



perfmon2: a flexible
performance monitoring
interface for Linux

perfmon2: une interface
flexible pour l'analyse de
performance sous Linux

Stéphane Eranian

HP Labs

July 2006

Ottawa Linux Symposium 2006

Ottawa, Canada

© 2006 Hewlett-Packard Development Company, L.P.

The information contained herein is subject to change without notice



Agenda

- What is performance monitoring?
- What is the PMU?
- What is the problem?
- Overview of the perfmon2 interface
- Status
- Existing tools
- Current & future work

What is performance monitoring?

The action of collecting information related to how an application or system performs

- Information obtained by instrumenting the code
 - extract program-level or system-level information
 - statically: compilers (-pg option), explicit code (LTTng, Xenmon)
 - dynamically (code rewrite): HP Caliper, Intel PIN tool, Kprobes
 - example: count basic-block execution, number of ctxsw/s
- Information obtained from CPU/chipset
 - extract micro-architectural level information
 - exploit hardware performance counters
 - example: count TLB misses, stall cycles, memory access latency

Why is monitoring important?

- Transparent HW improvements are slowing down
 - multi-thread, multi-cores, NUMA instead of clock speed
 - HW changes impact SW more than in the past
- SW compute needs are continuing to grow fast
- major SW evolutions necessary to better exploit HW
 - example: SW must be multi-threaded for multi-core CPU
- Monitoring is key to detecting/solving problems
 - where are the stalls, what's the memory bandwidth utilization?
 - develop smart tools: must tell why and what to do about it?
 - better monitoring HW: stall, cache analysis, bus utilization

What is the PMU?

- Piece of CPU HW collecting micro-architectural events:
 - from pipeline, system bus, caches, ...
- All modern CPU have a PMU
 - architected for IA-64, AMD64, now finally for IA-32
- PMU is highly specific to a CPU implementation
 - large differences even inside a processor family (e.g., X86)
- PMU becoming a key value-add
 - improving fairly rapidly
- Many new PMUs go beyond just collecting counts
 - Itanium PMU: where cache misses occur, src/tgt of branches

Why do we need kernel support?

- Some operations require privileged access
 - processing/setup of PMU interrupts on counter overflow
 - writes/read registers
- Some PMUs allow certain operations at user level:
 - IA-64: read PMU data register (PMD), start and stop
 - X86 : read counter (RDPMC)

Diversity of PMU HW

- Itanium 2 9000 (Montecito):
 - uses indexed registers: PMC, PMD. PSR for start/stop
 - 12 counters (47bits), interrupt on overflow, atomic freeze
 - PMC for opcode filters, code & data range restrictions
 - 3 PMD+1 PMC to capture where cache/TLB misses occur
 - 16 PMD+ 1 PMC for Branch Trace Buffer
- AMD64:
 - uses MSR registers (PERFEVTSEL, PERFCTR)
 - 4 counters (40 bits), interrupt on overflow, no atomic freeze
- Pentium 4:
 - uses MSR registers (CCCR, ESCR, PERFCTR)
 - 18 counters (40 bits), interrupt on overflow, no atomic freeze
 - Precise Event Based Sampling (PEBS), HyperThreading

Diversity of usage models

- Type of measurement
 - counting
 - sampling (profiling): at some interval (time or other), record in a sample information about the state of a program/CPU/system.
- Scope of measurement
 - system-wide: across all threads running on a CPU
 - per-thread: a designated thread (self-monitoring or unmodified)
- Scope of control:
 - from user level programs: monitoring tools, compilers, MRE
 - from the kernel: SystemTap, scheduler
- Scope of processing:
 - off-line: profile-guided optimization (PGO), manual tuning
 - in-line: dynamic optimization (DPGO)

Existing monitoring interfaces

- Oprofile (John Levon):
 - included in mainline kernel and most distributions
 - system-wide profiling only, support all major platforms
- Perfctr (Mikael Pettersson)
 - separate kernel patch
 - provides per-thread, system-wide monitoring
 - designed for self-monitoring, basic sampling support
 - support all IA-32, PowerPC
 - used by PAPI toolkit
- VTUNE driver (Intel)
 - open-source driver specific to VTUNE

no standard and generic interface exists

Goals of the perfmon2 interface

- Provides a **generic** interface to access the PMU
 - not dedicated to one application, avoid fragmentation
- Be portable across all PMU models/architectures
 - almost all PMU-specific knowledge in user level libraries
- Supports **per-thread** monitoring
 - self-monitoring, unmodified binaries, attach/detach
 - multi-threaded and multi-process workloads
- Supports system-wide monitoring
- Supports counting and sampling
- No special recompilation
- **Builtin**, efficient, robust, secure, simple, documented

Perfmon2 interface (1)

- Core interface allows read/write of PMU registers
- Uses the **system call** approach (rather than driver)
- Perfmon2 **context** encapsulates all PMU state
 - each context uniquely identified by file descriptor
 - file sharing semantic applies for context access
- Leverages existing mechanisms wherever possible
 - e.g., file descriptors, signals, mmap, ptrace

```
int pfm_create_context(pfarg_ctx_t *ctx, void *a, size_t sz)
int pfm_write_pmcs(int fd, pfarg_pmc_t *pmcs, int n);
int pfm_write_pmds(int fd, pfarg_pmd_t *pmcs, int n);
int pfm_read_pmds(int fd, pfarg_pmd_t *pmcs, int n);
int pfm_load_context(int fd, pfarg_load_t *ld);
int pfm_start(int fd, pfarg_start_t *st);
```

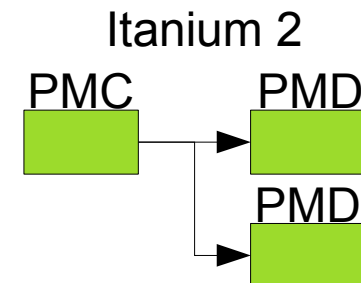
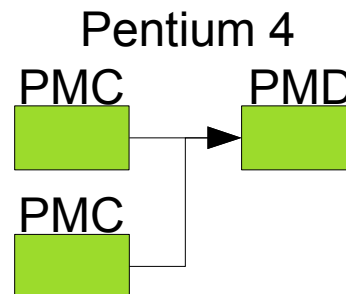
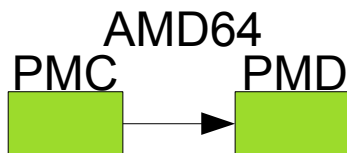
```
int pfm_stop(int fd);
int pfm_restart(int fd);
int pfm_create_evtsets(int fd, pfarg_setdesc_t *st, int n);
int pfm_delete_evtsets(int fd, pfarg_setdesc_t *st, int n);
int pfm_getinfo_evtsets(int fd, pfarg_setinfo_t *it, int n);
int pfm_unload_context(int fd);
int close(int fd);
```

Perfmon2 interface (2)

- Uniformity makes it easier to write portable tools
- Counters are always exported as 64-bit wide
 - emulate via counter overflow interrupt capability if needed
- Exports logical view of PMU registers
 - PMC: configuration registers
 - PMD: data registers (counters, buffers, ...)
- Mapping to actual registers depends on PMU
 - defined by PMU description kernel module
 - visible in `/sys/kernel/perfmon/pmu_desc/mappings`
- Possibility to have virtual PMU registers
 - can map to SW or non-PMU processor/system resource
 - e.g, `PMC256 = ibr0 (IA-64)`, `PMD260 = current->pid`

Perfmon2 interface (3)

- Same ABI between ILP32 and LP64 modes
 - all exported structures use fixed-size data types
 - EM64T, AMD64: 32-bit tools run unmodified on 64-bit kernel
 - kernel: no 32 \leftrightarrow 64 syscall argument conversions
- Vector arguments for read/write of PMU registers
 - portable: decoupled PMC/PMD = no dependency knowledge
 - extensible: no knowledge of # registers of PMU
 - efficient and flexible: can write one or multiple regs per call



Basic self-monitoring per-thread session



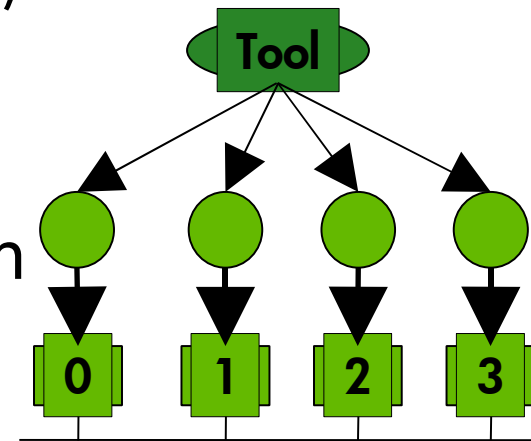
```
pfarg_ctx_t ctx; int fd;
pfarg_load_t load;
pfarg_pmd_t pd[1]; pfarg_pmc_t pc[1];
pfmlib_input_param_t inp;
pfmlib_output_param_t outp;
pfm_find_event("CPU_CYCLES", &inp.pfp_events[0]);
inp.pfp_plm = PFM_PLM3; inp.pfp_count = 1;
pfm_dispatch_events(&inp, NULL, &outp);
pd[0].reg_num = pc[0].reg_num = outp.pfp_pc[0].reg_num;
pfm_create_context(&ctx); fd = ctx.ctx_fd;
pfm_write_pmcs(fd, pc, 1);
pfm_write_pmds(fd, pd, 1);
load.load_pid = getpid();
pfm_load_context(fd, &load);
pfm_start(fd, NULL);
/* run code to measure */
pfm_stop(fd);
pfm_read_pmds(fd, pd, 1);
printf("total cycles %"PRIu64"\n", pd[0].reg_value);
close(fd);
```

Per-thread session

- Thread = kernel visible thread (task)
- PMU state is saved/restored on context switch
 - multiple per-thread sessions can run concurrently
- Context attachment:
 - one context per thread
 - thread must be stopped: leverage `ptrace()`
 - can attach at fork or when thread is already running
- Context inheritance is never done by kernel
 - leverage 2.6 `ptrace()` options (`PTRACE_O_TRACE*`)
 - for many measurements, tools must be aware of new thread
 - can selectively follow across: `fork`, `exec`, `pthread_create`
 - aggregation done by the tool, if needed

System-wide session

- Monitors across all threads running on one CPU
- Uses exact same programming sequence as per-thread
 - type selected when context is created
 - monitored CPU is current CPU during `pfm_load_context()`
- System-wide (SMP) built as union of CPU-wide sessions
 - flexibility: measure different metric on different CPUs
 - scalability: strong affinity (processor, cache)
 - simplicity: easier error handling, no IPI
 - ready for HW buffer: Pentium 4 PEBS
- Mutual exclusion with per-thread session
 - due to some HW limitations
 - current SW implementation limitations



Support for sampling

- Support for Event-Based Sampling (EBS) in kernel
 - period p expressed as $2^{64}-p$ occurrences of an event
- Can request notification when 64-bit counter overflows
 - notification = message and is extracted via `read()`
 - support for `select/poll/asynchronous` notification (SIGIO)
- Number of sampling periods = number of counters
 - allows overlapping sampling measurements
- Optional support for kernel level sampling buffer
 - amortize cost of notification by notifying only when buffer full
 - buffer remapped read-only to user with `mmap()`: zero copy
 - periods can be **randomized** to avoid biased samples
 - can indicate per PMD: list of PMDs to record/reset on overflow

Custom sampling buffer formats

- No single format can satisfy all needs
 - must keep complexity low and extensibility high
- Export kernel interface for plug-in formats
 - easier to port existing tools/infrastructure: OProfile
 - easier to exploit HW features: Pentium 4 PEBS buffer
- Each format provides:
 - 128-bit UUID for identification (passed on context creation)
 - A handler function called on counter overflow
- Each format controls:
 - Where and how samples are stored
 - What gets recorded, how the samples are exported
 - When a user notification must be sent to user

Existing sampling formats

- Default format (builtin):
 - linear buffer, fixed header followed by optional PMDs values
- Oprofile format (IA-64 for now):
 - 10 lines of C, reuse all generic code, small user level changes
- N-way sampling format (released separately):
 - implements split buffer (up to 8-way)
 - parsing in one part while storing in another: fewer blind spots
- Kernel call stack format (experimental, IA-64):
 - records kernel call stacks (unwinder) on counter overflow
- Pentium 4 Precise Event Based Sampling (PEBS)
 - 100 lines of C, first interface to provide access to feature!

Event sets and multiplexing (1)

- What is the problem?
 - number of counters is often limited (4 on Itanium®2 PMU)
 - some events cannot be measured together
 - some measurements require a lot of events:
 - e.g., cycle breakdown on Itanium®2 CPU requires at least 15 events
- Solution:
 - create sets of up to m events when PMU has m counters
 - multiplex sets on actual PMU HW
 - global counts **approximated** by simple scaling calculation
 - higher switch rate = smaller blind spots = higher overhead
- Kernel support needed to minimize overhead
 - switching always occur in context of the monitored thread

Event sets and multiplexing (2)

- Each set encapsulates the full PMU state
 - identified by user-specified unique number (0-65535)
 - sets are placed in ordered list based on their unique number
- Timeout-based switching
 - granularity depends on kernel timer tick (HZ)
 - actual vs. requested timeout is reported to user
- Overflow-based switching
 - after threshold of n overflows of a counter
 - threshold specified per counter and per set
 - can be used to build cascading counters
- Switch mode determined per set, can mix & match
- Works with counting **and** sampling

PMU description module

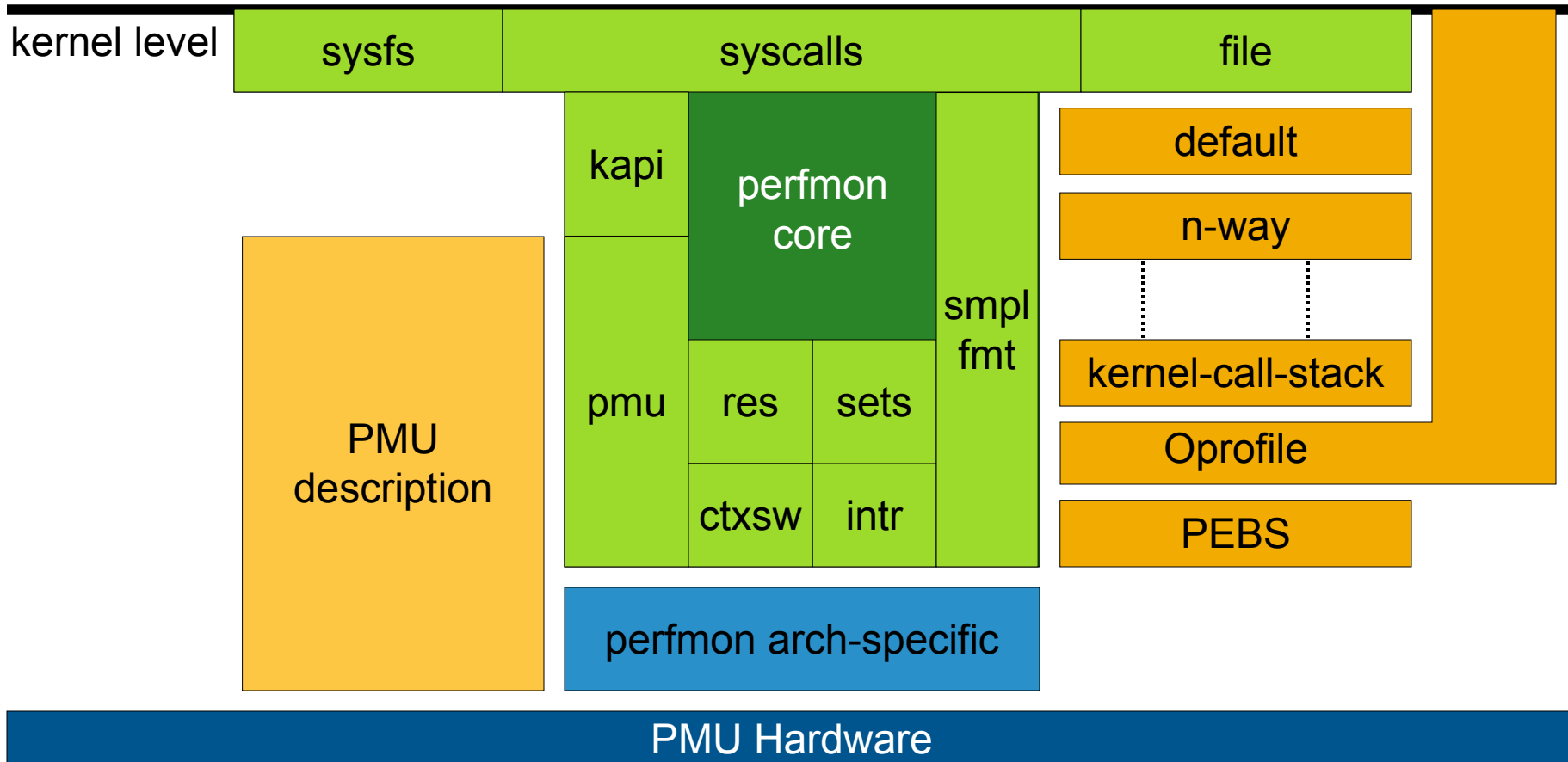
- Controls logical \Rightarrow actual PMU register mappings
- PMC and PMD mapping description tables
 - PMC: type, logical name, default value, reserved bit fields
 - PMC may have write-checker callbacks
- Arch-specific table contains actual location
 - e.g., Pentium M: PMC0 \Rightarrow PERFEVTSEL0 \Rightarrow MSR@0x186
- Implemented by kernel module
 - auto-loading on context creation, auto-probing on insertion
 - easier for: support of new HW, maintenance

```
$ cat /sys/kernel/perfmon/pmu_desc/mappings
PMC0:0x100000:0xffcffffff:PERFEVTSEL0
PMC1:0x100000:0xffcffffff:PERFEVTSEL1
PMD0:0x0:0xffffffffffffffff:PERFCTR0
PMD1:0x0:0xffffffffffffffff:PERFCTR1
```

Perfmon2 architecture summary



user level



Security

- Cannot assume tools are well-behaved
- Vector arguments, sampling buffers have max. size
 - tuneable via `/sys`
- Per-thread and system-wide contexts
 - can only attach to thread owned by caller
 - each context type can be limited to a users group (via `/sys`)
- Reading of PMU registers
 - direct access (some arch): limited to self-monitoring
 - interface access: can only read registers declared used
- PMU interrupt flooding
 - enough blind spots to ensure kernel makes forward progress
 - need to add interrupt throttling mechanism

Kernel-level perfmon2 interface (KAPI)

- Address need to use perfmon2 interface from kernel
 - e.g., SystemTap, scheduler
- Cannot use system call interface
 - no task, no file descriptor, no signal, no `read()`, no `mmap()`
- File descriptor replaced with opaque descriptor (pointer)
- Wait for notification using a `completion` structure
- Limited to system-wide, other functionalities supported

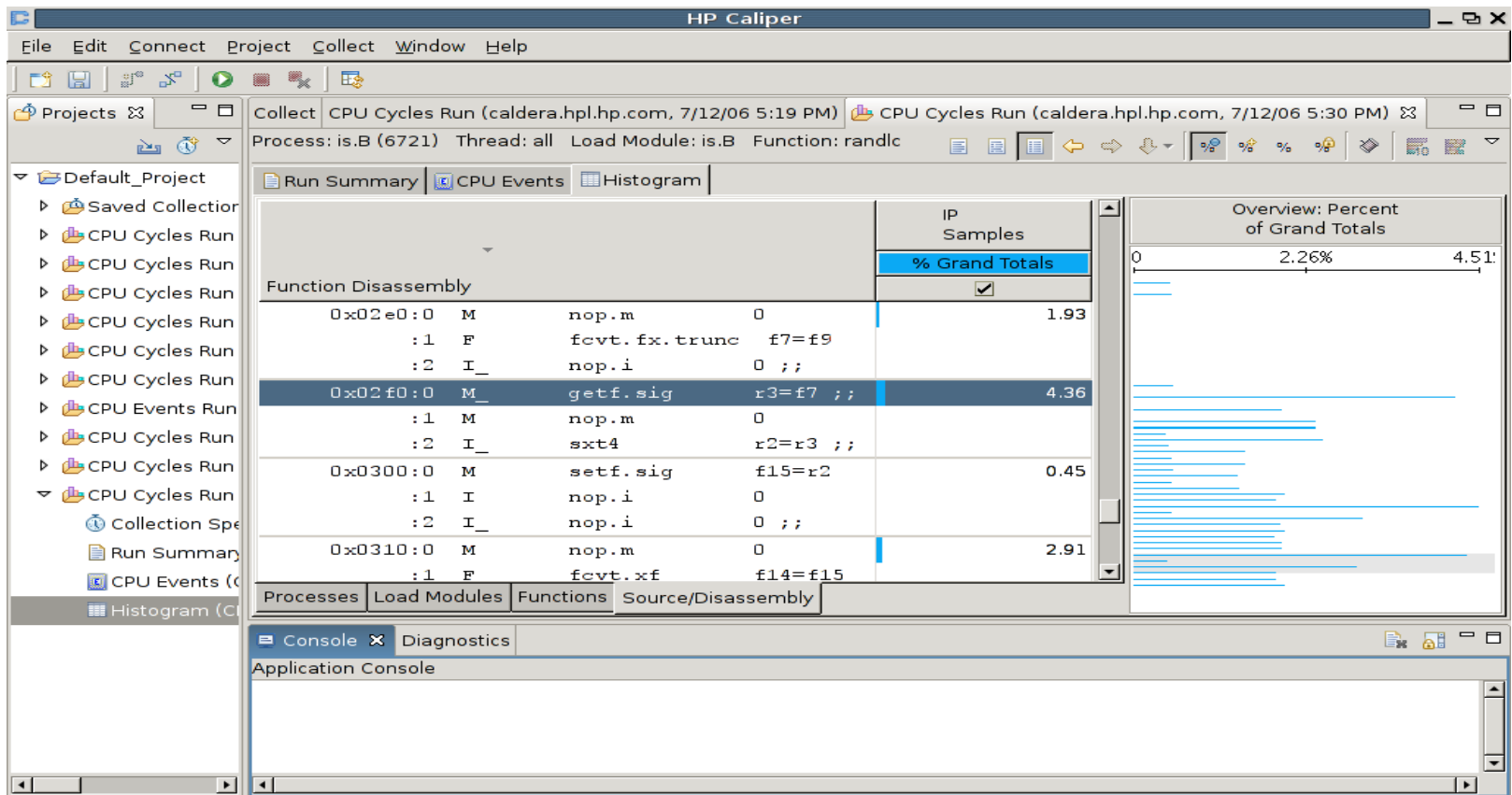
```
int pfmk_create_context(pfarg_ctx_t *ct, void *a, size_t s, struct completion *c, void **b, void **d)
int pfmk_write_pmcs(void *d, pfarg_pmc_t *pmcs, int n);
int pfmk_write_pmds(void *d, pfarg_pmd_t *pmcs, int n);
int pfmk_read_pmds(void *d, pfarg_pmd_t *pmcs, int n);
int pfmk_load_context(void *d, pfarg_load_t *ld);
int pfmk_start(void *d, pfarg_start_t *st);
int pfmk_stop(void *d);
int pfmk_restart(void *d);
int pfmk_create_evtsets(void *d, pfarg_setdesc_t *st, int n);
int pfmk_delete_evtsets(void *d, pfarg_setdesc_t *st, int n);
int pfmk_getinfo_evtsets(void *d, pfarg_setinfo_t *it, int n);
int pfmk_unload_context(void *d);
pfmk_close(void *d)
```

Status

- Perfmon2 v2.0 (IA-64 ONLY)
 - Included in SLES9 and RHEL4
 - no event sets, no PMU descriptions, `perfmonctl()` syscall
- Perfmon2 v2.2
 - compatibility layer with v2.0 on Itanium processors
 - available as a kernel patch for now, merging with mainline
- Supported processors for v2.2
 - all Itanium processors, incl. Montecito
 - Intel P6 (PII, PIII, Pentium M), architected IA-32 (Core Duo/Solo)
 - Intel Pentium 4 (32-bit, EM64T), incl. HyperThreading, PEBS
 - AMD64: all models, 32-bit and 64-bit modes
 - MIPS: 5K, 20K (Phil Mucci)
 - PowerPC: Power5 very experimental (IBM)

Tools using the interface (1)

- Caliper(HP) 4.0 (free for non commercial use)
 - Per-thread, source level profiles, preset metrics, IA-64 ONLY
 - Java-based standalone GUI (local, remote) or Eclipse plug-in



The screenshot displays the HP Caliper application window. The main area shows a 'Function Disassembly' table with columns for address, instruction type, instruction, and IP Samples. The 'getf.sig' function is highlighted, showing 4.36 IP samples. To the right, an 'Overview: Percent of Grand Totals' bar chart shows the relative contribution of various functions.

| Function Disassembly | | | | IP Samples |
|----------------------|----|---------------------|----------|----------------|
| | | | | % Grand Totals |
| 0x02e0:0 | M | nop.m | 0 | 1.93 |
| :1 | F | fcvt.fx.trunc f7=f9 | | |
| :2 | I_ | nop.i | 0 ;; | |
| 0x02f0:0 | M_ | getf.sig | r3=f7 ;; | 4.36 |
| :1 | M | nop.m | 0 | |
| :2 | I_ | sxt4 | r2=r3 ;; | |
| 0x0300:0 | M | setf.sig | f15=r2 | 0.45 |
| :1 | I | nop.i | 0 | |
| :2 | I_ | nop.i | 0 ;; | |
| 0x0310:0 | M | nop.m | 0 | 2.91 |
| :1 | F | fcvt.xf | f14=f15 | |

Tools using the interface (2)

- BEA JRockit 1.4.2 for Linux/ia64:
 - Dynamic Profile Guided Optimization (DPGO)
- PAPI toolkit (U. of Tennessee)
 - popular toolkit to write portable monitoring tools
- pfmon/libpfm 3.2 (HP Labs) (GPL/MIT)
 - pfmon: count, collect profiles per-thread or system-wide
 - libpfm helper library: what to measure \Rightarrow values of PMC
 - all IA-64 features: opcode match, DEAR, BTB, range restrictions
 - supports for AMD64, P6
 - adding Pentium 4 (IBM), MIPS (Phil Mucci)
- q-tools-0.3 for Linux/ia64 (HP Labs) (GPL)
 - qtools: statistical system-wide gprof replacement (user, kernel)

Current and future activities

- Mainline integration
 - submitted for review on lkml
 - lots of cleanups, simplifications, bug fixes, perf. improvements
- Oprofile
 - ensure continuity of service on non-IA64 HW
- Improving Pentium 4 user level support
- keep up-to-date with new HW, e.g., Core 2 CPUs
- Xen support for PMU virtualization
- PMU arbitration layer
 - allows for better sharing of PMU resources between tools
 - removes mutual exclusion between system-wide and per-thread

Conclusions

- Monitoring is key to achieve world-class performance
 - only way to understand how SW and HW interact
- Perfmon2 is the most advanced monitoring interface
 - custom sampling formats, event set multiplexing, PMU desc.
- Perfmon2 is portable and very extensible
- Built a strong community of users, contributors
 - Intel, AMD, IBM, HP, SGI, Redhat, various universities and individuals
- Once in mainline, expect that better tools will emerge

visit our website at: <http://perfmon2.sourceforge.net>



i n v e n t