Access the performance-counter on Ubuntu/Linux

Performance Counters for Linux (PCL) is a new kernel-based subsystem that provides a framework for collecting and analyzing performance data. These events will vary based on the performance monitoring hardware and the software configuration of the system. Linux includes this kernel subsystem to collect data and the user-space tool perf to analyze the collected performance data.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler (or code profiler).

Linux hardware performance measurement using counters, trace-points, software performance counters, and dynamic probes. Perf as one of the two most commonly used performance counter profiling tools on Linux. Perf basically use to analyses the core internal bottleneck right up to the driver level.

Linux support many profiling tools like perf, trace-cmd, blktrace, strace and oprofile.

Below steps were tested on the XU3/XU4 platforms.

Build Pref tool

In order to build perf you need to install following pakages.

```
sudo apt-get install flex bison libdw-dev libnewt-dev binutils-dev libaudit-dev libgtk2.0-dev libperl-dev libpython-dev libunwind-*
```

Ensure the following kernel config options are enabled

```
$ zegrep "CONFIG_PERF_EVENTS|CONFIG_HW_PERF_EVENTS" /proc/config.gz
CONFIG_PERF_EVENTS=y
CONFIG_HW_PERF_EVENTS=y
```

CONFIG_PERF_EVENTS	support for various performance events provided by software and hardware.
CONFIG_HW_PERF_EVENTS	support enables hardware performance counter support for perf events.
CONFIG_CGROUP_PERF	perf_event per-cpu per-container group (cgroup) monitoring.
CONFIG_PERF_USE_VMALLOC	some architectures that have d-cache aliasing issues, such as Sparc and ARM, should select PERF_USE_VMALLOC in order to avoid these for perf mmap.
CONFIG_FRAME_POINTER	kernel image will be slightly larger and slower, but it will give very useful debugging information
CONFIG_KALLSYMS	kernel print out symbolic crash information and symbolic stack backtraces.

CONFIG_TRACEPOINTS	Kernel event like usb, wireless event, malie event can handled using tracepoint.
CONFIG_FTRACE	Kernel ftrace is an internal tracer designed to help out developers and designers of systems to find what is going on inside the kernel.
CONFIG_KPROBES	Events are similar to tracepoint based events
CONFIG_KPROBE_EVENTS	
CONFIG_UPROBES	Uprobe based trace events are similar to kprobe based trace events.
CONFIG_UPROBE_EVENTS	
CONFIG_LOCK_STAT	Kernel lock stats is required.
CONFIG_DEBUG_INFO	Kernel enable debug info.

Enable PMU setting in Device tree Perf events can be enable in the exynos5422_evt0.dtsi.

```
root@odroidcu4n:/usr/src/odroidxu3-3kr# git diff
arch/arm/boot/dts/exynos5422_evt0.dtsi
diff --git a/arch/arm/boot/dts/exynos5422_evt0.dtsi
b/arch/arm/boot/dts/exynos5422_evt0.dtsi
index 2f9c95d..74c7c7e 100755
--- a/arch/arm/boot/dts/exynos5422_evt0.dtsi
+++ b/arch/arm/boot/dts/exynos5422_evt0.dtsi
@@ -91,6 +91,17 @@
                };
        };

+       arm-pmu {
+               /* compatible = "arm,cortex-a15-pmu";
+                  interrupt-parent = <&combiner>;
+                  interrupts = <1 2>, <7 >, <16 6>, <19 2>;
+                */
+               ompatible = "arm,cortex-a7-pmu";
+               interrupt-parent = <&gic>;
+               interrupts = < 192 4>, < 193 4>, < 194 4>, < 195 4>;
+       };
+
        watchdog@10020000 {
                compatible = "samsung,s3c2410-wdt";
                reg = <0x101D0000 0x100>;
```

Note: Perf counters for arm,cotrex-a7-pmu is supported in the kernel. But not for arm,cortex-a15-pmu.

Once kernel is build using above config flags. Perf could be install using following command. If the distribution supports it.

```
$ sudo apt-get install linux-tools
```

Otherwise you need to build the took from the kernel source code tree.

getting the source

```
$ git clone --depth 1 https://github.com/hardkernel/linux.git -b
odroidxu3-3.10.y
$ cd linux/tools/perf
$ make -j `getconf _NPROCESSORS_ONLN` perf
```

Install perf tool.

```
$ cp perf /usr/bin/
```

test perf is installed and running.

```
$ perf list

List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                                [Hardware event]
  instructions                                        [Hardware event]
  cache-references                                    [Hardware event]
  cache-misses                                        [Hardware event]
  branch-instructions OR branches                     [Hardware event]
  branch-misses                                       [Hardware event]
  bus-cycles                                          [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend     [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend       [Hardware event]
  ref-cycles                                          [Hardware event]

  cpu-clock                                           [Software event]
  task-clock                                          [Software event]
  page-faults OR faults                               [Software event]
  context-switches OR cs                              [Software event]
  cpu-migrations OR migrations                        [Software event]
  minor-faults                                        [Software event]
  major-faults                                        [Software event]
  .
  .
  .
```

Perf tool Features

perf tool measures the performance of the application and trace down to the kernel event that got triggers.

perf help

```
root@odroidcu4n:~# perf --help

 usage: perf [--version] [--help] COMMAND [ARGS]

 The most commonly used perf commands are:
```

```
   annotate        Read perf.data (created by perf record) and display
annotated
code
   archive         Create archive with object files with build-ids found in
perf
.data file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   diff            Read two perf.data files and display the differential
profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional
informati
on
   kmem            Tool to trace/measure kernel memory(slab) properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the
profi
le
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace
out
put
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   trace           strace inspired tool
   probe           Define new dynamic tracepoints

 See 'perf help COMMAND' for more information on a specific command.
```

perf is used with several sub commands:

```
    stat:   This perf command provides overall statistics for common
performance events,
                including instructions executed and clock cycles consumed.
                Options allow selection of events other than the default
measurement events.
    top:    This perf command help monitor top-like dynamic view of hottest
functions.
    record:     This perf command records performance data into a file which
can be later analyzed using perf report.
    report:     This perf command reads the performance data from a file and
```

```
analyzes the recorded data.
    list:   This perf command lists the events available on a particular
machine.
```

Perf Example

Perf stats:

```
~# perf stat -B dd if=/dev/zero of=/dev/null count=1000000
1000000+ records in
1000000+ records out
512000000 bytes (512 MB) copied, 1.41271 s, 362 MB/s

 Performance counter stats for 'dd if=/dev/zero of=/dev/null count=1000000':

       1414.727540 task-clock                #    0.998 CPUs utilized
                10 context-switches          #    0.007 K/sec
                 0 cpu-migrations            #    0.000 K/sec
               148 page-faults               #    0.105 K/sec
   <not supported> cycles
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
   <not supported> instructions
   <not supported> branches
   <not supported> branch-misses

       1.416998532 seconds time elapsed
```

Node: perf stats provide summary of the kernel events.

Perf top:

Suppose we want to run-time analyses of the kernel events just like user space top/htop use below command.

perf top -z

```
~# perf top -z
Samples: 127K of event 'cpu-clock', Event count (approx.): 3315322731
 98.92%  [kernel].head.text      [k] 0xc0023f10
  0.08%  perf                    [.] sort__dso_cmp
  0.07%  perf                    [.] perf_top__mmap_read_idx
  0.06%  perf                    [.] perf_evsel__parse_sample
  0.06%  libc-2.21.so            [.] memset
  0.06%  perf                    [.] perf_evlist__mmap_read
  0.05%  perf                    [.] add_hist_entry.isra.8
  0.05%  libpthread-2.21.so      [.] pthread_mutex_lock
  0.05%  libslang.so.2.3.0       [.] 0x000630ce
  0.04%  perf                    [.] perf_event__preprocess_sample
```

```
  0.03%  libpthread-2.21.so      [.] __pthread_mutex_unlock_usercnt
  0.03%  libslang.so.2.3.0       [.] SLsmg_write_chars
  0.03%  perf                    [.] symbols__insert
  0.03%  libc-2.21.so            [.] __libc_calloc
  0.03%  perf                    [.] maps__find
  0.03%  libc-2.21.so            [.] strstr
  0.02%  libc-2.21.so            [.] strcmp
  0.02%  perf                    [.] dump_printf
  0.02%  perf                    [.] dso__find_symbol
  0.01%  perf                    [.] symbol_filter
```

```
Note: You can watch all the supported list events on the perf top.
```

perf top -z -e task-clock

```
~# perf top -z -e task-clock

 Samples: 1M of event 'task-clock', Event count (approx.): 2069462202
99.22%  [kernel].head.text          [k] 0xc0023f10
 0.06%  perf                        [.] sort__dso_cmp
 0.06%  libc-2.21.so                [.] memset
 0.06%  perf                        [.] add_hist_entry.isra.8
 0.06%  perf                        [.] perf_evlist__mmap_read
 0.06%  perf                        [.] perf_top__mmap_read_idx
 0.05%  perf                        [.] perf_evsel__parse_sample
 0.04%  perf                        [.] perf_event__preprocess_sample
 0.04%  libpthread-2.21.so          [.] pthread_mutex_lock
 0.03%  libslang.so.2.3.0           [.] SLsmg_write_chars
 0.03%  libpthread-2.21.so          [.] __pthread_mutex_unlock_usercnt
 0.03%  perf                        [.] maps__find
 0.03%  libslang.so.2.3.0           [.] 0x000548d2
 0.03%  perf                        [.] dump_printf
 0.03%  libc-2.21.so                [.] strcmp
```

Perf record

perf record is use to record the kernel event into perf.data, that file can then be analyzed, possibly on another machine, using the perf report and perf annotate commands.

```
~# perf record dd if=/dev/zero of=/dev/null count=1000000
1000000+ records in
1000000+ records out
512000000 bytes (512 MB) copied, 1.50392 s, 340 MB/s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.232 MB perf.data (~10121 samples) ]
~#
~# ls perf.data
perf.data
```

Perf report

Samples collected by perf record are saved into a binary file called, by default, perf.data. The perf report command reads this file and generates a concise execution profile.

<codeb> Samples: 6K of event 'cpu-clock', Event count (approx.): 1503750000 13.68% dd [kernel.kallsyms] [k] vector_swi 10.97% dd [kernel.kallsyms] [k] lock_acquire

```
9.64%  dd  dd                 [.] 0x000020fc
7.45%  dd  [kernel.kallsyms]  [k] __srcu_read_lock
7.27%  dd  [kernel.kallsyms]  [k] __srcu_read_unlock
6.42%  dd  [kernel.kallsyms]  [k] __clear_user_std
6.12%  dd  [kernel.kallsyms]  [k] lock_release
4.66%  dd  libc-2.21.so       [.] __GI___libc_read
4.59%  dd  libc-2.21.so       [.] __GI___libc_write
4.24%  dd  [kernel.kallsyms]  [k] fsnotify
3.14%  dd  [kernel.kallsyms]  [k] vfs_write
2.81%  dd  [kernel.kallsyms]  [k] vfs_read
2.79%  dd  libc-2.21.so       [.] __GI___memcpy_neon
2.76%  dd  [kernel.kallsyms]  [k] fget_light
1.78%  dd  [kernel.kallsyms]  [k] SyS_write
1.75%  dd  [kernel.kallsyms]  [k] SyS_read
1.68%  dd  [kernel.kallsyms]  [k] rw_verify_area
1.60%  dd  [kernel.kallsyms]  [k] ret_fast_syscall
1.36%  dd  [kernel.kallsyms]  [k] read_zero
1.10%  dd  [kernel.kallsyms]  [k] local_restart
1.08%  dd  [kernel.kallsyms]  [k] __fsnotify_parent
0.90%  dd  [kernel.kallsyms]  [k] debug_smp_processor_id
```

</code>

External Links

You can find more on following links.

https://perf.wiki.kernel.org/index.php/Tutorial

http://www.brendangregg.com/perf.html

http://forum.odroid.com/viewtopic.php?f=61&t=2393