Paper 231-29

Taking a Deeper Look into the SAS® Intelligence Architecture

How to Use AIX Tools to Examine Interactions of Base SAS with the IBM @server pSeries

Frank Bartucca, IBM Corporation, Cary, NC
Laurie Jaegers, Raleigh, NC

ABSTRACT

In addition to SAS Software and programs, a deployed SAS business solution can contain host computer hardware and software, data storage systems, databases, third-party tools, networking equipment and so on from multiple vendors. The success of such a heterogeneous implementation of the SAS Intelligence Value Chain is dependent on the strengths of the interactions or "links" between the individual components deployed. This paper focuses on a set of AIX tools and on how they can be used to examine and tune the interactions between SAS Software and the POWER4 based IBM @server pSeries.

INTRODUCTION

A high-level description is given of the interfaces between Base SAS and AIX running on IBM @server pSeries. Next, the AIX trace facility is presented as the underlying mechanism used by several of the higher-level AIX monitoring and performance tools. Finally the POWER4 built-in hardware performance monitor (HPM) is described, along with a software tool kit that uses the HPM. Several examples are given to illustrate how these tools are used with Base SAS.

INTERFACES

The common wisdom that tells us “a chain is only as strong as its weakest link” extends to deployed business solutions, and a critical step in tuning interactions within a solution is to understand what comprises its links. The interfaces between SAS, AIX and the IBM @server pSeries are no different in that regard, and merit discussion before examining what their interactions are.

The high-level model of Base SAS Version 9.1 running on IBM @server pSeries, shown in Figure 1, illustrates the interfaces between SAS, AIX and the IBM @server pSeries. Figure 1 also shows a SAS process running on a system with other applications such as DB2 and WebSphere. The spaces between the processes illustrate the point that these processes do not directly interface with one another. Therefore, all forms of communication between the applications must involve the operating system. A direct interface between processes could be set up using shared memory, but operating system calls would still be made for efficient synchronization.

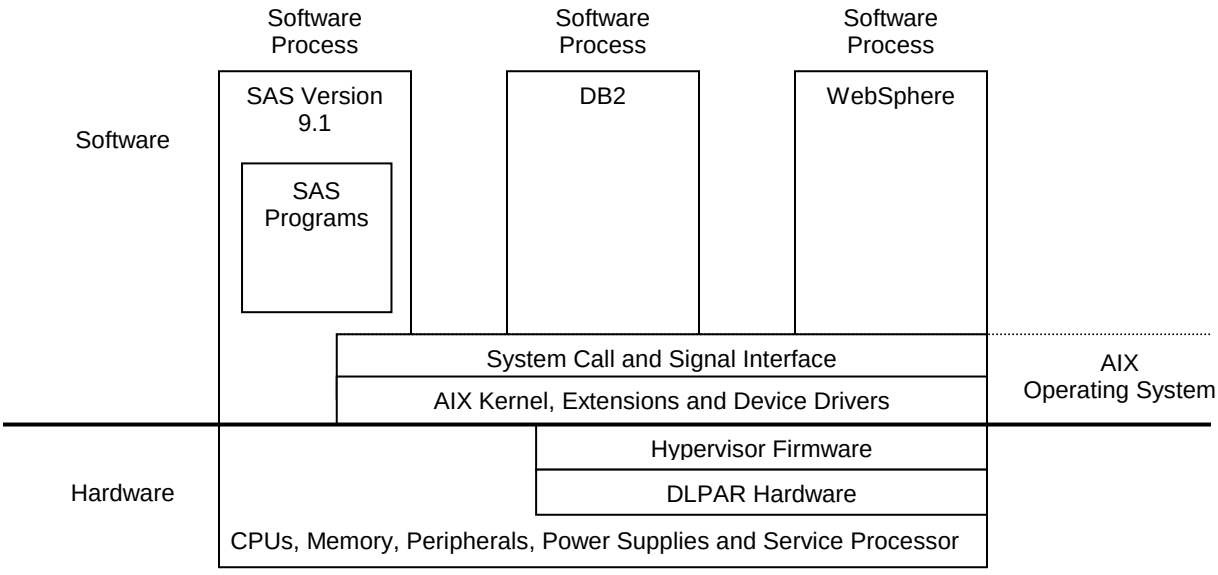

Figure 1. Interfaces Between SAS, AIX and the IBM @server pSeries

The bottom of Figure 1 shows the pSeries hardware layer, which consists of central processing units (CPUs), memory, peripherals, power supplies and the service processor. Next is the dynamic logical partitioning (DLPAR) hardware, which the hypervisor controls to allow multiple, independent operating system images to run concurrently on a single server. The AIX kernel along with any loaded device drivers and other kernel extensions are at the next level. AIX 5.2 is an operating system

that provides an execution environment for single threaded and multithreaded programs. Usually, when AIX runs, it runs directly on the pSeries CPUs in kernel mode. Sometimes, AIX calls hypervisor routines to support debuggers using breakpoints and to perform virtual memory and DLPAR tasks. In this case, calls to the hypervisor put the CPU in hypervisor mode, which is the highest level of privilege. So the levels of privilege from lowest to highest are user, kernel and hypervisor.

The next layer shown in Figure 1 is the system call and signal layer: SAS and AIX interface at this layer. Above this layer are kernel-only system processes and user-level single and multithread processes. Multithreading is a prominent new architectural feature that was introduced with SAS Version 9.1. Base SAS Version 9.1 is a system of single threaded and multithreaded programs and libraries that provides an execution environment for SAS programs. Base SAS runs directly on the pSeries CPUs in user mode.

SYSTEM CALL INTERFACE

SAS uses the system call interface to request service from AIX. System calls cause an interrupt to AIX and an automatic switch of the interrupted processor to kernel mode to run the selected system call. The system call continues to execute in the SAS thread context but at a higher privilege level. Because system calls execute in the context of the caller, the amount of CPU, memory and input/output (I/O) resources used during the execution of the system call are added to the resource use counters of the caller.

SIGNAL INTERFACE

The signal interface allows AIX to interrupt the normal execution of SAS to inform SAS about an event external to SAS or to inform SAS about an exception caused by SAS. There are a number of different signals and SAS can register SAS internal routines to handle or “catch” signals. SAS can also choose to ignore any number of signals although some signals cannot be caught or ignored. If SAS does not register a signal handler, then AIX performs the default action for that signal.

OVERVIEW OF THE TRACE FACILITY

The trace facility is a standard feature of the AIX operating system that allows real-time monitoring of system events. The AIX kernel, device drivers and other kernel extensions are instrumented with code that can generate a few words of information about system events *each time* an event occurs. Just to name a few, examples of system events include system calls, signals, device interrupts, file system access, thread creation and thread termination.

Words of information generated by a traced event are called the *trace data* of the event, while the sections of code in the kernel that generate the trace data are called *trace hooks*. There are approximately 400 permanent trace hooks installed in the shipped AIX kernel; each of these hooks has a unique number associated with it called a trace hook identifier, or *hook ID*. The *trace program* allocates buffers, enables trace hooks and copies data from trace buffers to trace files.

RECORDING TRACE DATA

Trace hooks can be individually enabled or disabled, and normally trace hooks are disabled so no trace data are generated. For the trace facility to run, some or all of the trace hooks must be enabled. Then, the enabled trace hooks write binary trace data to a pair of *trace buffers* that are pinned in main memory. The trace program allocates the buffers and pins them in main memory to protect them from being paged out to disk. When a trace buffer becomes full, the trace hooks switch to writing their data from the full buffer to the alternate trace buffer and the trace program transfers the trace data from the full buffer to the trace log file on disk; this is the default mode of operation.

The trace data generated for each traced event consist of a word that contains the trace hook identifier and the hook type. The hook identifier and word type are followed by a variable number of words of event information and, optionally, followed by a time stamp.

TRACE FACILITY OVERHEAD

For maximum usability, the trace facility was designed to have minimal impact on system performance. When the trace hooks are actively recording data in the trace buffers, the CPU overhead is less than 2 percent. When the trace program transfers the contents of a buffer to disk, the additional CPU overhead is usually less than 5 percent. This low overhead is possible because the trace facility does not make any attempt to process raw trace data as they stream into the trace buffers or are copied to disk. Enough information is available to post-process the raw trace.

Once a trace is captured, several tools that post-process the trace can be used to extract useful information such as performance and monitoring data about the workload running on the system. Discussion of these tools follows.

AIX PERFORMANCE AND MONITORING TOOLS

Several AIX tools can run the underlying trace facility and produce: CPU activity statistics for processes or threads, I/O statistics for trace intervals, CPU activity statistics of process and interrupt handlers for network-related events, process and thread activity profiles for trace intervals, synchronization lock activity statistics and execution profiles for trace intervals. Most of these tools can post-process a trace file that was captured previously. Full documentation for these commands can be found at http://publib16.boulder.ibm.com/pseries/en_US/infocenter/base/aix52.htm.

curt: the *C*PU *u*tilization and *r*eporting *t*ool takes an AIX trace file as input and outputs a number of statistics related to CPU activity at the process or thread level.
filemon: takes an AIX trace file as input and produces detailed reports of the disk based I/O activity that took place during the

traced interval. Reports can be generated that report on four different layers of I/O activity: the logical file system, virtual memory segments, logical volume manager and physical disk and adapter I/O processes.

netpmon: takes an AIX trace file as input and outputs reports on CPU activity by process and interrupt handlers, how much activity is network-related, network device driver I/O, socket calls and I/O from network file systems.

pprof: takes an AIX trace file as input and outputs up to five types of reports for process and thread activity during the trace interval: lists of kernel level threads sorted by actual CPU time, information for all processes with a common ancestor, lists of all processes grouped by families, information about each type of kernel thread and lists of all kernel threads sorted by start time.

splat: the *sim*ple *l*ock *a*nalysis *t*ool takes an AIX trace file as input and generates reports on the use of synchronization locks. These locks are the simple and complex locks provided by the AIX kernel, user-level mutex variables, read and write locks and condition variables provided by the pthread library. An additional level of control exists for tracing kernel locks. The command `locktrace` controls which kernel locks can be traced by the trace facility. The default mode is not to trace any locks. The command `locktrace -l` reports if lock tracing is enabled or disabled.

tprof: takes a trace file and produces an execution profile. To produce the execution profile tprof uses the trace hook with hook ID 234 (`HKWD_KERN_PROF`) along with other information in the trace. The `HKWD_KERN_PROF` trace hook records the contents of the instruction address register when a system clock interrupt occurs. The system clock interrupt occurs 100 times per second per CPU. The tprof tool takes this information and correlates the recorded instruction addresses with the range of addresses occupied by programs, kernel extensions and libraries.

trcrpt: takes a raw trace file and converts the binary trace hook data into human readable form. By default trcrpt produces one line of output for each trace event found in the trace file. The trcrpt command does not produce any summary reports.

OVERVIEW OF THE POWER4 HARDWARE PERFORMANCE MONITOR

The hardware performance monitor (HPM) is the hardware analog of the software-based trace facility. Most modern CPUs include additional logic to count the occurrence of hardware events generated by other macros on the chip. This type of logic is usually referred to as *instrumentation:* therefore, the POWER4 CPU is called an instrumented device. An automobile is a common instrumented device: it has sensors in its engine that collect data on engine speed, temperature and charging system output.

The HPM provides information regarding instruction execution, cache hits, branch prediction, floating point operation, address translation and other architectural features of the POWER4 CPU. Usually the main reason an HPM is designed into a chip is because its hardware designers want to collect performance and operational data to help them design the next generation chip. But some HPMs like the one in the POWER4 CPU can also be used to investigate software operation and performance. Employment of the HPM can aid in determining how efficiently the software is driving the hardware.

HPM OVERHEAD

At the hardware level, the HPM counters operate with no negative impact to the POWER4 CPU hardware performance. Either when a performance monitor counter (PMC) overflows, or a predefined counting threshold is reached, an interrupt is generated. At the software level, overhead is generated because the HPM device driver in AIX has to run to handle the interrupt and collect information from the counters. This housekeeping activity adds a small amount of overhead to AIX.

HPM TOOLKIT

The HPM Toolkit is a research tool developed at IBM Research for performance measurement of applications running on IBM systems that support POWER3 and POWER4 processors running AIX 5L or AIX 4.3.3. The toolkit contains three application programs and a subroutine library. As of the time of this writing it is available for licensing from IBM at http://www.alphaworks.ibm.com/tech/hpmtoolkit.

hpmcount: starts an application and, at the end of execution, provides wall clock time, hardware performance counters information, derived hardware metrics and resource utilization statistics.

hpmstat: collects information from system level hardware performance counters.

libhpm: an instrumentation library that provides instrumented programs with summary output containing the same information as hpmcount for each instrumented region in a program (resource utilization statistics are provided only once, for the whole section of the program that is instrumented). This library supports serial and parallel (MPI, threaded and mixed mode) applications, written in Fortran, C and C++.

peekperf: a graphical user interface for graphical visualization of the performance file generated by libhpm.

tcount: a sample program that is shipped with AIX 5.2 that uses the system-level performance monitor API; tcount accepts event numbers, counting flags and a workload to execute to produce a listing of the performance counters per CPU. The sample program tcount is located at /usr/sample/pmapi/tcount. The `tcount` sample program from the file set `bos.pmapi.samples` was used to make some of these measurements. AIX installation CDs include optional file sets of the form `bos.pmapi.xxxxx`.

EXAMPLES

For illustration purposes, a test workload was run with Base SAS using a selected set of performance and monitoring tools.

TEST WORKLOAD

These examples focus on the performance of SAS programs running on SAS Version 9.1 on an IBM 1.2-GHz POWER4+

p650 server running the AIX 5.2 OS. A large general linear model was selected as the test workload to drive an IBM p650 server with eight 1.2-GHz POWER4+ CPUs, 32MB of L3 cache and 32GB of main memory. This workload was selected because it is memory and CPU intensive. This model requires a minimum MEMSIZE of 512M.

USING AIX PERFORMANCE AND MONITORING TOOLS WITH BASE SAS

The following output is from the header of the report produced by curt. The header includes the command line used to record the trace as well as the command used to run curt. These lines help document the commands that have been used to generate the trace and produce the report:

```
Command used to gather AIX trace was:
trace -fap -C all -d -L 268435456 -n -o glm_run43 -T 268435456 -J curt,pprof,tprof
Command line was:
curt -a pid.43 -i glm_run43.raw -o glm_run43.curt -n run43.gen -m run43.trcnm -epstP
```

The -C all, -L and -T flags tell trace to allocate 256MB per CPU for trace buffers and to limit the size of trace files to 256MB per CPU. On an eight CPU system, this causes 2GB to be allocated and pinned for trace buffering. The trace command must be run as root to pin this much memory. The three flags in -fap tell trace to run in single mode and to stop collecting data as soon as one of the trace buffers fills, to run in the background and to include the CPU ID of the current processor with each recorded event. The -n flag adds more information to the trace log header and -o overrides the /var/adm/ras/trcfile default trace log file and writes trace data to glm_run43 in the current directory. The -d flag in the trace command line tells trace to set up the trace and run the workload without actually starting trace data collection. The section of SAS code that was traced was bracketed with trcon and trcoff commands as follows.

```
x trcon;     /* turn on the trace */
proc glm data='./allp';
  class version product contain resp;
  model var7=
    version resp(version) product version*product product*resp(version)
    contain version*contain contain*resp(version) product*contain
    version*product*contain product*contain*resp(version);
  test h=version e=resp(version);
  test h=product version*product e=product*resp(version);
  test h=contain version*contain e=contain*resp(version);
  test h=product*contain version*product*contain
       e=product*contain*resp(version);
run;
x trcoff;     /* turn off the trace */
```

For curt to display process names instead of process IDs, a file that maps process names to process IDs was specified with the -a flag. The file pid.43 was produced and properly formatted with by running the following command while the trace was running:

```
ps -Ao pid=,comm=|sed "s/ *//">pid.43
```

Running curt against the raw trace file with all the options turned on with -epstP produced a large report with 15542 lines. The following lines taken from the application summary part of the report show that during the traced interval the sas process was the dominant workload using about 87% of the processing time and running at least 154 threads.

```
                    Application Summary (by Pid)
                    ----------------------------
     -- processing total (msec) --          -- percent of total processing time --
    combined  application      syscall     combined  application      syscall  name (Pid)(Thread Count)
    ========  ===========      =======     ========  ===========      =======  ====================
5993278.3855  5979940.6030   13337.7825      87.4788      87.2842       0.1947  sas(356352)(154)
   3771.0777    3359.3813     411.6964       0.0550       0.0490       0.0060  IBM.CSMAgentRMd
(331940)(2)
  2734.1448     542.3905    2191.7544       0.0399       0.0079       0.0320  syncd(86120)(13)
  1499.5870     855.5995     643.9874       0.0219       0.0125       0.0094  dtgreet(241860)(1)
   920.7898     557.1357     363.6541       0.0134       0.0081       0.0053  IBM.HostRMd(278668)(1)
   812.3500     475.6885     336.6615       0.0119       0.0069       0.0049  rmcd(299158)(2)
```

The top ten lines taken from the thread summary part of the report, listed below, show a clustering of processing for the top eight threads. The ninth thread in the rank ordering has accumulated roughly half of the processing time of any of the top eight threads.

```
                   Application Summary (by Tid)
                    ----------------------------
     -- Processing total (msec) --          -- percent of total processing time --
    Combined application      syscall     combined application      syscall name (Pid Tid)
    ========  ===========      =======     ========  ===========      =======  ====================
231813.0362 231726.5559       86.4803       3.3836       3.3823       0.0013 sas (356352 938193)
231765.6087 231681.7973       83.8114       3.3829       3.3817       0.0012 sas (356352 1216685)
231208.2998 231097.5819      110.7179       3.3748       3.3731       0.0016 sas (356352 1179893)
231187.7007 231078.5298      109.1709       3.3745       3.3729       0.0016 sas (356352 1208517)
230914.8827 230830.3079       84.5748       3.3705       3.3692       0.0012 sas (356352 1204299)
230596.9152 230490.4496      106.4656       3.3658       3.3643       0.0016 sas (356352 1106005)
```

```
230072.2900 229935.5333     136.7567       3.3582       3.3562       0.0020 sas (356352 1044557)
229982.0550 229863.0776     118.9774       3.3569       3.3551       0.0017 sas (356352 864507)
105612.7035 105527.6429      85.0606       1.5415       1.5403       0.0012  sas(356352  1134765)
105435.6518  105348.6231      87.0287       1.5390       1.5377       0.0013  sas(356352  1101867)
```

In the following example, `tprof` is used to validate the results obtained using `curt` and to locate individual subroutines that are consuming the largest CPU bandwidth. The table below shows that 86.96% of total CPU time is taken up by the `sas` process. The `wait` process is the idle process that runs when a CPU has nothing to do. It has one thread for each CPU.

```
Process                                  Freq  Total Kernel   User Shared  Other
=======                                  ====  ===== ======   ==== ======  =====
sas                                       137  86.96   0.14   0.00  86.81   0.00
wait                                        8  11.91  11.91   0.00   0.00   0.00
PID-1                                       3   1.00   0.19   0.02   0.79   0.00
IBM.CSMAgentRMd                             2   0.05   0.00   0.00   0.04   0.00
/usr/sbin/syncd                             6   0.03   0.03   0.00   0.00   0.00
.
.
.
.
.
.
/usr/sbin/hostmibd                          1   0.00   0.00   0.00   0.00   0.00
/usr/sbin/syslogd                           1   0.00   0.00   0.00   0.00   0.00
/usr/sbin/cron                              1   0.00   0.00   0.00   0.00   0.00
=======                                  ====  ===== ======   ==== ======  =====
Total                                     186 100.00  12.32   0.03  87.66   0.00
```

From the listing below from `tprof,` it was calculated that of the time that all processes spent executing shared library code, 99.7% of that time was spent in the shared library `sasglm`.

```
Total % For All Processes (SH-LIBs) = 87.66
Shared Object                                                          %
=============                                                        ======
/sas/sas91-1118/sasexe/sasglm                                         87.40
/sas/sas91-1118/sasexe/sasvmth                                         0.06
/usr/lib/libpthreads.a[shr_xpg5_64.o]                                  0.05
/usr/lib/libc.a[shr.o]                                                 0.04
/sas/sas91-1118/sasexe/tkmk.so                                         0.02
/sas/sas91-1118/sasexe/sase7                                           0.02
/usr/lib/libc.a[shr_64.o]                                              0.02
/sas/sas91-1118/sasexe/tkepdl.so                                       0.01
/usr/lib/libpthreads.a[shr_xpg5.o]                                     0.00
```

Highlighted lines below from the `tprof` listing show that during the 14-minute `sas` run, 52% of the eight CPU capacity of the partition was spent running a single subroutine, the subroutine `tkzinner`, and 35% was spent running the subroutine `tkzvaxpy`.

```
Profile: /sas/sas91-1118/sasexe/sasglm
Total % For All Processes (/sas/sas91-1118/sasexe/sasglm) = 87.40

Subroutine                                    %   Source
==========                                 ====== ======
.tkzinner                                  51.89 /sas/m901_unx/tkvsubs/src/tkzinner.c
.tkzvaxpy                                  35.11 /sas/m901_unx/tkvsubs/src/tkzvaxpy.c
.glmxswp                                    0.09 /sas/m901_unx/stat/src/glmxswp.c
```

TRACE FACILITY RESULTS

Figure 2 illustrates SAS Version 9.1 thread creation and termination. These results were obtained using the AIX trace facility. The x-axis shows the progression of wall clock time in seconds: 0 s corresponds to the start of the trace interval. During the trace interval, 154 separate thread IDs were detected that were owned by the SAS processes. Two threads were created before the trace interval and were not terminated during the trace interval. The other 152 threads were created and terminated during the trace interval. Threads were rank ordered by time of creation, with the first thread that was created in the trace interval given the rank of 0 and the last thread that was created given the rank of 151. Thread rank is plotted on the y-axis. At any particular time, the vertical height of the shaded area is proportional to the number of threads that exist in the SAS process. Existence of a thread does not necessarily mean that it is active. The horizontal width of the shaded area for any given thread rank equals the length of time (in seconds) that thread existed. From time 0 seconds to approximately 800 seconds, threads appear to be created and deleted in groups of eight.

USING THE POWER4 HARDWARE PERFORMANCE MONITOR WITH BASE SAS

The following examples discuss using the POWER4 hardware performance monitor to dig deeper into understanding CPU and memory utilization. Since general linear modeling usually requires matrix multiplication, it generally produces a fair

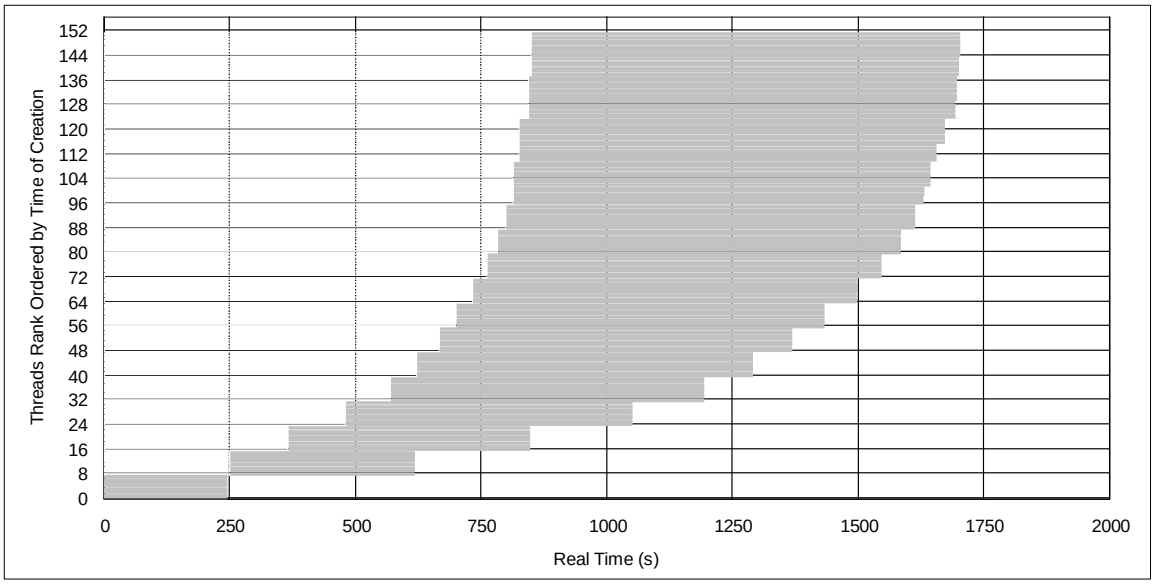

Figure 2. Thread Creation and Termination

amount of computational intensity. Measurements are taken on the floating-point units while SAS is performing matrix multiplication to investigate how efficiently SAS employs the floating-point units. These examples show the output reports that were produced by `tcount` and `hpmcount` on the large general linear model running on SAS Version 8.2 and SAS Version 9.1.

The POWER4 HPM tools currently support sixty-three event groups. Up to eight events can be counted at one time in each event group. The event group `pm_hpmcount2` shown below is used to record instruction execution events for three types of floating point FP instructions. The POWER4 CPU has two floating-point units, `FPU0` and `FPU1`. Performance monitor counter one (`PMC 1`) increments for FP divide instructions. `PMC 2` increments for execution of floating point fused multiply and add type of instructions {fpA = fpB + (fpC * fpD) is performed with a single multiply-add instruction}. `PMC 3` and `PMC 4` increment when `FPU0` or `FPU1` produces a result receptively. `PMC 5` counts processor cycles over the measurement interval. `PMC 6` increments when either `FPU0` or `FPU1` executes a store instruction. `PMC 7` increments when any instruction completes execution. `PMC 8` increments when the load store unit (LSU) executes an FP load instruction.

```
Group 60: pm_hpmcount2
Counter  1, event 84: PM_FPU_FDIV
Counter  2, event 83: PM_FPU_FMA
Counter  3, event 22: PM_FPU0_FIN
Counter  4, event 27: PM_FPU1_FIN
Counter  5, event 82: PM_CYC
Counter  6, event 84: PM_FPU_STF
Counter  7, event 78: PM_INST_CMPL
Counter  8, event 78: PM_LSU_LDF

SAS Version 8.2
CPU   PMC 1          PMC 2    PMC 3          PMC 4
===  =======  ===========  ============  ==========
[0]      338            0         18213        13205
[1]       31            0          2070         1458
[2]        0            0          1048          642
[3]        0            0            52           12
[4]  5877907  77511422877  77665109908  77539014833
[5] 90594060 410648509362 149577427079 411120230241
[6]        0            0           564          778
[7]        0            0           516          792
=== ======== ============ ============ ============
ALL 96472336 488159932239 227242559450 488659261961

CPU       PMC 5         PMC 6          PMC 7           PMC 8
===  ============= ============  ============ ============
[0]     5809223767          7304    3685895458         13933
[1]     8612521721          1337    5752291480          2139
```

```
[2]     3190873050         1690    2131931676         2066
[3]      654204884           64     174016865           80
[4]  1176965933958  77552882396  389259895077 155168242134
[5]  4117218539298 148936273582 1814368240929 821683118037
[6]     5512809871          967    2628590815          631
[7]     5114269132          972    2134932312          437
===  ============= ============  ============ ============
ALL  5323078375681 226489168312 2220135794612 976851379457

SAS Version 9.1
CPU    PMC 1    PMC 2        PMC 3        PMC 4
===  ======= ============ ============ ============
[0]  4052217  60232282660  60246054289  27820492313
[1]  4053684  60402055531  60415859253  27977016392
[2]  4093745  60775766386  60851221609  28072827511
[3] 14078072  60183997773  60275742090  27871927771
[4]  4033786  60601397175  60614968077  28206952540
[5]  4043442  60430110772  60443913172  28131123712
[6]  5350887  63105589665  63374649236  29596044537
[7]  4922479  62425148876  62496648957  29115723987
=== ======== ============ ============ ============
ALL 44628312 488156348838 488719056683 226792108763

CPU      PMC 5        PMC 6        PMC 7        PMC 8
=== ============= ============  ============ ============
[0]  888771553871  27807291944  269400474210 120488841114
[1]  892092608078  27962798392  270176410713 120830864525
[2]  901444683390  28068386322  274489326094 121624912721
[3]  904737454405  27859238676  270354148722 120457322660
[4]  895890462537  28197115699  270965167855 121219185374
[5]  895922832946  28121143671  270231362153 120876953285
[6]  928228943610  29412281111  287055715459 126369505759
[7]  913629899345  29065314808  280774189688 124897962261
=== ============= ============ ============= ============
ALL 7220718438182 226493570623 2193446794894 976765547699
```

HPM RESULTS PRESENTED GRAPHICALLY

Group 60 (`pm_hpmcount2`) performance monitor counter data for Version 9.1 are normalized to Version 8.2 data that were obtained under the same conditions: executing the large general linear model workload on an IBM p650 server equipped with eight 1.2-GHz POWER4+ CPUs, 32MB of L3 cache and 32GB of memory running AIX 5.2-02. Figure 3 shows that SAS Version 9.1 and SAS Version 8.2 HPM results are the same for each of the following counters: fused multiply add instructions executed (`FPU_FMA`), floating point instruction results produced (`FPU_FIN`), load-store unit floating point store instructions executed (`LSU_STF`), total instructions completed (`INST_CMPL`) and load-store unit floating point load instructions executed (`LSU_LDF`).

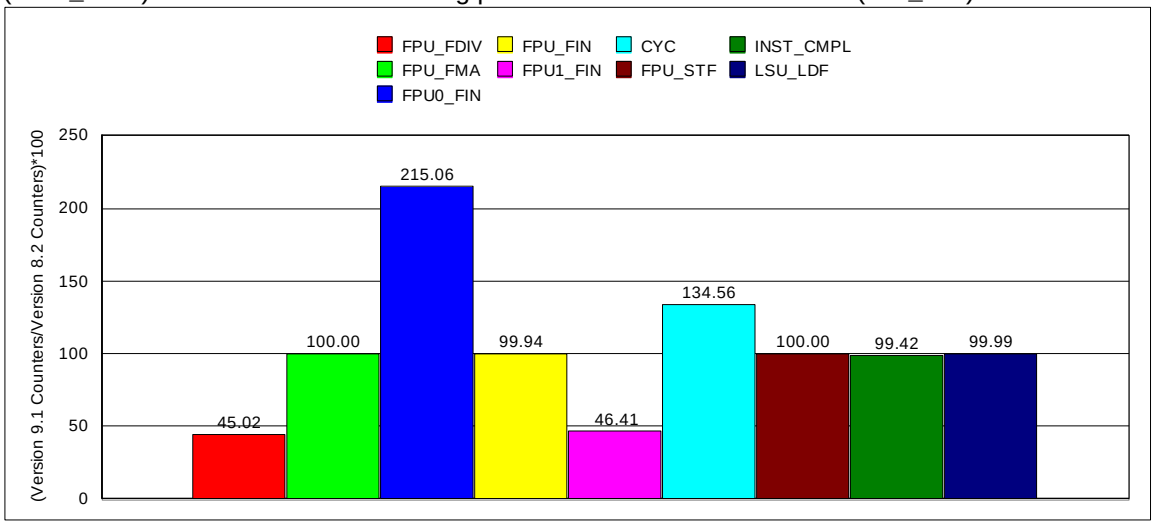

Figure 3. SAS Version 9.1 Normalized to SAS Version 8.2 HPM Results for Group 60 Counters

Figure 4 shows the raw counter values in a side-by-side comparison of SAS Version 9.1 and SAS Version 8.2.

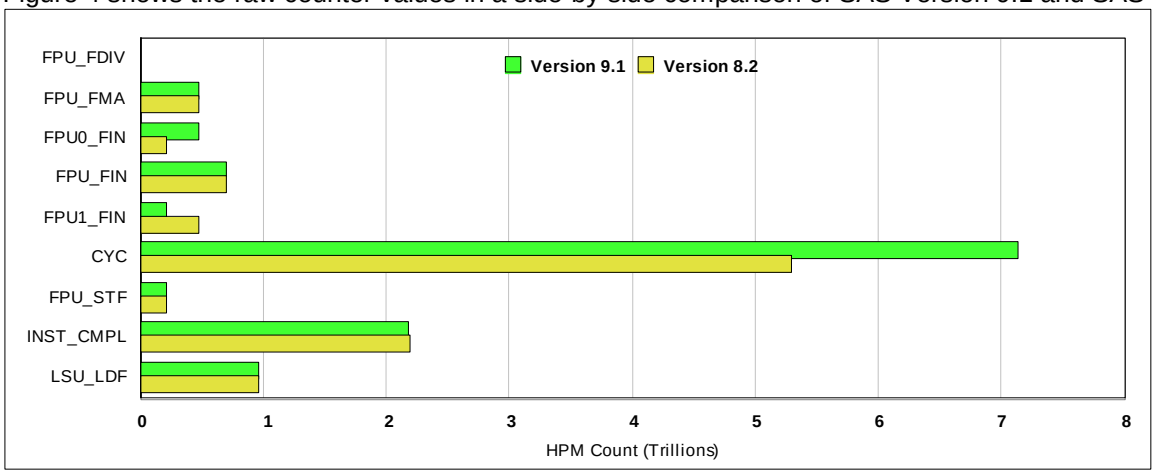

Figure 4. HPM Group 60 Counter Data for Floating Point Computational Intensity Analysis

Figure 5 shows Group 28 (`pm_fpu3`) performance monitor counter data for SAS Version 9.1 executing the large general linear modeling workload on an IBM pSeries p650 server equipped with eight 1.2-GHz POWER4+ CPUs, 32MB of L3 cache and 32GB of memory running AIX 5.2-02. The number of floating point divide instructions (`FPU_FDIV`), floating point to/from fixed point conversion instructions (`FPU_FRSP_FCONV`), fused multiply add instructions executed (`FPU_FMA`) and total instructions completed (`INST_CMPL`) are about the same for SAS Version 9.1 and SAS Version 8.2. More total CPU cycles were taken to complete the 14-min 15-s run by SAS Version 9.1, whereas the SAS Version 8.2 run took 1 hr 14 min to complete.

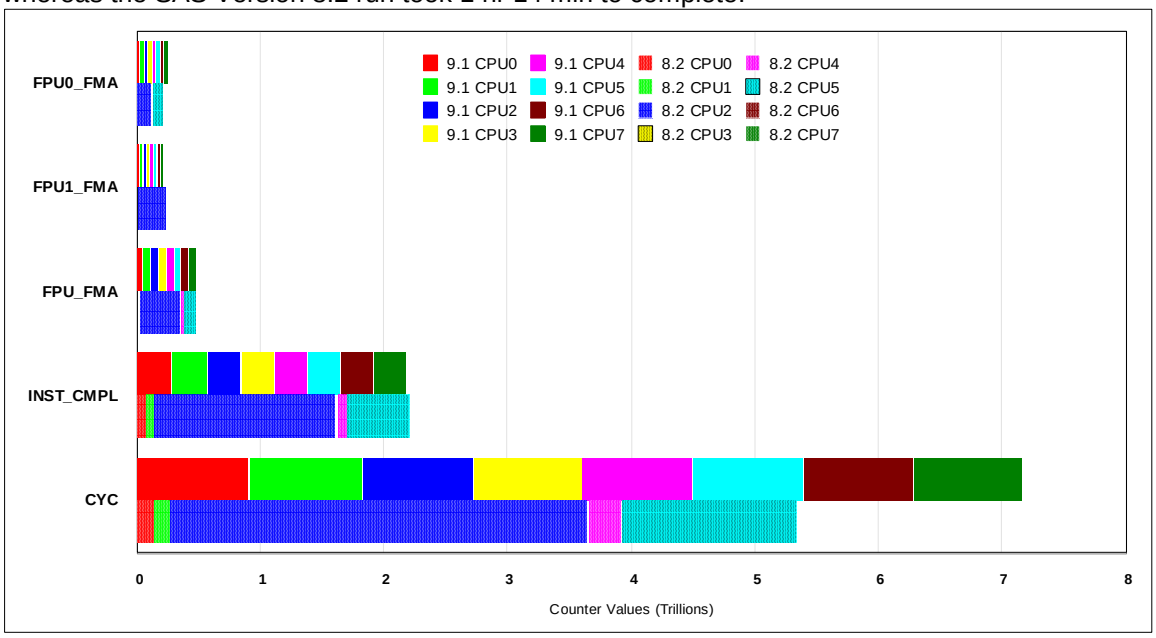

Figure 5. SAS Group 28 HPM Counter Data

Figure 6 focuses on the execution of the fused multiply add (`FMA`) instruction, which is the most efficient instruction to use for matrix multiplication. It appears that SAS Version 9.1 has a slight preference for `FPU0` for the `FMA` whereas v8.2 has a slight preference for `FPU1`. However, the important result to note here is that each version executed almost the same total number of `FMA` instructions.

CONCLUSION

The POWER4 hardware performance monitor and the AIX trace facility are two complementary features of the IBM @server pSeries. Together with higher-level software tools, they provide the means to investigate the interactions between SAS, AIX and the pSeries. An introduction to the AIX trace facility and POWER4 HPM was presented along with a high-level model of the interfaces between SAS, AIX and the IBM @server pSeries.

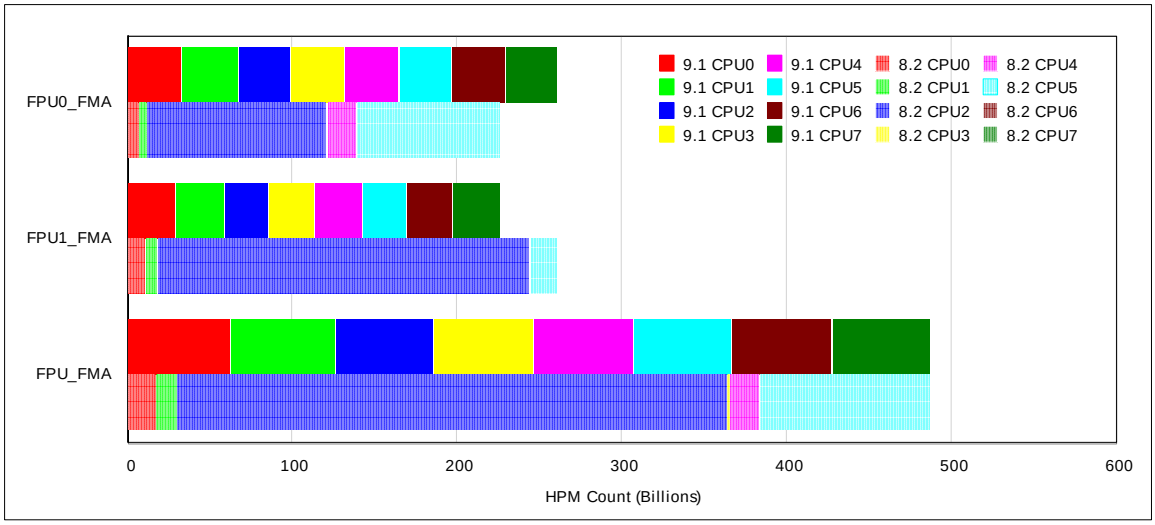

Figure 6. Execution of the Fused Multiply Add Instruction

Examples were given of using both the trace facility and the HPM, with a large general linear model selected as the workload. Actual text-based output from higher-level tools was presented and discussed and some of the data were presented in graphical form. The `curt` and `tprof` tools were run against a captured trace of the executing workload, which showed that most of the CPU time was spent performing two subroutines. The trace facility was also used to investigate thread creation and termination. The results from the HPM toolkit showed that SAS Version 9.1 distributed the computational load over the eight CPUs available on the p650.

SPECIAL NOTICES

The information contained in this document has not been submitted to any formal IBM test and is distributed “AS IS”. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. The use of this information or the implementation of any techniques described herein is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. Customers attempting to adapt these techniques to their own environments do so at their own risk.

IBM is not responsible for printing errors in this publication that result in pricing or information inaccuracies.

Any performance data contained in this paper were obtained in a controlled environment. Some measurements quoted in this paper may have been made on development-level systems. There is no guarantee these measurements will be the same on generally available systems. Some measurements quoted in this paper may have been estimated through extrapolation. Actual results may vary. Users of this paper should verify the applicable data for their specific environment.

REFERENCES

Luiz DeRose, "The Hardware Performance Monitor Toolkit". In the Proceedings of Euro-Par 2001. Manchester, UK, August 2001, pp: 122-131.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author: Frank Bartucca
Email: ibmsas@us.ibm.com

Author: Laurie Jaegers
Email: ljaegers@bellsouth.net