



update on the perfmon2 monitoring interface

Stéphane Eranian

HP Labs

Gelato Meeting, April 2006 – San Jose, CA

© 2005 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice



Agenda

- Kernel level interface and implementation
- User level tools

kernel interface versions

- Version available in `/proc/perfmon`
- Stable version: v2.0
 - mainline kernel
 - SLES9, SLES10 (Code 10)
 - RHEL4, RHEL5
 - support Itanium processors only
- Development version: v2.2
 - separate kernel patch against 2.6.17-rc1
 - multi-architecture
 - very likely to become Linux standard monitoring interface

What's new in 2.2?

- Multiple system calls instead of perfmonctl()
 - backward compatibility layer on IA-64
- Multi-architectures:
 - IA-64, AMD X86-64, Pentium M/P6, MIPS64(Mucci), Power5 (IBM)
 - Pentium 4/Xeon(Intel) includes support for HyperThreading and PEBS (Thanks to kernel sampling formats!)
- Kernel level event set multiplexing
- PMU description tables as kernel modules
- Virtual PMD registers to map any SW resource
- /sys interface for infos, stats, config

Event sets

- What is the problem?
 - never enough counters
 - Itanium 2 PMU: 4
 - Montecito PMU: 12 (still not enough!)
 - event constraints:
 - event A and B cannot be measured together
 - if event A is measured then only event C can be measured
 - some measurements require a lot of events:
 - e.g., cycle breakdown requires at least 15 events on Itanium 2 PMU
- Solution:
 - create sets of up to m events when PMU has m counters
 - multiplex sets on actual PMU
 - total counts **approximated** by simple scaling calculation

Event sets

- Kernel level switching for efficiency
 - switching ALWAYS occur in the monitored context
- Each set encapsulates the full PMU state
 - identified by user-specified unique number
 - sets can dynamically be added, modified, or removed
 - sets are ordered based on their unique number
- Round-robin set switching
 - timeout-based switching:
 - timeout specified per set
 - granularity depends on OS timer, actual timeout returned to user
 - overflow-based switching:
 - after n overflows of a “trigger” counter
 - multiple simultaneous triggers are supported

PMU description modules

- Make it easier to update, add new PMU support
 - isolate PMC/PMD mapping information in a table
 - minimize changes to arch-specific code
- PMU mapping table provided by kernel module
 - only one table loaded at a time
 - automatically loaded on first use, no need to be root
- Virtual PMDs: expose SW/HW resources as PMD
 - e.g.: PMD255 = current->pid, PMD254 = current stack ptr
 - trivial to include in samples or read
 - module implements read/write callbacks
 - goal: architected Virtual PMD, arch-specific virtual PMD

Current work

- Working at getting code accepted into mainline
 - perfctr (Mikael Petterson): not pursuing kernel merge anymore
 - OProfile (John Levon):
 - already integrated with perfmon2 on IA-64 (Oprofile sampling format)
 - working with RedHat on migrating x86 OProfile
- Lots of code cleanups:
 - split C and header files
 - help from IBM, Redhat and many others
- Better integration with NMI watchdog timer on x86
- Kernel level perfmon2 interface for SystemTap

libpfm

- Libpfm library: latest 3.2-060421
 - Itanium, Itanium 2, dual-core Itanium 2 (Montecito) processors
 - AMD X86-64 processors
 - Intel Pentium M/P6 processors
 - MIPS20k/MIPS5k processors (Phil Mucci)
- Ability to specify bitmask of available PMC registers
 - useful when only subset of PMU registers are available
- Expecting support for P4 and Power5

q-tools

- System-wide statistical profiler from HPLabs (davidm)
- Provides flat profile and call graph for user and kernel level execution
- Does not requires recompilation

- Version 0.3: to be released very soon
 - includes support for Montecito processor
 - works on perfmon v2.0 and v2.2

pfmon

- New version (3.2-060426) finally released today!
- Collect counts or profiles per-thread or system-wide
- New features:
 - support for perfmon v2.0 and v2.2
 - event set multiplexing support for counting (v2.2 only)
 - multi-arch support:
 - Itanium, and Itanium 2 processors
 - Montecito processor: 12 counters, 41bit opcode match, 64kB code range, 16-entry ETB.
 - AMD X86-64, Intel Pentium M/P6
 - IA-64 (v2.2): excl./incl. intr. in system-wide
 - IA-64 (v2.2): no-start, insecure mode,
 - exclude idle thread
 - new sampling modules: inst-hist, dear-hist (IA-64)

pfmon multiplexing example

- define multiple sets and time-multiplex:

```
pfmon -ecpu_cyclesia64_inst_retired,nops_retired,stores_retired
      -eloads_retired
      --switch-timeout=10 -us-c noploop 10000000000
13,328,847,960 CPU_CYCLES
29,989,576,774 IA64_INST_RETIRE
 9,996,523,758 NOPS_RETIRE
 1,412 STORES_RETIRE
 0 LOADS_RETIRE
```

- careful of blind spots:

```
pfmon -eloads_retired -us-c noploop 10000000000
```

```
1,900 LOADS_RETIRE
```

```
pfmon -enops_retired -us-c noploop 10000000000
```

```
10,000,005,431 NOPS_RETIRE
```

- run for long enough, look for ratios more than exact counts

pfmon dear-hist module

- Itanium processors only
- instruction/date view of D-EAR (cache/tlb) samples

```

# instruction addr view
# sorted by count
# showing per distinct value
# L2    : 5 cycles load latency
# L3    : 14 cycles load latency
# %L2   : percentage of L1 misses that hit L2
# %L3   : percentage of L1 misses that hit L3
# %RAM  : percentage of L1 misses that hit memory
#count  %self  %cum   %L2    %L3    %RAM   instruction addr
  6349  11.09%  11.09%  26.02%  69.98%  4.00%  0x400000000000077d0
  6206  10.84%  21.93%   3.53%   4.80%  91.67%  0x400000000000077a0
  5427   9.48%  31.41%  75.97%  23.75%   0.28%  0x4000000000000c570
  5083   8.88%  40.29%   0.08%  44.21%  55.72%  0x4000000000000d751
  5004   8.74%  49.03%  50.60%  31.65%  17.75%  0x4000000000000c5c1
  4844   8.46%  57.50%  37.82%  51.03%  11.15%  0x4000000000000c5b0
  4812   8.41%  65.90%   0.29%  91.77%   7.94%  0x4000000000000c581
  3697   6.46%  72.36%   0.00%   9.36%  90.64%  0x40000000000001ce1
  2110   3.69%  76.05%   0.00%  50.57%  49.43%  0x40000000000006390
  2057   3.59%  79.64%   9.29%  62.91%  27.81%  0x40000000000001d60

```

pfmon dear-hist module (2)

- same program but data view

```
# data addr view
# sorted by count
# showing per distinct value
# L2    : 5 cycles load latency
# L3    : 14 cycles load latency
# %L2   : percentage of L1 misses that hit L2
# %L3   : percentage of L1 misses that hit L3
# %RAM  : percentage of L1 misses that hit memory
# #count  %self  %cum   %L2    %L3    %RAM    data addr
      55   0.10%  0.10%  61.82% 30.91%  7.27%  0x2000000000017ebd0
      34   0.06%  0.16%  58.82% 38.24%  2.94%  0x200000000000e5390
      30   0.05%  0.21%  70.00% 23.33%  6.67%  0x200000000000da878
      29   0.05%  0.26%  79.31% 10.34% 10.34%  0x200000000000e4e68
      26   0.05%  0.30%  50.00% 34.62% 15.38%  0x200000000000dd050
      26   0.05%  0.35%  92.31%  3.85%  3.85%  0x200000000000d7a88
      25   0.04%  0.39%  60.00% 40.00%  0.00%  0x200000000000d93d8
      24   0.04%  0.44%  87.50%  8.33%  4.17%  0x200000000000dcd8f8
      24   0.04%  0.48%  58.33% 37.50%  4.17%  0x200000000000e0c50
      24   0.04%  0.52%  91.67%  8.33%  0.00%  0x200000000000eaac0
```

pfmon dear-hist module (3)

- show latency distribution

```
# level view
# sorted by value
# showing per distinct value
# L2    : 5 cycles load latency
# L3    : 14 cycles load latency
# #count    %self    %cum  lat(cycles)  lat(ns)
  14335    25.04%    25.04%         5         3
  10868    18.99%    44.03%         7         5
   1274     2.23%    46.25%         9         6
     21     0.04%    46.29%        10         7
    481     0.84%    47.13%        11         7
     3      0.01%    47.14%        12         8
     7      0.01%    47.15%        13         9
  11501    20.09%    67.24%        14         9
   6408    11.19%    78.43%        15        10
    288     0.50%    78.94%        16        11
```

HP Caliper

- works with v2.0 and v2.2
- Montecito example: combining IP-EAR and Data-EAR

```
void
product(short *restrict a, short *restrict b, short *restrict c)
{
    unsigned int i;

    for(i=0; i < N; i++)
        a[i] = b[i]*c[i];
}
$ caliper cycles prog; caliper dcache_miss prog;
$ caliper report -ri --join merge cycles dcache_miss
```

% Total					Avg.	
Cycles	Sampled	Dcache	Dcache	Line		
Per	Dcache	Latency	Laten.	Slot		
Bundle	Misses	Cycles	Cycles	Col,Offset		
12.0	5147	56555	11.0	0x00e0:0	[bundle]	
	0	0			M_ ld2	r25=[r26] ; ;
	0	0			:1 M nop.m	0
					:2 I_ pmpy2.r	r23=r24,r25 ; ;

Miss latency=11 cycles, split bundle, total cost of bundles = 11+1= 12 !

Conclusion

- perfmon2 offers most complete set of features
 - perfmon2 is portable
 - implementation on several key architectures
 - working on merging v2.2 in mainline
 - growing community of user and contributors
-
- all software and documentation available at:
<http://perfmon2.sf.net>



i n v e n t