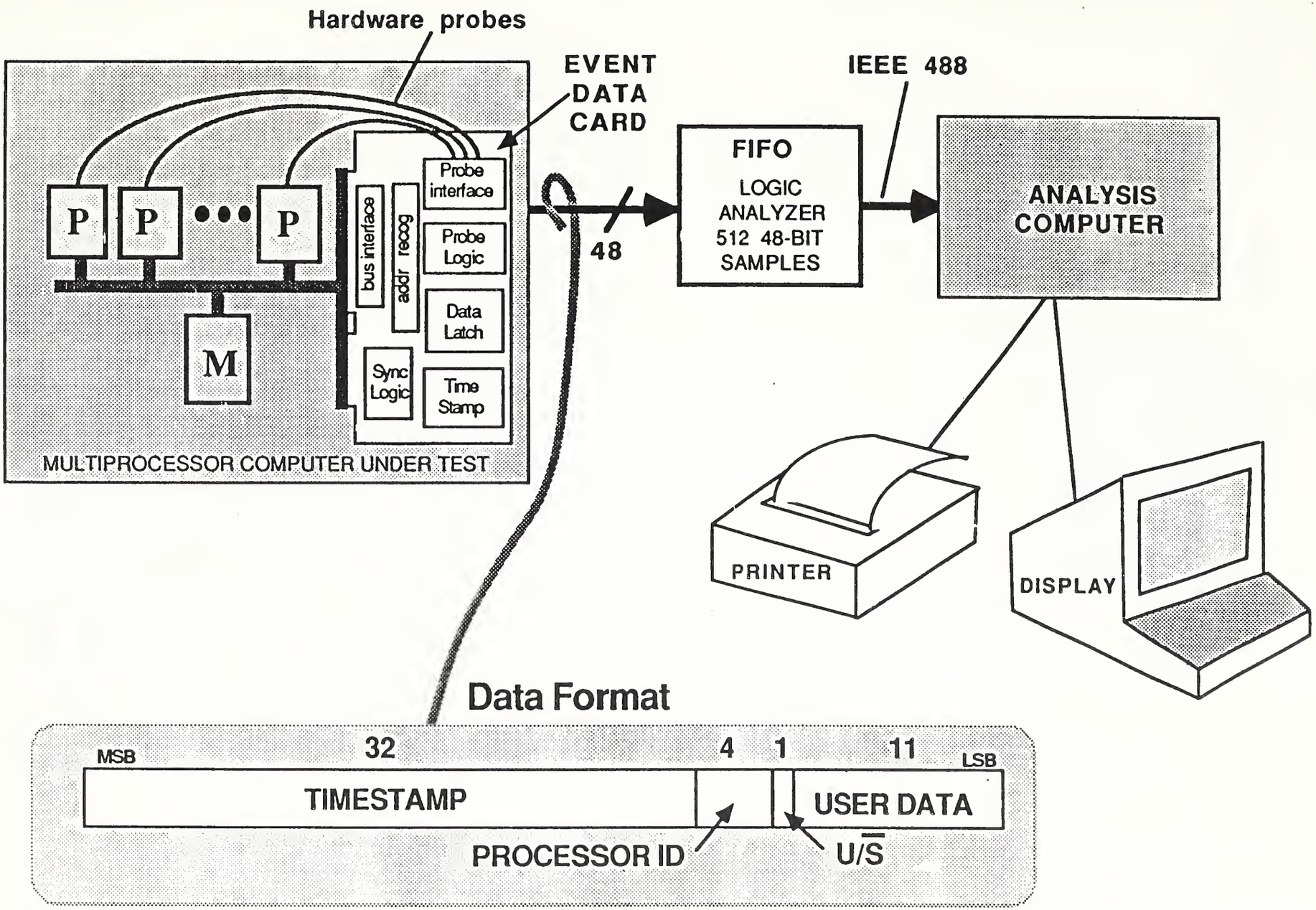

Figure 1 - TRAMS - Trace Measurement System

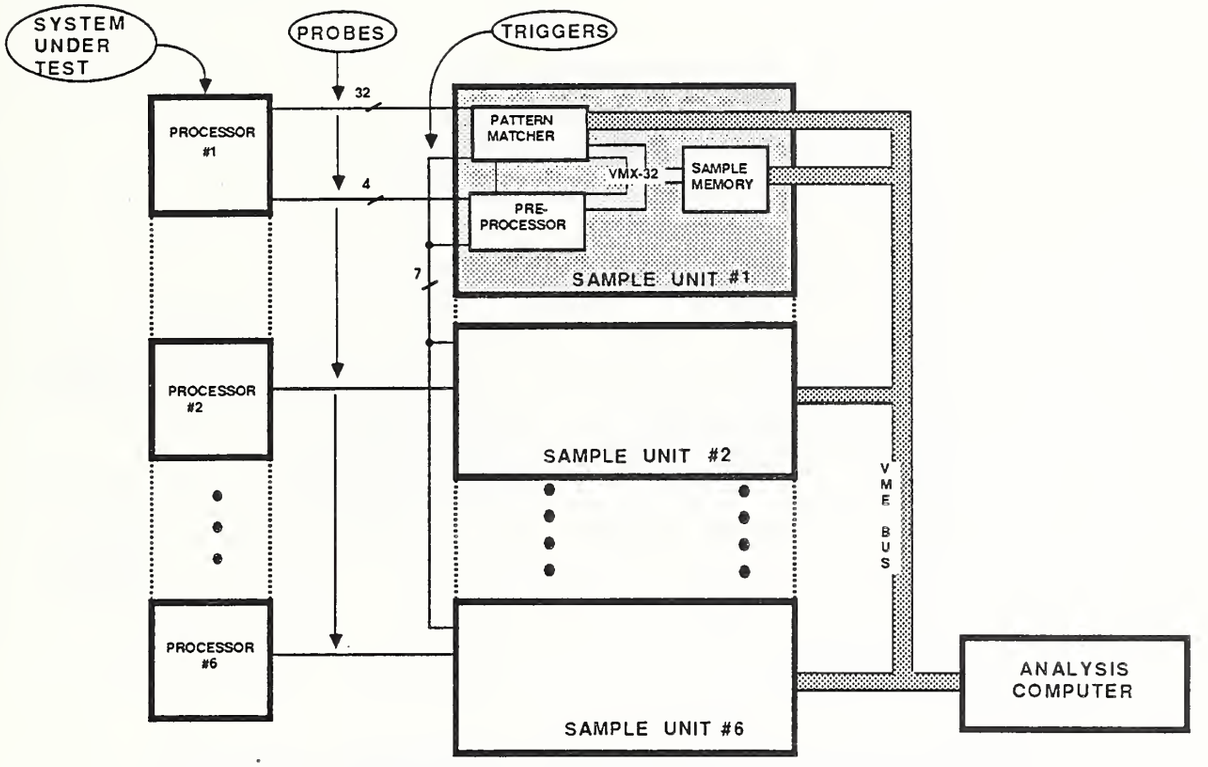

FIGURE 2 - Resource Monitoring System (REMS)

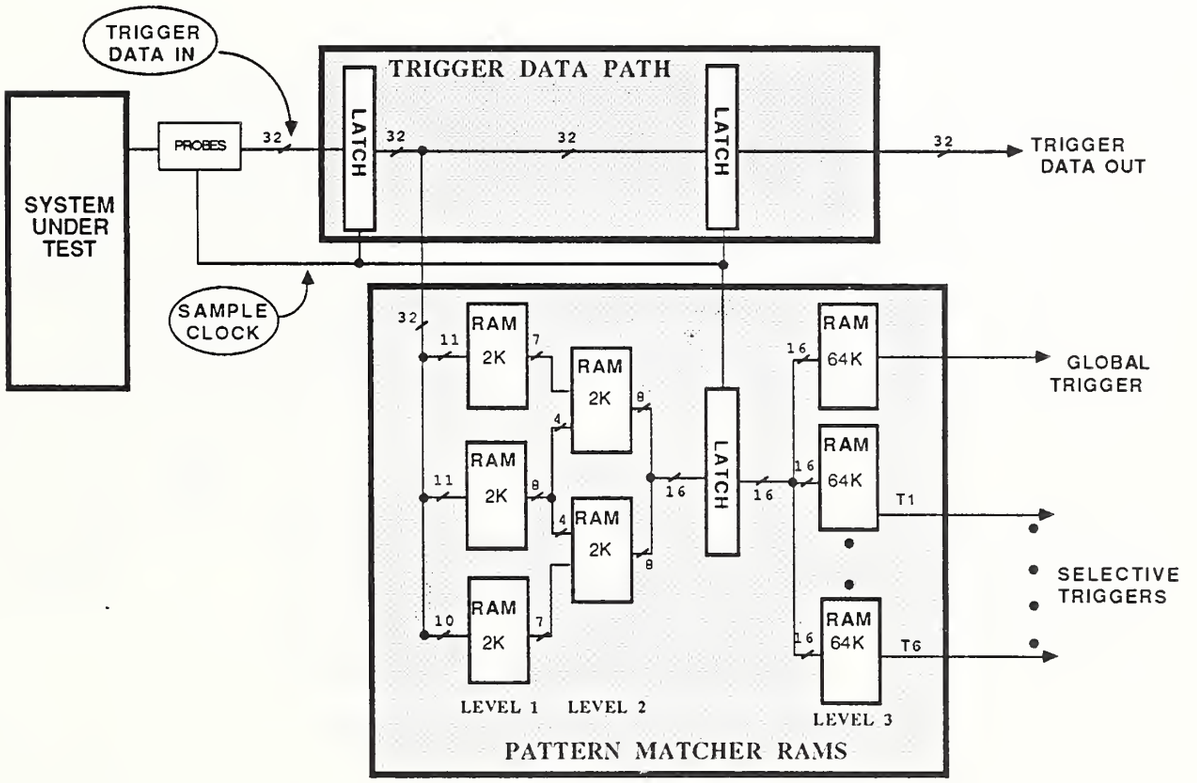

FIGURE 3 - Pattern Matcher Memories and Data Latches (REMS)

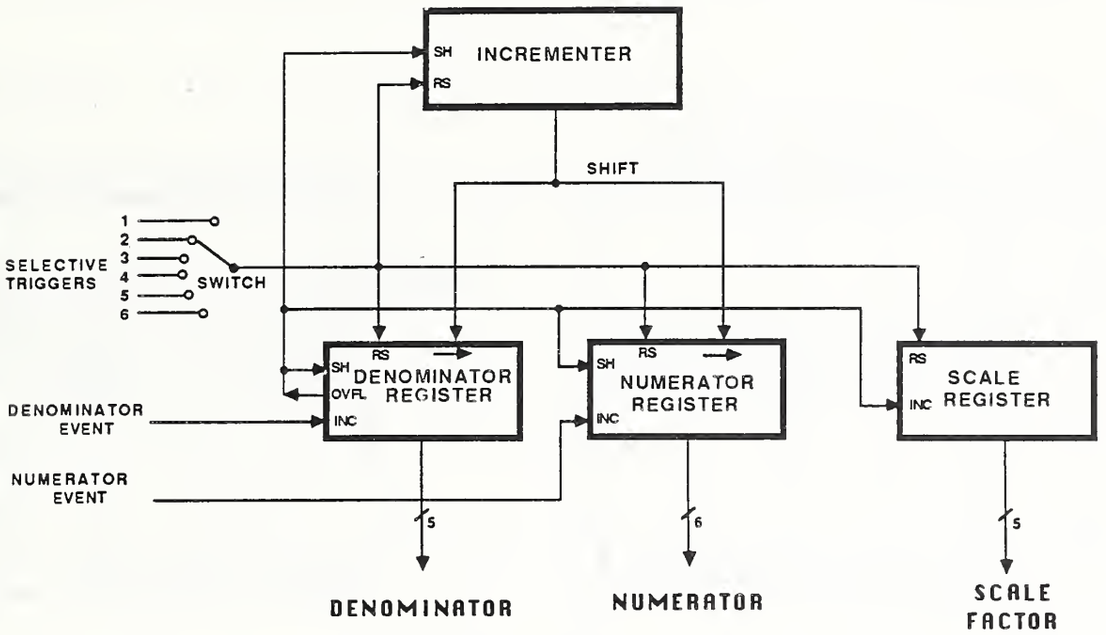

FIGURE 4 - Normalizing Counter Pair (Ratio Counter)

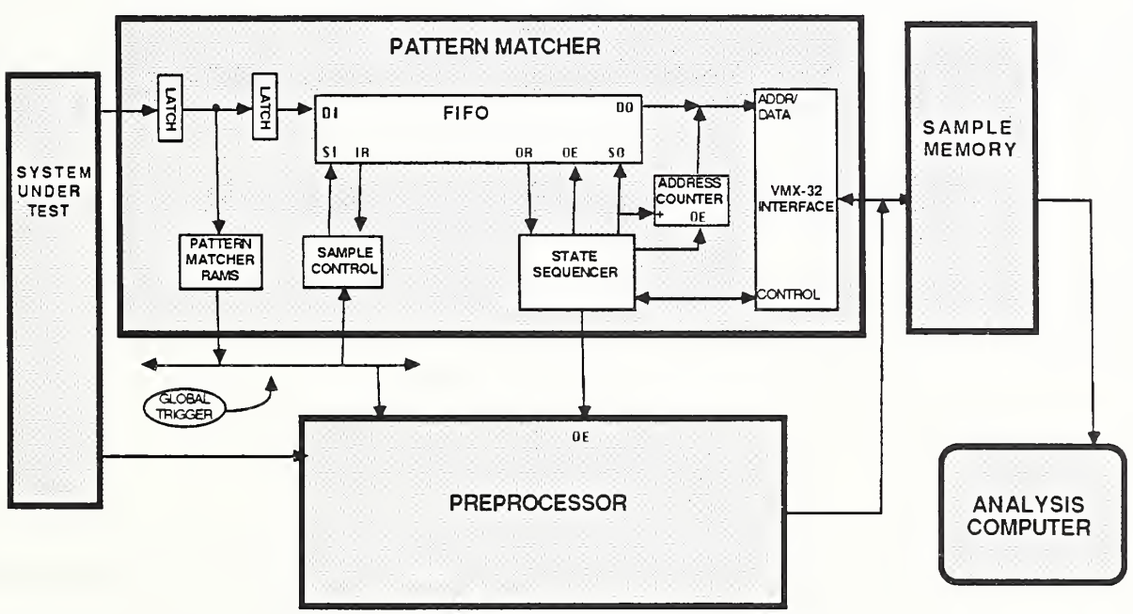

FIGURE 5 - Data Path (REMS)

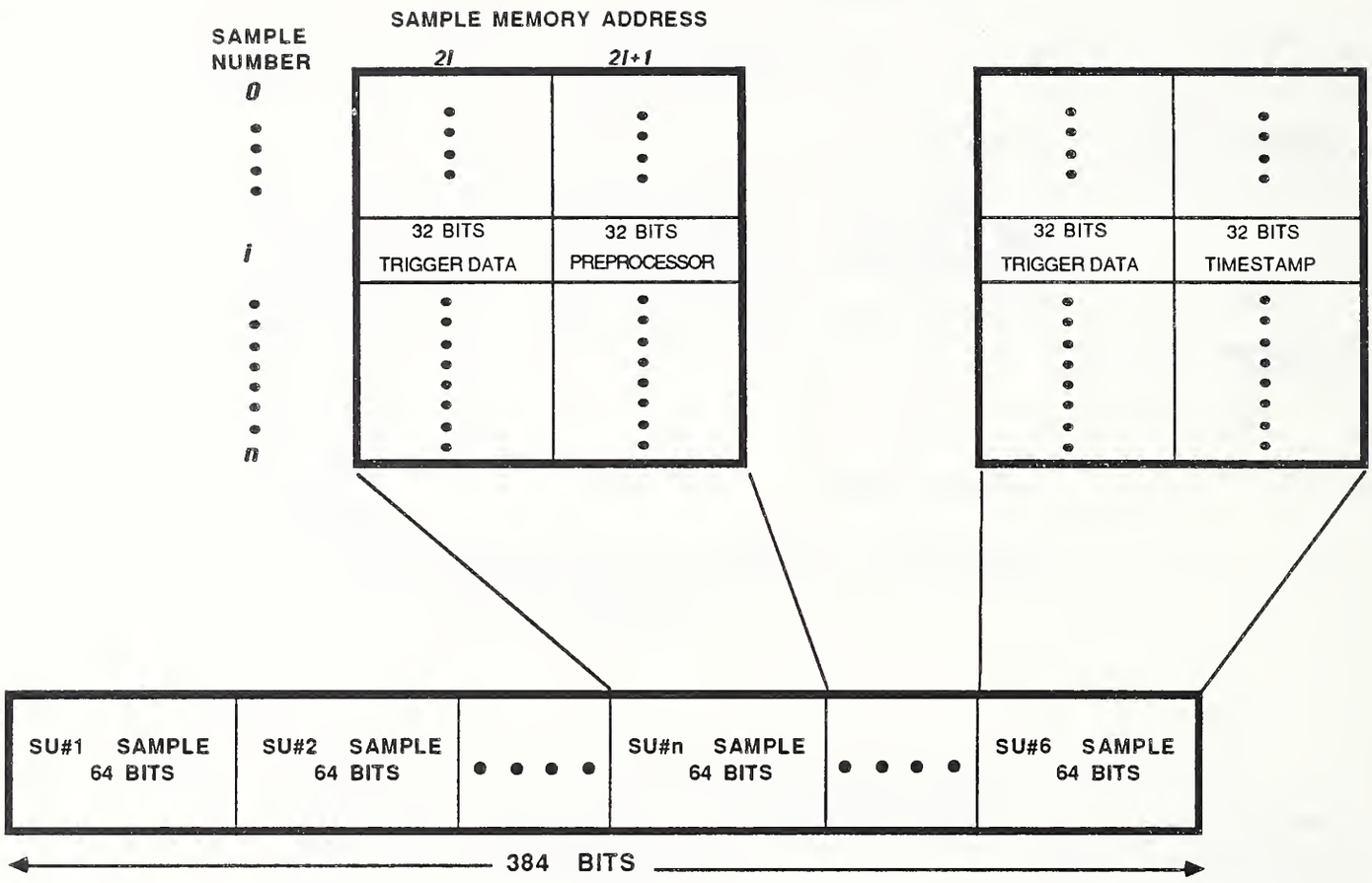

FIGURE 6 - Sample Word Format

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 87-3663	2. Performing Organ. Report No.	3. Publication Date OCTOBER 1987

4. TITLE AND SUBTITLE

Recommended Instrumentation Approaches for a Shared-Memory Multiprocessor

5. AUTHOR(S)

George Nacht, Alan Mink

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Two approaches for the design of performance measurement instrumentation for a shared memory, tightly coupled, MIMD multiprocessor are presented. The TRAce Measurement System (TRAMS) is a hybrid measurement tool used to obtain trace measurement information. The Resource Measurement System (REMS) is a non-intrusive hardware measurement tool used to obtain both trace measurement and resource utilization information. The TRAMS approach provides a hardware assist to the more traditional software approach of obtaining timestamps from the operating system at each event to be measured. This hardware assist reduces the artifact that is introduced in a test program and is a feasible and economical approach to providing measurement capabilities to a wide range of multiprocessors. Manufacturers could offer this type of measurement tool as a plug-in option. The REMS approach provides more detailed and extensive measurement information than does the TRAMS approach and introduces no artifact to the test program, but it does this at a significantly higher cost. When access to pertinent signals is restricted the applicability of such a hardware tool is limited.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

computer performance; multiprocessor computers; parallel computers; performance measurement

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

28

15. Price

$11.95

NBSIR 87-3663
Jun 13, 2016