(some of the) Linux performance tool research at HP Labs

David Mosberger
HP Labs, Palo Alto
CERN, Oct. 20, 2004

© 2004 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice

Outline

- Background
- Vision & approach
- Recent work
  - perfmon
  - Qprof
  - q-syscollect & q-view
    - statistical call-count collection
    - blind-spot-free profiling
- Summary
- Q & A

Background

- Goal is to give an overview of some of the performance tool research going on at HP Labs
- Material presented here covers work by:
  - Hans Boehm <Hans.Boehm@hp.com>
  - Stéphane Eranian <eranian@hpl.hp.com>
  - David Mosberger <davidm@hpl.hp.com>
- Some of this work is in support of GELATO:

http://www.gelato.org/

– a great resource for institutions interested in the advancement of Itanium Linux, including UCB, CERN, CSCS, INRIA, UNSW, and many others

Our vision

"radically simplify the development of efficient software..."

Why is this important?

hp invent

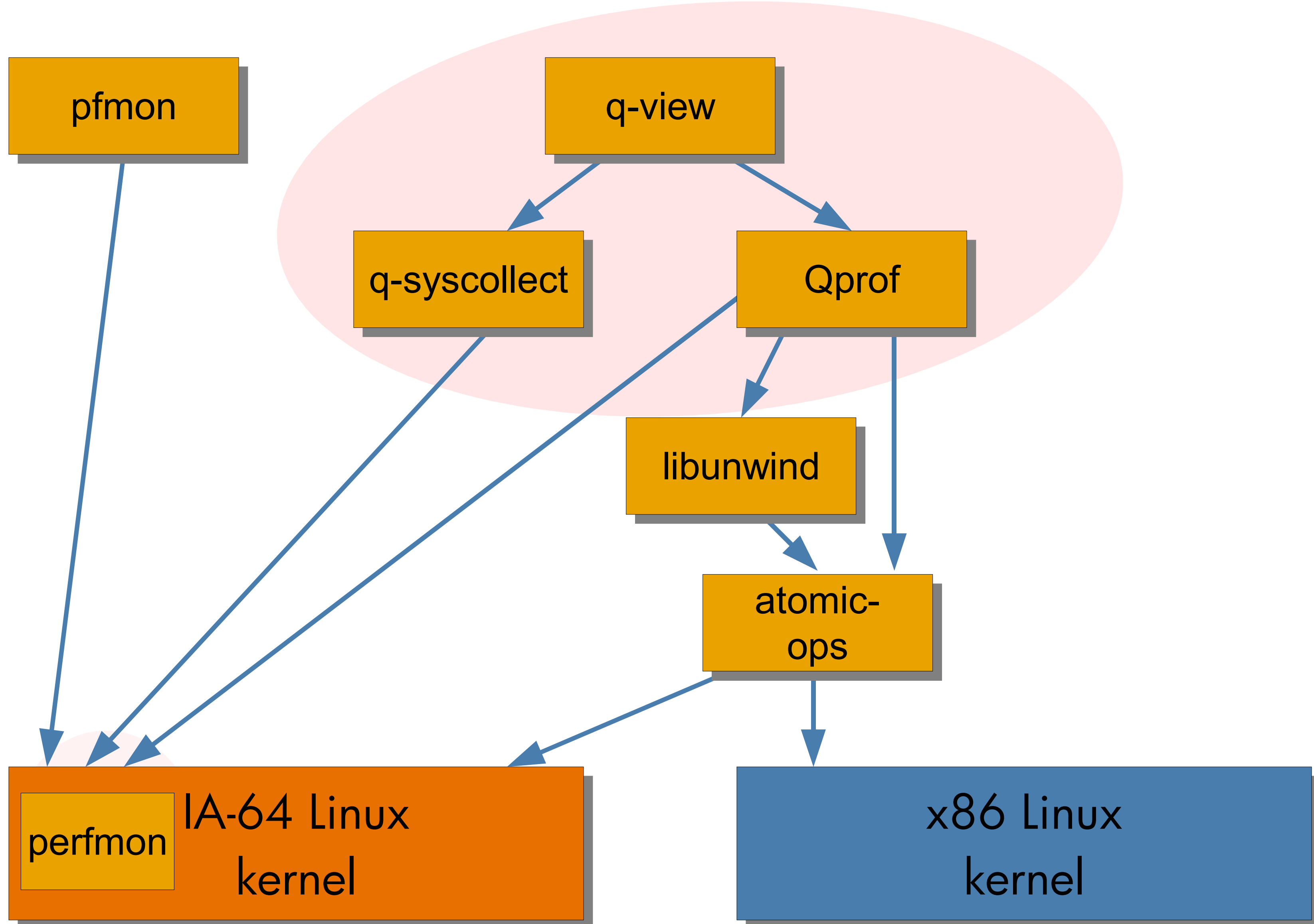
- More efficient software lets you do more on the same machine
  - great for servers
- More efficient software lets you do the same with less memory, less power, or with less waiting
  - great for workstations/desktops
- Many programmers spend a great deal of time tuning their programs; making this more efficient can greatly reduce software-development costs

Why is it hard?

hp invent

- Scope for radical simplification is huge:
  - HPC- (think Fortran) or UNIX-applications (C & C++)
  - managed runtime-environments (MREs; Java, Mono, ...)
  - single machine (small/large) and clustered environments
  - on-line vs. off-line optimization, etc.
- We focus on a manageable subset of this scope and use a bottom-up approach:
  - single machine (not clusters, yet)
  - C & C++ apps
  - MREs & on-line optimization in the not too distant future

Some of what we have so far…



The perfmon kernel sub-system

What is perfmon?

- An architecture-independent kernel-interface for performance-monitoring:
  - one interface that can support all performance-tools

```
int perfmonctl (int fd, int cmd, void *arg, int nargs)
```

PFM_CREATE_CONTEXT	PFM_READ_PMDS	PFM_START
PFM_WRITE_PMCS	PFM_LOAD_CONTEXT	PFM_STOP
PFM_WRITE_PMDS	PFM_UNLOAD_CONTEXT	PFM_RESTART

  - minimalistic: user-level libraries for complicated stuff
  - support per-process and system-wide monitoring
  - support sampling, not just event-counting
  - built-in, efficient, robust, secure, & documented
- A complete implementation for IA-64 Linux kernel:
  - supports all features of Itanium, Itanium 2, and future CPUs

Who is/could be using perfmon?

- pfmon (is)
  - gives raw access to all perfmon features
- PAPI (is)
  - gives abstract access to event-counters
- Oprofile (is)
  - can use event-counters as profiling-sources
- Qprof (is)
  - likewise
- q-syscollect (is)
  - uses perfmon to obtain code profile & call-counts
- Vtune (could)

Perfmon: a simple example

```
$ pfmon --follow-all --us-c \
   -ecpu_cycles,ia64_inst_retired -- \
     cc hello.c

15,642,024 CPU_CYCLES         /usr/lib/.../cc1
27,346,418 IA64_INST_RETIRED /usr/lib/.../cc1
 4,411,048 CPU_CYCLES         as
 5,484,922 IA64_INST_RETIRED as
27,172,698 CPU_CYCLES         /usr/bin/ld
33,930,949 IA64_INST_RETIRED /usr/bin/ld
   415,230 CPU_CYCLES         /usr/.../collect2
   507,735 IA64_INST_RETIRED /usr/.../collect2
   814,656 CPU_CYCLES         cc
 1,150,182 IA64_INST_RETIRED cc
```

Qprof

Qprof: removing the excuse not to profile

- No recompilation
- No relinking
- No kernel-modules
- User-installable

- Portable to IA-64, x86, Alpha Linux
- Supports threads & shared libraries
- More functionality if installed with
  - perfmon: profile on cache-misses, ...
  - libunwind: call-stack profiling
- Use by itself or with q-view

Implementation techniques:
- Relies on dynamic linking
- Preloaded library sets up timer, signal-handler looks at IP
- Intercepts some library-routines (e.g., *pthread_create()*)

Research interest:
- demands interesting & practical lock-free data-structures

Qprof installation & use

```
$ wget hpl.hp.com/.../linux/qprof-0.4.tar.gz
$ tar xzvf q-prof-0.4.tar.gz
$ cd qprof-0.4
$ make install
$ export QPROF_GRANULARITY=function
$ export QPROF_COLOR=red QPROF_INTERVAL=1000
$ . alias.sh
$ qprof_start
$ du -h -s $HOME
...
du(__strtol_internal)                    1    (  3%)
libc.so.6(strlen)                        3    (  9%)
libc.so.6(__lxstat64)                   17    ( 49%)
libc.so.6(__libc_open64)                 2    (  6%)
...
```

q-syscollect + q-view = gprof without the pain

q-syscollect + q-view

- No recompilation
- No relinking
- No kernel-modules
- No dyn. loader tricks
- Itanium 2-specific

- 100% safe
- Supports threads & shared libraries
- Can monitor kernel-level exection (even at lowest-level, such as TLB-miss handler)
- Separates data-collection (q-syscollect) from data visualization (q-view)

Implementation techniques:

- Relies on perfmon to collect data on all processes and kernel

Research interest:

- exploits Itanium 2 BTB to collect call-counts *statistically*

q-syscollect + q-view in action

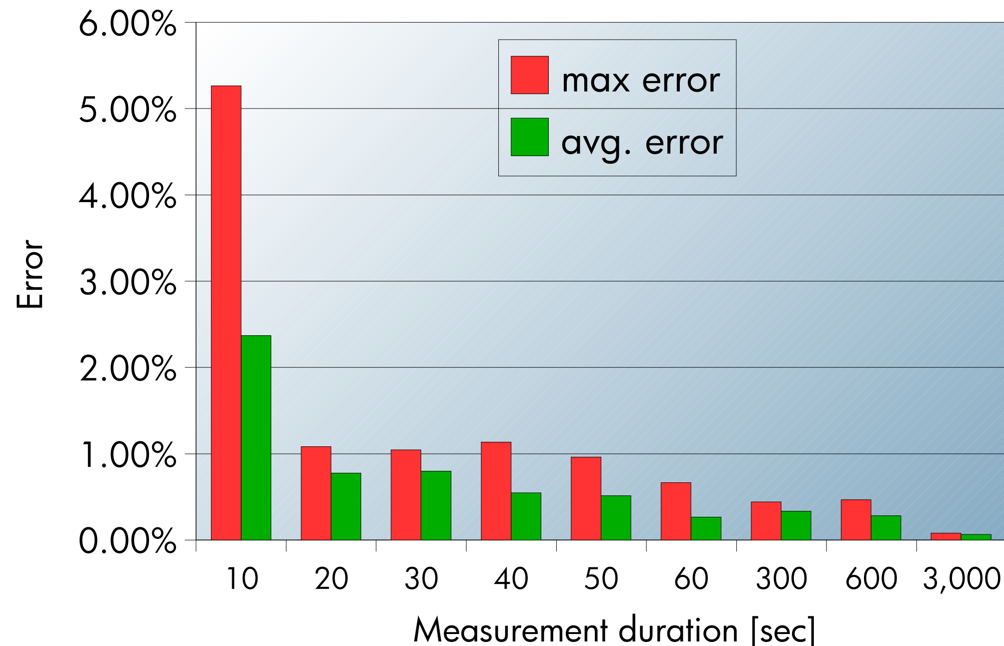
```
$ cc -O2 tst.c -o tst
$ q-syscollect tst
$ ls .q
  tst-pid19.edge   tst-pid19.hist   tst-pid19.info
xterm-pid34.edge xterm-pid34.hist xterm-pid34.info
$ q-view .q/tst-pid19.info
% time   self   cumul   calls self/call tot/call name
 36.75   7.33    7.33    120M     61.2n    61.2n cos
 10.46   2.08    9.41   29.3M     71.2n    71.2n tan
  8.91   1.78   11.19       -         -        - main
  5.88   1.17   13.67   29.6M     39.7n     163n f08
    ...
Call-graph table:
index %time         self  children called name
                    3.61      0.00 59.1M     f08 [25]
                    1.84      0.00 30.0M     f07 [26]
                    1.88      0.00 30.7M     f06 [27]
[30]   39.0         7.33      0.00  120M  cos
```

q-syscollect: How does it work?

- Flat profile:
  - Obtained in standard fashion
    - sample instruction-pointer (IP) every *N* ticks
    - “tick” can be any PMU event (CPU--cycles, stalls, TLB-misses, etc.)
- Call-graph:
  - Take advantage of advanced PMU features:
    - Branch-Trace-Buffer (BTB) configured to record return-branches *only*
    - Every *M*-th return-branch:
      - stop the BTB
      - read out the branch-address and target of 4 most recent returns
      - record info in a 2-dimensional histogram
      - resume normal execution
    - Randomize on *M* to avoid (serious) bias

Are statistical call-counts accurate?

- They appear to be
  - empirically, found to be accurate even for relatively short runs (20-30 sec) and complex call-graphs
  - error analysis for loop calling 10 functions:



New feature in q-syscollect v0.2: blind-spot-free profiling

- The challenge:
  - When interrupts are masked/disabled, the PMU interrupt can't get through ⇒ *blind spots*
  - Example:
    - kernel-profile for signal-delivery benchmark:

```
% time   self calls self/call name
 25.05 12.44 88.8M      140n _spin_unlock_irq
 17.00  8.44 59.8M      141n _spin_unlock_irqrestore
  8.54  4.24  231M     18.4n __copy_user
  8.36  4.15     -         - break_fault
  7.52  3.73 89.0M     42.0n __do_clear_user
  3.84  1.91 29.3M     65.1n setup_sigcontext
  2.85  1.42 29.8M     47.6n setup_frame
```

Blind-spot-free profiling (cont.)

- Possible solutions:
  - Non-Maskable Interrupt:
    - can be dangerous; doesn't help with blind-spots created by low-level handlers such as software TLB-miss-handlers
    - on Itanium, it can be masked via PSR.I
  - INIT events:
    - Itanium-specific, truly non-maskable event, but expensive:
      - goes through PAL & SAL firmware layers and executes in physical mode
  - q-syscollect approach:
    - Take advantage of avanced PMU features:
      - sample branches in the BTB and use the sampled info to determine most recently executing basic-block

Blind-spot-free profiling: some caveats

- Due to the BTB-based collection of IP-samples:
  - Profiling granularity limited to basic-block level
    - Not a problem for function-level profiling
  - Qualitatively accurate results with standard kernel, quantitative accurate results need a small kernel patch
  - BTB can sample at most one location per sampling period ⇒ sampling period must be greater than the longest period for which interrupts are disabled
  - Same hardware (BTB) is used for call-count collection and blind-spot-free profiling ⇒ can only do one at a time
    - future perfmon supports multiplexing to avoid this limitation

Blind-spot-free profiling: result for signal-delivery benchmark

- Collected with:
  - q-syscollect -k -i -C 100, with kernel-patch applied:

```
% time self calls self/call name
  7.17 4.05      -         - __copy_user
  6.63 3.75      -         - break_fault
  6.31 3.57      -         - __do_clear_user
  5.80 3.28      -         - recalc_sigpending_tsk
  4.75 2.69      -         - rse_clear_invalid
  3.89 2.20      -         - __dequeue_signal
  3.23 1.83      -         - ia64_leave_kernel
```

  - without kernel-patch:
    - highest-ranked 6 functions remain the same, but time spent in `__copy_user` & `break_fault` drops significantly

Summary

- We are using a bottom-up approach to build up a suite of performance tools and related infrastructure (e.g., atomic-ops and libunwind)
- We use perfmon and the Itanium 2 performance monitoring unit to push the boundaries of what can be measured in a non-intrusive manner
- Along the way we have built some handy and powerful performance-tools, though we're not claiming production-quality (with the exception of perfmon)
- Watch out for future developments in this area...

For further info...

http://www.hpl.hp.com/research/linux/

hp
invent