Inter-Office Memorandum

To	Mesa Users	Date	October 27, 1980
From	Jim Sandman	Location	Palo Alto
Subject	**Performance Measurement Tool**	Organization	SDD/SS/Mesa

XEROX

Filed on: [Iris]<Mesa>Doc>PerformanceTool.bravo (and .press)

A tool for the performance measurement of Mesa programs is described below. It allows users to identify places in their programs and then collect timing and frequency statistics of program execution between these places. The system is implemented as a set of commands that can be executed from the Mesa Debugger, plus a routine that intercepts all conditional breakpoints and collects statistics about them. Existing Debugger commands are used to specify what points are to be monitored, and additional commands are provided for controlling the measurements and outputting the results.

Concepts

A *node* is defined to be a point in a program where a breakpoint can be set by the Mesa Debugger. In fact, nodes are implemented via conditional breakpoints, so that while monitoring is turned on, the *functioning of all conditional breakpoints is different*. In particular, conditional breakpoints cause performance data to be gathered rather than a breakpoint to be taken.

A *leg* is defined by a pair of nodes, one called the *from* node and the other the *to* node. A leg is the code executed between these nodes. Interesting items measured about a leg include the number of times this leg was executed and the time required to execute the leg.

Facilities are also provided for associating a *histogram* with any node or leg, thereby providing more detailed distribution information about the entry than is provided by counts, sums, and averages.

Since *processor time* or *task time* is not available on the Alto, the measure of computing is simply the *elapsed time* between the time the *from* node is executed and the time the *to* node is executed.

The concept of nodes and legs is borrowed from the Diamond Extended Test Module. The performance tool was first written by Paul Jalics, and has since been extended by the Mesa Group.

Terminology

Node Table

A table maintained by the measurement module containing information about each node. A node for each conditional breakpoint is entered into this table by the `Collect nodes` command or by the measurement module when it encounters a conditional breakpoint that is not already in the table. The node table has 20 entries.

NodeID

The index of a node in the node table, used in commands to identify a particular node. This is the same as the breakpoint number assigned by the Debugger.

Leg Table

A table maintained by the measurement module containing various information about each leg. Legs are entered into this table by the command `Add Legs` or by the measurement module when it encounters a new leg and automatic insertion is enabled. The leg table has 41 entries, one of which is reserved.

LegID

The index of a leg in the leg table. The LegID for a particular conditional breakpoint does not change during a measurement session and is used in commands to identify a particular leg.

Histogram

An optional table that may be associated with either a node or leg that records the distribution of a variable associated with the node or leg by incrementing counters in a number of *buckets*. The distribution may be either *linear* or *logarithmic*. In a linear distribuiton, a *base* may be specified which will be used as the offset for the first bucket. In a logarithmic distribution, the buckets are indexed by the number of leading binary zeros in the *value*. A *scale* is used to adjust the value for an optimal fit into the number of buckets. There is a storage pool of 256 words that is shared among all histograms to hold buckets and histogram information.

Node Histogram

A histogram associated with a node. The histogram variable of the node is the first variable in the conditional expression attached to the breakpoint that defines the node. The value is treated as a 16 bit unsigned quantity. For a simple node histogram, the value is adjusted by subtracting the base (if any) and dividing by the scale factor; the resulting quotient is recorded. A logarithmic node histogram has a maximum of 16 buckets because the value is a 16 bit quantity.

Leg Histogram

A histogram associated with a leg. The histogram variable of the leg is the 32 bit time of the leg in units of ticks. The value is adjusted by shifting the value to the right by the scale. A logarithmic leg histogram has a maximum of 32 buckets because the value is a 32 bit quantity.

Components

PerfTool is the component of the measurement system that is loaded with client programs built on top of Alto/Mesa. This configuration contains two modules: **PerfMonitor** and **PerfBreakHandler**. **PerfMonitor** initializes the **PerfTool**. **PerfBreakHandler** contains a breakpoint handler that intercepts all conditional breakpoints and accumulates statistical information about nodes and legs. **PerfTool** must be loaded and started in the system it will monitor. This may be done by including **PerfTool** in the client configuration whose control module starts **PerfDefs.PerfMonitor** or by executing the following command in the Alto Executive:

```
>Mesa PerfTool Client
```

PerfPackage is the component that is loaded as a UserProc into the Debugger. It implements the basic commands required to manipulate the node table and the leg table and to output measurement results. **PerfPackage** must be loaded into the Debugger before its commands can be executed. It also needs the **UserProc** package. The easiest way is to load it when installing the Debugger by executing the following command in the Alto Executive:

```
>XDebug PerfPackage/l
```

The **PerfPackage** creates a window through which all interaction with the tool takes place.

Operation

When the break handler intercepts a breakpoint, it checks to see if the breakpoint is conditional. If so, it finds the node corresponding to the breakpoint, increments its counters, and processes its histogram if one exists. If tracking of legs is enabled, the leg table is searched for the legs of which this node is a part. Otherwise, the breakpoint is resumed.

In the simple case, a leg is tracked as follows. The break handler intercepts a conditional breakpoint that is the *from* node of the leg (from) and some time later it intercepts a conditional breakpoint that is the *to* node of the leg (to). At this point, the leg's time is recorded, its count is incremented, and its histogram (if any) is processed.

This simple model of tracking a leg is complicated by recursion, signals, and multiple processes. With recursion, from may be encountered several times before to is encountered. With signals, a process may be unwound after it encounters from but before it encounters to. With multiple processes, one process may encounter from and then another immediately encounter to.

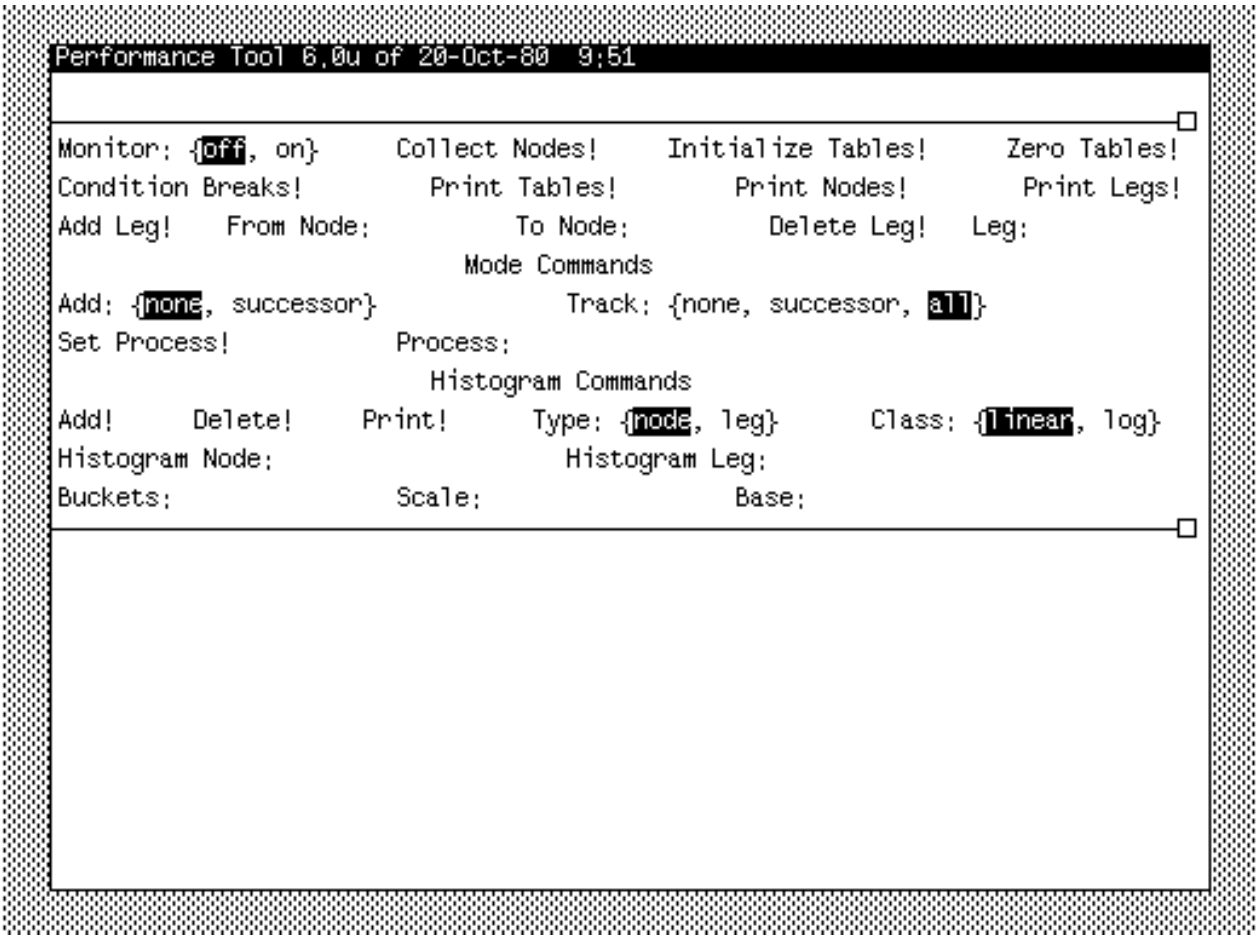
To deal with the complication of multiple processes, there is the concept of the *tracked process*. If the tracked process is not NIL then only those conditional breakpoints that are encountered by the tracked process are treated as nodes. All others are simply resumed as if they did not exist. If the tracked process is NIL, then all processes are tracked.

To deal with these complications, there is a *leg owner*. A leg owner is the process that last encountered from. When to is encountered and the current process is its owner, then the leg is recorded and the leg owner is cleared. If the current process is not the owner, the leg is ignored. As a result of ignoring legs, from and to may be counted more times than the leg between them is counted.

Normally, when a node is encountered all legs of which that node is a part are tracked. Alternatively only the leg defined by the last node encountered and the current node is tracked.

Window and Commands

Interaction with the **PerfPackage** is done through its window. There are three subwindows, the message subwindow, the parameter subwindow, and the log subwindow. Error messages and warnings are displayed in the message subwindow. Commands are invoked in the parameter subwindow. All output is displayed in the log subwindow. An illustration of the window during a sample session is shown below.



General Commands

`Monitor: {off, on}`

turns off/on performance monitoring. All conditional breakpoints will be monitored when the monitor is on, and will behave as normal conditional breakpoints with it is off.

`Collect Nodes!`

enters all currently existing conditional breakpoints as nodes in the node table.

`Initialize Tables!`

completely reinitializes all tables and counters. The node table, the leg table, and all histograms are cleared.

`Zero Tables!`

zeros out all counts and sums from the tables (including the total time spent measuring) but leaves all other information in the tables unchanged. This command is useful for preserving the measurement environment but just zeroing out the counts and sums collected so far.

`Condition Breaks!`

makes all non-conditional breakpoints into conditional breakpoints by adding the condition "1" to them.

`Print Tables!`

displays all the summary statistics gathered so far and the complete contents of the node table and the leg table. This command may be aborted by typing **^DEL**.

`Print Nodes!`

displays the contents of the node table. A NodeID followed by an asterisk has a histogram associated with it. This command may be aborted by typing **^DEL**.

`Print Legs!`

displays the contents of the leg table. A `LegID` followed by an asterisk has a histogram associated with it. This command may be aborted by typing **^DEL**.

`Add Leg!`

adds the leg specified by `From Node` and `To Node` to the leg table. If a designated leg entry is already in the leg table, the leg is not affected.

`From Node:`

contains the `NodeID` of the *from* node for the `Add Leg` command. The character "*" may be used as a wildcard.

`To Node:`

contains the `NodeID` of the *to* node for the `Add Leg` command. The character "*" may be used as a wildcard.

`Delete Leg!`

deletes the specified leg from the leg table.

`Leg:`

contains the `LegID` used by the `Delete Leg` command.

Mode Commands

`Add: {none, successor}`

if set to `none`, prevents the **PerfBreakHandler** from adding legs that are not in the table. This is the default mode for automatically adding legs. If set to `successor`, the **PerfBreakHandler** adds legs that are not in the table. These legs may be deleted if there is no room in the leg table when legs are added by the `Add Legs` command.

`Track: {none, successor, all}`

if set to `none`, the **PerfBreakHandler** disables tracking of legs. If set to `successor`, the **PerfBreakHandler** tracks only the leg defined by the last node encountered and the current node. If set to `all`, the **PerfBreakHandler** tracks all legs in the table. This is the default mode for tracking legs.

`Set Process!`

tells the **PerfBreakHandler** to track only those legs that are executed by the process specified by `Process`. Nodes encountered by other processes will not be recorded. An octal ProcessHandle as obtained from the Debugger's `List Processes` command is acceptable as input to this command. The default case is to track all processes.

`Process:`

used by the `Set Process` command. It contains an octal ProcessHandle as obtained from the Debugger's `List Processes` command. If `Process` is empty, all processes are tracked.

Histogram Commands

`Add!`

adds a histogram and associates it with either `Histogram Node` or `Histogram Leg`, depending on the value of `Type`. The command gets its parameters from the `Class`, `Buckets`, `Scale`, and `Base` fields.

`Delete!`

deletes the histogram associated with the specified node or leg.

`Print!`

displays the histogram associated with the specified node or leg. This command may be aborted by typing **^DEL**.

`Type: {node, leg}`

if set to `node`, the above histogram commands operate on the histogram associated with the node specified by `Histogram Node`. If set to `leg`, the above commands operate on the histogram associated with the leg specified by `Histogram Leg`.

`Class: {linear, log}`

used to specify the kind of distribution of the histogram to the `Add` command.

`Histogram Node:`

contains a `NodeID` for specifing a node to the `Add`, `Delete`, and `Print` commands.

`Histogram Leg:`

contains a `LegID` for specifing a leg to the `Add`, `Delete`, and `Print` commands.

`Buckets:`

used to specify the number of buckets to the `Add` command.

`Scale:`

used to specify the scale of the histogram to the `Add` command. Note that since scaling of a leg histogram is done by shifting instead of dividing, the scale is entered as a power of two.

`Base:`

used to specify to the `Add` command the base of the distribution of values for linear histograms.

Limitations

1. Time base: The time base available on the Alto is a 26-bit counter, where the basic unit of time is 38 microseconds. Thus the counter turns over every 40 minutes, and no individual time greater than 40 minutes is meaningful on the Alto. Total times are 32-bit numbers and will overflow after 340 minutes.

2. Overhead calculation: Due to implementation restrictions and timer granularity, some of the overhead of processing a breakpoint is incorrectly assigned to the client program instead of the **PerfTool**. As a result, leg times will be about ten microseconds high for each node that was enountered while processing that leg. Elapsed time is similarily affected. This effect is particularly noticeable with short legs. Relative times between legs may give better information about program performance.

3. Counter sizes: In a long measurement session, the node, leg, or histogram counters may overflow. Node and leg counters are twenty-two bits, while histogram counters are sixteen bits. If a node or leg counter overflows, a "*" follows the count when the field is listed.

4. Recursive procedure calls, UNWINDs, multiple processes: As mentioned in the section on operation, the above interfere with the simple start to end concept of a leg. With recursion and multiple processes, the start node of a leg may be tripped several times before the end node is tripped. With unwinding, the start node of a leg may be tripped and the end node never reached. If any of these cause a leg to be ignored, the referenced field in the Leg Table has a "~" following it when the table is listed.

5. Table sizes: The node table contains 20 entries. (Note that the **PerfBreakHandler** automatically extends the number of conditional breakpoints that can be set in the debugger from five to 20.) The leg table currently has 40 entries. Note that this number is small when compared to the 20*20 possible legs. For this reason, there exist a number of commands to give the user control over exactly what legs are in the table.

6. Memory requirements: The **PerfTool** requires seven pages of resident memory; three for **PerfBreakHandler**'s code, and four for **PerfTool**'s frames. This may affect the performance of systems that use a lot of memory.

7. Worry mode: The **PerfBreakHandler** operates in worry mode; as a consequence, you may find that you cannot `Quit` from the Debugger after your session. Use the `Kill` command instead.

Getting Started

The steps required for using the measurement tool are outlined below.

1. obtain the `.bcd` files for **PerfTool** and **PerfPackage**.

2. install the **PerfPackage** in the Debugger.

3. start your program with the **PerfTool** included.

4. enter the Debugger and set conditional breakpoints as desired.

6. turn measurements on by setting the `Monitor` parameter to `on`.

7. collect nodes and manipulate the leg table as desired.

8. proceed with program execution.

9. return to the debugger via an interrupt or an unconditional breakpoint.

10. display results with the `Print` commands.

Sample Session

The following annotated listing of `Debug.log` and `Perf.log` should give a fair idea of the use of the measurement tool. It monitors the time required for the swapper to allocate real memory pages.

```
Alto/Mesa Debugger 6.0 of 25-Oct-80 22:50
26-Oct-80 14:01

You called?
>SEt Root configuration: Mesa
```
-- allocate most of memory to exercise the swapper
```
>SEt Module context: SegmentsB
> AllocatePages[140]
37400B^
```
-- set breakpoints to monitor the `AllocVM` procedure
```
>SEt Module context: Swapper
>Break Entry procedure: AllocVM Breakpoint #1.
>ATtach Condition #: 1, condition: pages = 1
```
-- a histogram is attached to this breakpoint and the variable `pages` will be counted.
```
>Break Xit procedure: AllocVM Breakpoint #2.
```
-- the `Condition Breaks` command is used to make this a conditional break.

```
Performance Tool 6.0 of 21-Oct-80 15:09
26-Oct-80 14:02

Conditionalized breaks
Collected nodes
Leg from 1 to 2 added
```

```
 - - - - - - N O D E   T A B L E   C O N T E N T S - - - - - - - - - - -
Node Global  Program  Number of  Config   Module
 Id  Frame   Counter  References Name     Name
---- ------  -------  ---------- -------- --------
  1  174000     2076          0  Mesa     Swapper
  2  174000     2521          0  Mesa     Swapper
Added Histogram for Node 1
Added Histogram for Leg 0
```

-- execute `Proceed`, followed by an interrupt to the Debugger.

```
Total Elapsed Time of Measurements =          69.627:210
Elapsed Time less PerfMonitor Overhead =      67.100:932
Total Overhead of PerfMonitor Breaks =         2.526:278
Total number of Perf Breaks handled =              1,394
Average Overhead per Perf Break =                  1:812
% of Total Time spent in PerfMonitor =              3.62

 - - - - - - N O D E   T A B L E   C O N T E N T S - - - - - - - - - - -
Node Global  Program  Number of  Config   Module
 Id  Frame   Counter  References Name     Name
---- ------  -------  ---------- -------- --------
  1* 174000     2076         697  Mesa     Swapper
  2  174000     2521         697  Mesa     Swapper

- - - - - - - - L E G    T A B L E    C O N T E N T S - - - - - - - - - -
Leg  From  To     # of Times   Total Time      Average Time   % of
Id   Node  Node   Referenced   sec.msec:usec   sec.msec:usec  Time
---  ----  ----   ----------   -------------   -------------  -----
  0*   1 ->  2          697      27.328:536          39:208   39.24


Histogram for Node 1
Number of References              697
Sum of Values                   2,454
Average Value                       3
Scale Factor                        1
Base                                0
     Value       Count
 -------------- -------
              0       0
              1       3
              2     190
              3      76
              4     297
              5     131
              6       0
              7       0
       Overflow       0


Histogram for Leg 0
Number of References              697
Sum of Values                 719,172
Average Value                   1,031
Scale Factor (2^n)                  0
     Value       Count
 -------------- -------
              1       0
```

2	0
4	0
8	0
16	0
32	34
64	99
128	1
256	0
512	1
1,024	562
2,048	0
Overflow	0