SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 21(5), 465–477 (MAY 1991)

NovAtel's Novel Approach to CPU Usage Measurement

DAVID R. CROWE
NovAtel Communications Ltd., 1020 64th Avenue NE, Calgary, Alberta, Canada, T2E 7V8

SUMMARY

This paper describes a set of CPU usage analysis tools, named CPU Probes, with the benefits of both hardware and software tools, and few of their disadvantages. CPU probes use a serial line unit or programmable interval timer as a clock to sample the CPU usage of a system, resulting in fast and accurate CPU usage estimates. CPU usage can be monitored for all processes simultaneously or as little as a single machine instruction. CPU probes are well suited for monitoring real-time system peformance. These tools have been implemented on PDP-11/RSX and UNIX System V/386 systems.

KEY WORDS CPU usage profilers Optimization Timing Real-time performance

INTRODUCTION

It seems to be a law that software systems expand to use all available CPU time as they are developed. Inevitably, selected parts need streamlining to meet performance requirements. Rapid and accurate identification of bottlenecks requires tools that can monitor the real-time performance of selected parts of the system without causing appreciable degradation.

Two types of CPU usage measurement tools are commonly available: hardware tools, such as logic analysers, and software tools, such as profilers. Hardware tools are of limited use on multi-tasking systems because of their expense and inability to interpret operating system data structures. Profilers [1–4] can provide detailed information about the CPU usage of functions within a single process. However, monitoring code must be inserted into the process being monitored. This degrades the performance of the process and requires it to be specially edited, compiled or linked. Profilers usually suspend execution of the monitored process completely while results are being output. Accounting packages [5] and process status monitors [6] can report the CPU usage of a process, but not the CPU usage of a portion of a process. Nor can they easily monitor the CPU usage of a process over an arbitrary period of time.

This paper describes the design and implementation of hybrid CPU usage measuring tools. NovAtel's 'CPU Probes' use hardware that is part of most general purpose computer systems, together with custom software that is executed separately from the process or processes being monitored. The CPU probes developed at NovAtel

0038–0644/91/050465–13$06.50

Received 10 July 1989
Revised 27 November 1990

were written in C and MACRO-11 under the RSX-11 M operating system on a PDP-11 computer. A serial line unit (SLU) [7] is used to generate ticks. The CPU probes have recently been ported to a UNIX system on an Intel 80386 computer (System V/386) with ticks being generated by an Intel 8254 programmable interval timer (PIT). [8] This version was written completely in C.

Comparison with profilers

The advantages of CPU probes over profilers are:

1. Processes do not need to be specially compiled or linked to be monitored.
2. Results can be displayed without suspending the monitored processes.
3. Monitoring can be started or stopped at any time.
4. Performance of a system being monitored is only slightly degraded.
5. Any part of the system, from all processes, a single process or repeated execution of a single instruction can be monitored.
6. Monitoring can be at a higher rate than the system clock.
7. Statistics are gathered independently from the system clock.

The disadvantages of CPU probes compared with profilers are

1. CPU probes do not provide information about where library functions are being called from.
2. CPU probes can only provide estimates of the number of times a function or line of code was executed, whereas profilers can insert counting code.
3. CPU probes may not give meaningful results for processes that change the code at some virtual addresses (e.g. overlaid tasks).

The first four advantages of CPU probes are critical to the monitoring of real-time systems. On such systems processes often run indefinitely and cannot be restarted or suspended during normal operation. The disadvantages indicate circumstances where profilers would be a more appropriate tool.

Accounting and process status monitors share some of the advantages of CPU probes (1 to 4), but do not provide the same detail and accuracy.

Comparison with hardware tools

The advantages of CPU probes over hardware tools are

1. CPU probes can access operating system data structures to determine useful information, such as the currently active process identification. Hardware tools are unable to do this.
2. CPU probes monitor activities in the virtual address space of a computer. Hardware tools can most easily monitor physical addresses, which will not be meaningful on a system with active paging or swapping.
3. CPU probes are not affected by the use of caches. Hardware tools may require caches to be turned off for monitoring to be meaningful.
4. CPU probes are not affected by the use of multiple memory buses. Hardware tools can only monitor the activities on one set of address lines at a time.
5. Hardware tools are expensive. CPU probes require no additional hardware, and only the development of a small amount of software.

The disadvantages of CPU probes over hardware tools are:

1. CPU probes cannot monitor interrupt service routines and other software that is executed with interrupts disabled.
2. CPU probes do degrade the performance of the system, if only slightly. Hardware tools may have no effect (unless, for example, caching has to be turned off).
3. CPU probes provide only an estimate of CPU usage, hardware tools can provide an exact figure.

The advantages of CPU probes make them more useful in virtual memory systems. Hardware tools may be more useful in simpler systems, particularly those without timer hardware suitable for CPU probes.

ARCHITECTURE

NovAtel has developed a unique distributed cellular switching system that addresses the needs of small to medium sized markets. The system consists of a Micro-Vax/VMS based master mobile centre (MMC) connected to one or more PDP-1l/RSX-llM-PIus or 80386/UNIX based switching mobile centres (SMCS), that each control several cell-sites. The cell-sites manage the radio interface to cellular phones. Each SMC manages the call processing functions for traffic in its cells and provides an interface to the wire-line telephone network. The MMC communicates with all SMCS, providing a comprehensive network maintenance and support system. The CPU probes described below were developed to ensure that the SMC met its real-time performance requirements.

The NovAtel CPU probe system (see Figure 1 for the RSX version, Figure 2 for the UNIX version) combines two existing hardware elements (numbered 1 and 2) with four custom software modules (numbered 3 to 6). The loop of elements 1 to 4 provides clocking and allows the system state to be sampled regularly by elements 5 and 6. The ‘single process monitor’ (element 5) monitors the CPU usage of part of a single process whereas the ‘system monitor’ (element 6) monitors the CPU usage of all processes. Accumulated information is formatted and output after monitoring is complete.

Tick generator

CPU probes must be activated at regular intervals to sample the system state. These intervals should not be correlated with ticks of the system clock. If they are, measurements may be biased for or against activities scheduled by operating system timers. Two types of devices that can provide such ticks are serial line units (SLUs) and programmable interval timers (PITs).

PITs, such as the Intel 8254, are ideal as tick generators, because the rate at which they interrupt is easily controlled. They can interrupt at a much higher rate than the system clock.

On systems without such a device, such as standard PDP-11 computers, an SLU can also be effective. An SLU can generate an interrupt after transmission of a byte, and can therefore interrupt at regular intervals. The interval will be determined by the bit rate of the SLU and number of bits required to transmit each byte. For

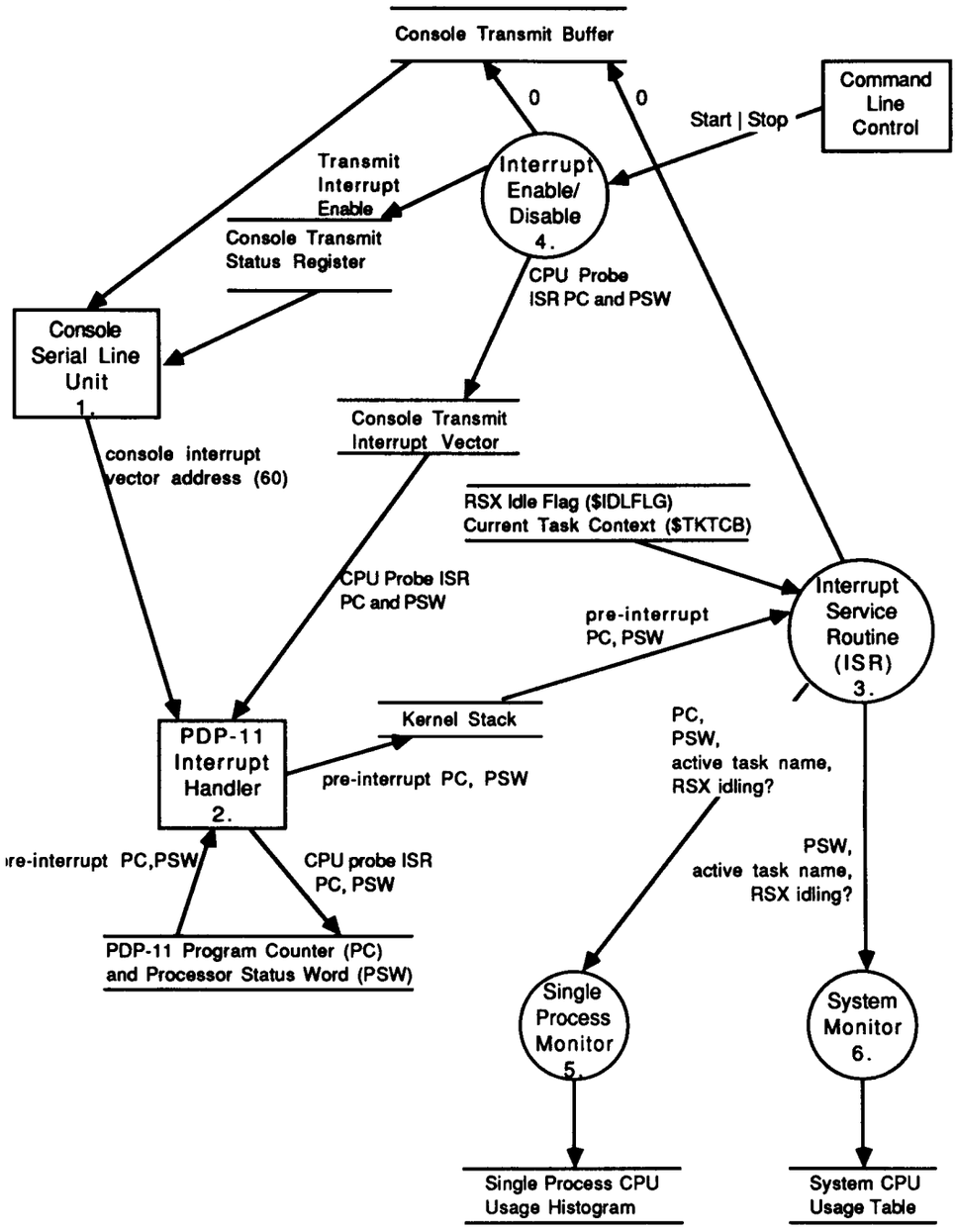

Figure 1. RSX CPU probe dataflow

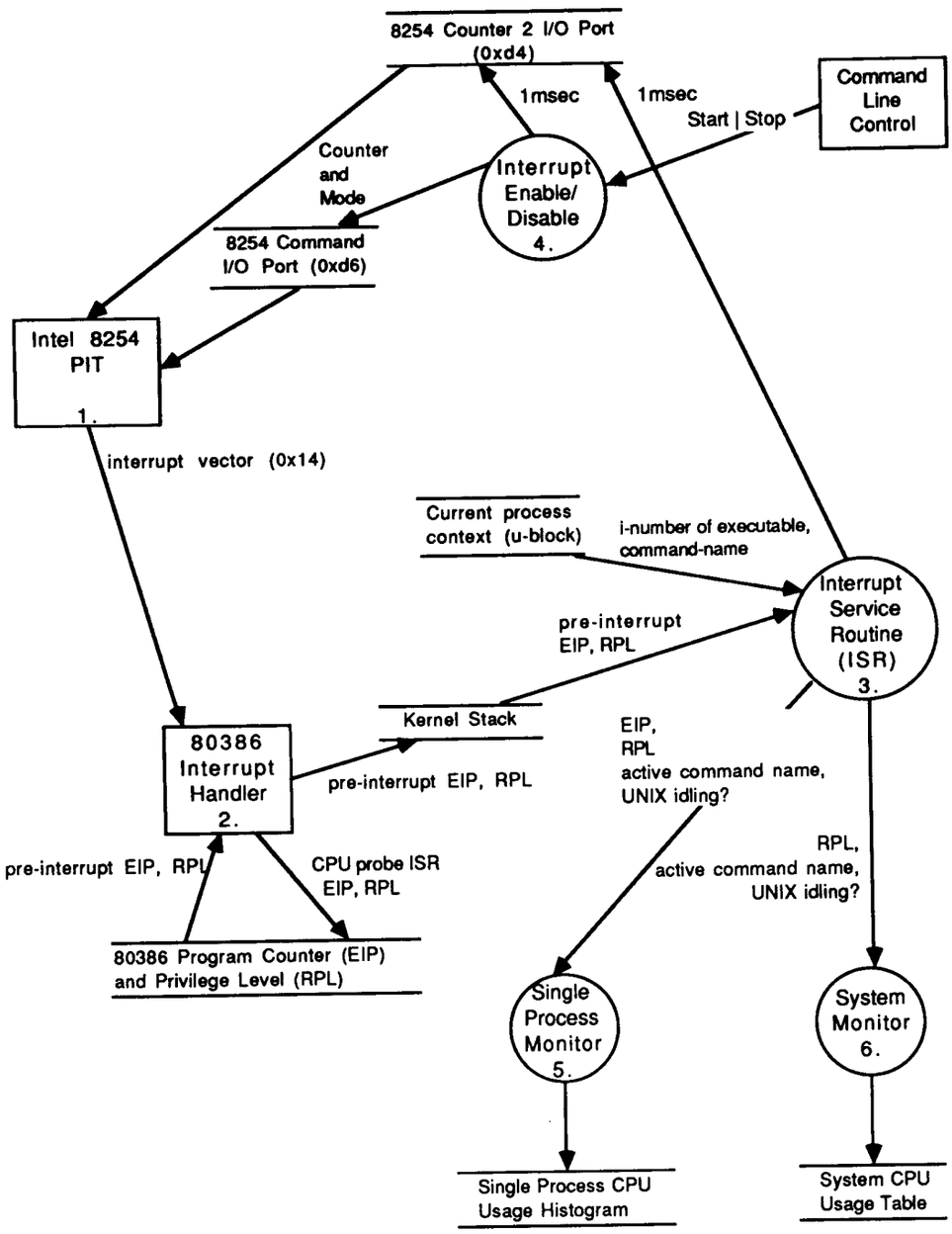

Figure 2. UNIX CPU probe dataflow

example, if an SLU transmits 9600 bits/s and each byte is transmitted as 10 bits (8 data bits and 1 start and stop bit) interrupts will occur at 960 Hz.

Collecting statistics at intervals uncorrelated with system clock ticks has proved to be important in practice. In one case, a CPU probe that used the system clock as a source of interrupts, reported that the system was almost completely idle when in fact it was using almost all available CPU time. The situation occurred because all processes were in deadlock, executing resource allocation code that woke up on every clock tick to determine whether a required resource was available.

Interrupt handling

A CPU probe interrupt service routine (ISR) has two major functions; to store information about the system state at the time of the interrupt and to arrange for another interrupt. The state information required is the currently active process, CPU mode (e.g. kernel or user) and whether the operating system is active or idle. For CPU probes of a single process, the program counter of the interrupted process is also required. All this information can be retrieved from the kernel stack, or other kernel data structures relating to the currently active process, on both RSX [9] (PDP-11) and UNIX [10] (80386) systems.

In order to arrange for the next interrupt, the device providing ticks may have to be reprogrammed. SLUs need to have a byte written to the transmit buffer to interrupt again. PITs may need to have the internal interval counter re-programmed. PITs can also be programmed to interrupt continuously; however this increases the probability of interrupt overload on a busy system.

A primary design objective of the NovAtel CPU probe ISRs was to minimize their effect on the system. We achieved this by collecting information that is easy to access and by optimizing the interrupt service routines carefully. We have not seen the system behave abnormally on PDP-11/73 CPUs, even with a CPU probe using an SLU transmitting at 9600 bits/s. At this rate, CPU probes use about 18 per cent of the CPU time (or 188 μs per interrupt). Similarly, on a 33 MHz 80386 CPU probes consume less than 5 per cent of the CPU time, while interrupting 1000 times per second. These estimates were derived by comparing the execution time of a CPU bound process with and without the CPU probes running.

Information collected by CPU probes would be completely accurate if devices could always interrupt the CPU. However, there are times when the priority of the CPU is raised, preventing interrupts from being serviced immediately. Because CPU probes use an ISR to monitor the system state, they cannot monitor other ISRs, including their own activities. CPU probes can, however, monitor other times when the operating system has raised the interrupt priority of the CPU. In these cases, the execution time at high priority will be assigned to the first instruction executed after the priority is lowered.

System-wide CPU usage monitoring

The system CPU probe monitors all processes in the system simultaneously. The CPU usage of each process, in each CPU mode, is displayed in a table (see Figure 3). This information is accumulated by sampling, on each interrupt, the CPU mode before the interrupt, the active process id and whether the operating system is idling.

```
>pat -sec 60

                    Percentage CPU Utilization
                     (7200 Samples, 8 missed)


Task Name     Kernel Mode    Supervisor Mode   User Mode   Total    +|-
=========     ===========    ===============   =========   ======   ====
 ...LDR              0.0             0.0          0.0       0.1    0.05
AM                   0.0             0.0          0.0       0.1    0.06
AMXM                 0.0             0.0          0.0       0.1    0,07
AU                   0.3             0.1          0.1       0.4    0.15
CAE                  0.8             0.0          0.1       0.9    0.22
CAN                  0.7             0.1          0.2       0.9    0.22
CN                   0.9             0.0          0.1       1.0    0.23
CP                   3.0             1.7          2.8       7.5    0.61
CRT                  0.3             0.1          0.1       0.4    0.15
CSIT1                1.2             0.4          1.5       3.1    0.40
CS2T1                0.5             0.2          0.7       1.4    0.27
CS3T1                2.5             0.5          3.4       6.5    0.57
CS4T1                1.3             0.3          1.3       2.9    0.39
CS5T1                1.4             0.4          1.4       3.2    0.40
CS6T1                1.7             0.4          1.7       3.7    0.44
ROGUE               35.4             0.0          7.8      43.2    1.14
DS                   1.8             0.3          1.8       3.9    0.45
DT                   0.5             0.2          0.4       1.2    0.25
F11ACP               0.0             0.0          0.0       0.0    0.05
FTSIDE               0.1             0.0          0.0       0.1    0.07
IDLE                 2.8             0.0          0.0       2.8    0.38
IN                   0.2             0.1          0.0       0.4    0.14
INSV1                0.2             0.0          0.0       0.2    0.11
JTCHK                0.1             0.0          0.0       0.1    0.07
JTRCV                1.3             0.3          0.2       1.9    0.31
JTSEND               4.9             1.8          0.9       7.5    0.61
LKM                  1.3             0.2          0.5       2.0    0.32
MCR. . .             0.0             0.0          0.0       0.0    0.04
OUTT1                0.4             0.0          0.0       0.4    0.14
PATV1                0.0             0.0          0.0       0.0    0.04
RUNV1                0.1             0.0          0.0       0.1    0.06
SM                   0.8             0.3          2.0       3.1    0.40
TCDLM                0.0             0.0          0.2       0.2    0.09
TM                   0.3                          0.1       0.4    0.15
XM                   0.2             0.0          0.2       0.4    0.14
---------           ----           -----        ----      -----
Total               64.6             7.6         27.7      99.9
```

Figure 3. Sample system wide CPU probe display

If the operating system is idling, the sample is assigned to an imaginary process named IDLE.

One design problem was that process ids are the logical index into the table of counters yet, at 32 bits, are not a convenient index to a table of reasonable size. An alternative would be to search the table, on each interrupt, for the process id of the active process, but this was thought to be too slow. As a compromise between simplicity and efficiency, process ids are hashed. Hashing uses an algorithm that

maps a value from a large range, pseudorandomly onto a small range. This small range maps a table containing somewhat more records than are expected to be required. This technique reduces the incidence of collisions, when several process ids hash onto the same record index.

When a collision does occur, the ISR examines the following records, looking for one that is empty or belongs to the currently active process. Up to five records are searched before the interrupt is assigned to a lost interrupts counter. Normally, on our systems, with 128 records, few interrupts are lost in this way.

The chosen hash algorithm exclusive-ors the four bytes of the process id together. It then clears the top bit to produce a value between 0 and 127. This algorithm is used because it is efficient, avoiding the use of expensive multiply and divide instructions, and works well in practice.

On RSX systems a ‘task name’ is used as a process id. A task name is from one to six letters taken from a restricted alphabet and compressed into four bytes. The six letters of the task name are used for display purposes.

On UNIX systems, the i-node of the executable image of the process is used as a process id. This assigns CPU usage for all invocations of a command to the same record, making it easier to monitor the CPU usage of commands that are used frequently but do not consume a large amount of CPU time on each invocation. The display shows the command name (up to 14 characters) which is the file name of the executable. The UNIX process id could be used instead.

Single process CPU usage monitoring

The second CPU probe developed by NovAtel monitors the CPU usage of all or part of a single process. The results are displayed as a histogram, graphically showing the regions of code that are consuming the most CPU time (see Figure 4). Each histogram bar also includes a 95 per cent confidence interval, shown as dashes around the estimated value (see the Appendix).

When this CPU probe is invoked, the process name, address range and CPU mode to monitor, as well as how long to monitor, are specified. All interrupts are counted, but detailed information is only kept for interrupts that occur when the specified process is executing in the desired CPU mode between the given addresses. If an interrupt passes muster, a counter in one of 512 records is incremented corresponding to the portion of the address range that the process is executing in.

To increase the efficiency of the CPU probe ISR, the address and range associated with each record are adjusted up to the nearest power of two. Several variables are calculated once, based on this range, to reduce the work done on each interrupt.

By varying the address range, this CPU probe can monitor the entire address space of a process, or as little as 1 kilobyte (with each record monitoring only 2 bytes). This CPU probe can automatically repeat its monitoring with continually smaller address limits, until an address range with CPU usage in both the highest and lowest bucket is found.

The histogram produced by this probe can be enhanced by a program that merges it with function names from the software being monitored (see Figure 5). This allows easier assignment of CPU usage to functions and comparison of different versions of a program that may have the same function at different addresses. The symbol names and addresses are extracted from the symbol table produced by the linker.

```
>plt  -see 180 rogue

Task RCGUE was active in user mode for 1860 of 21598 samples (9%).


Statistics starting at address 0000000, slot size 0200

Address   Hits
=======   ====
 .  .  .
002200:     23: *-|-
 .  .  .
015200:     88:  ********--|--
 .  .  .
020000:     24:  *-|-
020200:     77:  *******--|--
020400:     72:  ******--|--
020600:     78:  *******--|--
021000:     69: ******--|--
 .  .  .
024200 :    71:  ******--|--
024400:     59:  *****-|-
024600:     67:  ******-|-
025000:     67:  ******-|-
025200:    286:  ********************************----|----
025400:     13: *|
 .  .  .
032000 :    37: ***-|-
032200:    109:  ***********--|--
032400:     12: |
 .  .  .
033200:    120:  ************--|--
033400:    119:  ************--|--
033600:     25: *-|-
 .  .  .
042400:    442:  ************************************************-----|-----
 .  .  .
```

Figure 4. Sample single process CPU probe display

AN EXAMPLE

The best way to seethe power of CPU probes is through an example. Figures 3,4 and 5 show successive steps in isolating and identifying a CPU usage problem. The ‘problem’ was created as an illustration, but is similar to several real problems that have been encountered.

The first step in the analysis is to run the system-wide CPU probe (Figure 3). It clearly identifies that one process, named ROGUE, is consuming far more CPU time than any other.

The next step is to identify the parts of the process that should be investigated further. Figure 4 shows a probe of the entire ROGUE process in user mode. A user mode probe was chosen because it is easiest to identify problems with and, in our experience, CPU usage improvements in user mode usually result in kernel mode improvements. It is clear that the code between addresses 025200_8 and 025400_8 and between 042400_8 and 042600_8 deserves further attention.

The last step is to identify the functions within the process, or even individual machine instructions, most responsible for the CPU usage. This step may have to be repeated if several processes, or several address ranges within one process, are being investigated. To avoid having to match CPU probe slot addresses with function addresses, a symbolic CPU probe is used.

Figure 5 clearly shows the functions in the ROGUE process consuming the most CPU time. A further, more detailed, CPU probe would be needed to separate the CPU usage of the functions c$lmul, c$sav, c$ret and c$rets. However, in this case, knowledge of the system being probed makes that unnecessary. The function dbwrit is known to call all but one of the other functions. Examination of the source code for the ROGUE process reveals that it executes an infinite loop calling dbwrit on each iteration. CPU usage is not being consumed because the functions listed in Figure 5 are inefficient, but merely because they are being invoked so often.

BENEFITS

Rapid identification and optimization

Time and money can be saved by applying limited resources to optimizing software that is consuming a significant fraction of the total CPU time. CPU probes can quickly identify areas of most concern because they monitor the CPU usage of the entire system, or progressively smaller parts of a single process.

In one example, a system CPU probe showed that one process was consuming 25 per cent of the CPU time of the SMC whenever it ran, which was every few minutes. A single process CPU probe isolated the function responsible. Further analysis revealed that it only required execution occasionally and a simple, fast check could determine when. A more detailed probe revealed that much of the CPU usage was due to bit manipulation inside loops. Simple rearrangement of the code, guided by repeated probing, further reduced the CPU usage. After these changes the CPU usage of the process dropped from 25 to 1 per cent of the total.

Improved software quality

An unexpected benefit has been that higher quality code is written when CPU probes are available. The emphasis during initial programming is on writing clear and correct code. The use of programming tricks to optimize code is avoided since bottlenecks can be quickly identified and eliminated later. This has resulted in faster development of more efficient and maintainable code.

Accurate time measurement

The approximate execution time of a particular machine instruction, function or system call can be calculated by counting the number of times it executes during probing; assuming the number and rate of CPU probe interrupts are known.

For example, to estimate the execution time of an emulated 32 bit integer division on a 16 bit PDP-11, a program could be written to execute 10,000 of these operations. A detailed CPU probe could be used to separate the time spent performing the divisions from other system activities, including looping overhead. If 3500 interrupts

```
>mps  -sec 360 -low 025200 -high 042600 rogue  rogue.stb

Task ROGUE was active in user mode for 2717 of 43198 samples (6%).

Statistics starting at address 0025200, slot size 020

Address   Hits
=======   ====
024230-dbwrit
025200:     17: |
025220:     27: *|
025240:  29:  *|

024230-dbwrit  025276-dbmap
025260:     86: *****-|-

025276-dbmap
025300:    164:  **********-|-
025320:     93: *****-|-
025340:    194:  ***********-|  -.

025362-dbunma
025360:  85:  *****-|-
025400:  40:  **1

032142-cpybuf
032140:  47:  **-|-
032160:     17: |
032200:  25:  *|
032260:  26:  *|
032300:    141:  *********-|-
032320:  22:  *|
032440:  44:  *-|-

033354-re|sp
033340:    186:  **********--|--
033360:     35: **|
033400:  23:  *|
033420:  41:  **|
033440:  71:  ***-|-

033354-relsp  033472-szlkp
033460:  55:  **-|-

033472-szlkp
033500:  27:  *1
033520:  47:  **-|-
033600:  54:  **-|-

042300-c$lmul  042410-c$sav
042400:    113:  *******-|-

042410-c$sav  042426-c$ret  042436-c$rets
042420:    222:  **********  --|--

042436-c$rets
042440:  57:  **-|-

042462-emt
042460:    724:  ************************************************----|----
```

Figure 5. Sample symbolic CPU probe display

occurred while the system was executing division operations, and 960 interrupts occurred each second (9600 bits/s SLU transmitting 10 bits/byte), it can be estimated that each division consumes about 0·36 ms of CPU time (3500 × 1000)/(960 × 10000).

CONCLUSIONS

CPU probes, as their name suggests, do not solve CPU usage problems, they merely identify them. Like medical tests, they are not a cure. But by identifying the disease, they allow software doctors to concentrate on finding a cure.

CPU probes have provided NovAtel with the expected advantages. No costly measurement hardware is required because hardware that is usually present on general purpose computers is used. Probes can monitor any part of the system from all processes to single instructions because the monitoring code is implemented as a device driver, not as part of the processes being monitored. Processes can be monitored without being specially compiled or linked for the same reason. The performance of the system is only slightly degraded by the CPU probes, because all information is accumulated by incrementing counters based on readily available system state information. Care has been taken to ensure that counters can be accessed efficiently. Displaying the results does not interfere with the rest of the system because it is done by a normal process.

The CPU probes have also provided NovAtel with an unexpected benefit. Developers now write code more straightforwardly at first, knowing that areas requiring optimization can be easily determined during testing, rather than guessed at during coding.

CPU probes may also help extend the lifetime of a CPU bound system. The CPU usage of the system can be monitored regularly. If CPU probes indicate that CPU usage has risen to unacceptable levels, probes can then be used to implement the necessary reduction in CPU usage.

We have found CPU probes to be invaluable tools while a complex system with stringent performance constraints is being developed. They enable developers to focus on the most CPU intensive software, so improvements can be made in the overall performance of the system most effectively. The effort required to develop these tools was repaid quickly.

ACKNOWLEDGEMENTS

I would like to thank Jim Houston for initially coming up with the idea of using a serial line unit to implement CPU probes. Monty Ghitter contributed to its implementation and read and commented on a draft. William Gage, Dr. Arunas Slekys, Dr. Ian Davis, Roger Ingles and two anonymous referees also provided valuable comments.

APPENDIX: CPU PROBE CONFIDENCE INTERVAL

n samples are collected *(x_i: $0 \leq i < n$).* The *i* th sample (x_i) is 1 if the object being monitored is active, 0 if not. Let the number of active samples (x_i=1) be S1. The sample mean can be calculated as

$$m = \sum_{i=0}^{n-1} x_i/n = \mathrm{S1}/n$$

The sample variance can be calculated as

$$\begin{aligned} s^2 &= \left[\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2/n\right]/(n-1) \\ &= (\mathrm{S1} - \mathrm{S1}^2/n)/(n-1) \\ &= (\mathrm{S1} - \mathrm{Sl}\, m)/(n-1) \end{aligned}$$

A 95 per cent confidence interval for large samples (e.g. $n > 50$) can be calculated as [11]

$$\begin{aligned} k_{0\cdot95} &= 1\cdot96\, s/\sqrt{n} \\ &= 1\cdot96\sqrt{[\mathrm{S1}(1 - \mathrm{S1}/n)/[n(n-1)]]} \\ &= 1\cdot96\sqrt{[m(1-m)/(n-1)]} \end{aligned}$$

REFERENCES

1. C. Ponder and R. J. Fateman, 'Inaccuracies in program profilers', *Software—Practice and Experience*, **18,** 459–467 (1988).
2. E. Barkley and C. F. Schimmel. 'A performance study of the UNIX system V fork system call using CASPER', AT&T *Technical Journal*, September/October 1988, 100–109 (1988).
3. S. L. Graham and M. K. McKusick, 'An execution profiler for modular programs', *Software—Practice and Experience*, **13,** 671–685 (1983).
4. *UNIX System V/386 Release 3.2 Programmer's Reference Manual*, prof(1), profil(2), monitor, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
5. *UNIX System V/386 Release 3.2 Programmer's Reference Manual*, acctcms(1), acctcom(1), Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
6. *UNIX System V/386 Release 3.2* System *Administrator's Reference Manual*, ps(1), Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
7. *KDJ-11-D/S CPU Module User's Guide*, Digital Equipment Corporation, Maynard Mass., 1987.
8. *Microprocessor and Peripheral Handbook*, Vol. II, Peripheral, Intel Corporation, Santa Clara, California, 1989.
9. *RSX-11M-PLUS and Micro-RSX Executive Reference Manual*. Digital Equipment Corporation, Maynard Mass., 1987.
10. 80386 *Programmer's Reference Manual*, Intel Corporation, Santa Clara, California, 1986.
11. E. Kreyszig, *Introductory Mathematical Statistics*, Wiley, New York, 1970,