Data type profiling support in perf infrastructure

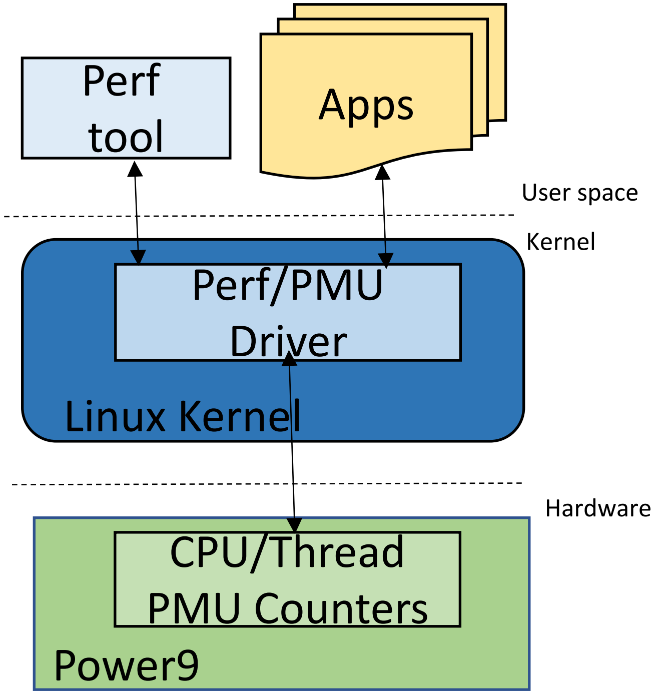
Athira Rajeev
(athirar.rajeev@gmail.cpm)
Linux Technology center @IBM

Agenda

- What is perf/PMU
    - Counting events
    - Profiling/sampling
    - Perf report/annotate
    - Perf mem record
- Data type profiling feature
    - Enablement in community by Namhyung
    - How it works
    - How it Is useful
- Enablement of data type feature in powerpc
    - Changes to enable this in powerpc
    - Current state, results and further plans

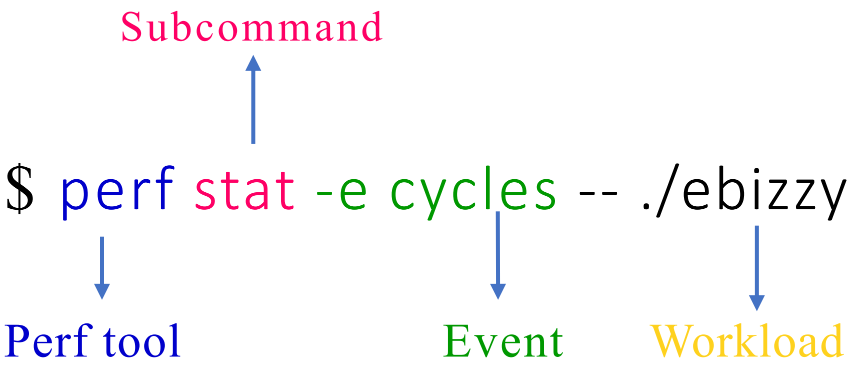
What is PMU/perf

- Counters: hardware/software units to count events.
  - Dedicated registers in processor for counting events : H/W counters
  - Variables in kernel for software events: S/W counters
- Performance Monitoring Unit (PMU)
  - Set of hardware counters built into core logic
  - Provides precise picture of CPU resource utilization
  - Instruments most of the core execution units



- Perf infrastructure has two main components
  - Perf kernel API (perf_event_open syscall)
  - Perf tool (user space tool, part of linux kernel source tree and supported by all Linux distro's)

Perf Counting



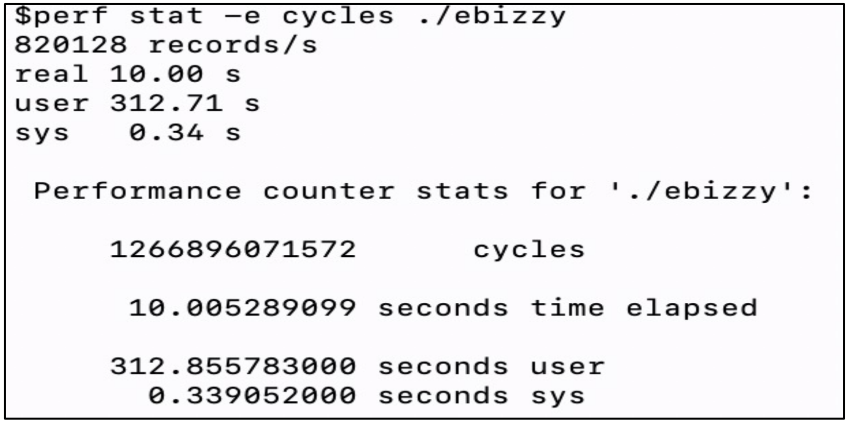
Subcommand

$ perf stat -e cycles -- ./ebizzy

Perf tool

Event

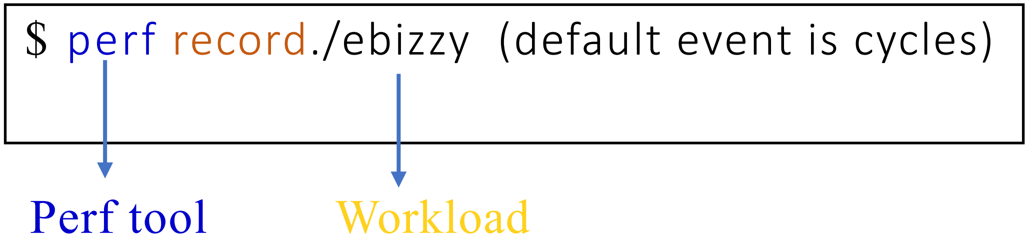
Workload



```
$perf stat –e cycles ./ebizzy
820128 records/s
real 10.00 s
user 312.71 s
sys   0.34 s

 Performance counter stats for './ebizzy':

     1266896071572       cycles

      10.005289099 seconds time elapsed

     312.855783000 seconds user
       0.339052000 seconds sys
```

Perf Sampling

- Ability to look at an instruction throughout its life-cycle in the pipeline
- When counter overflows, capture sample and saves in "perf.data"
- On PMI, sample details captured includes:
    - Instruction/data address, branch entries, call-graphs
- Useful to find hotspots in an application



$ perf record./ebizzy (default event is cycles)

Perf tool

Workload

Perf.data that will be referred to in further slides.

```
$ perf record -a sleep 20 --------> -a for system wide monitoring
[ perf record: Woken up 0 times to write data ]
[ perf record: Captured and wrote 637.382 MB perf.data (11916467 samples)
]
```

Perf report

- Reports samples recorded from "perf.data" file

  # perf report

```
# Samples: 11M of event 'cycles:P'
# Event count (approx.): 15056187404879
#
# Overhead  Command       Shared Object          Symbol
# ........  ............... .......................... ..................................................
#
   36.06%  ws1           [kernel.kallsyms]     [k] queued_spin_lock_slowpath
   27.51%  swapper       [kernel.kallsyms]     [k] __ppc64_runlatch_off
    5.99%  swapper       [kernel.kallsyms]     [k] enqueue_task_fair
    5.55%  ws1           [kernel.kallsyms]     [k] newidle_balance
    1.21%  ws1           [kernel.kallsyms]     [k] _raw_spin_lock_irq
    1.17%  ws1           [kernel.kallsyms]     [k] __schedule
    1.00%  ws1           [kernel.kallsyms]     [k] _raw_read_lock
    0.83%  ws1           [kernel.kallsyms]     [k] update_sg_lb_stats
    0.64%  swapper          [unknown]          [H] 0x00000000002aeddc
    0.60%  ws1           [kernel.kallsyms]     [k] do_dec_rlimit_put_ucounts
    0.60%  ws1           [kernel.kallsyms]     [k] update_cfs_group
    0.59%  swapper       [kernel.kallsyms]     [k] update_cfs_group
```

Maximum samples are from these top 4 functions

Perf annotate (enqueue_task_fair from perf.data)

- Support source code annotation
  - Drill down at instruction level

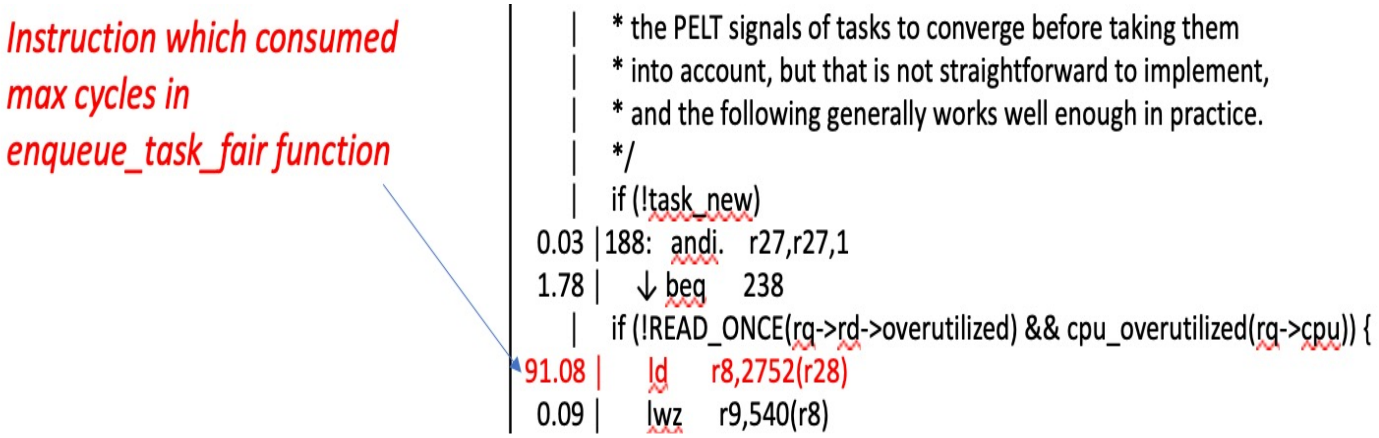
# perf report (annotated view)
Press 'a' on any sample

Instruction which consumed max cycles in enqueue_task_fair function

```
Samples: 11M of event 'cycles:P', 4000 Hz, Event count (approx.): 15056187404879
enqueue_task_fair  /lib/modules/6.8.0-rc6/build/vmlinux [Percent: local period]
  0.00 |      nop
       |    if (trace_sched_update_nr_running_tp_enabled()) {
       |    call_trace_sched_update_nr_running(rq, count);
       |    }
       |
       |    #ifdef CONFIG_SMP
       |    if (prev_nr < 2 && rq->nr_running >= 2) {
  0.00 |      cmplwi   r31,1
  0.05 |    ↓ ble      3a0
  0.00 |184:   nop
       |    * A better way of solving this problem would be to wait for
       |    * the PELT signals of tasks to converge before taking them
       |    * into account, but that is not straightforward to implement,
       |    * and the following generally works well enough in practice.
       |    */
       |    if (!task_new)
  0.03 |188:   andi.    r27,r27,1
  1.78 |    ↓ beq      238
       |    if (!READ_ONCE(rq->rd->overutilized) && cpu_overutilized(rq->cpu)) {
 91.08 |      ld       r8,2752(r28)
  0.09 |      lwz      r9,540(r8)
  0.00 |      cmpwi    r9,0
  0.21 |    ↓ bne      238
       |    unsigned long rq_util_min = uclamp_rq_get(cpu_rq(cpu),
UCLAMP_MIN);
```

Why data type profiling

- Perf annoate of enqueue_task_fair pointed to the instruction which consumed max cycles in that sample



Instruction which consumed max cycles in enqueue_task_fair function

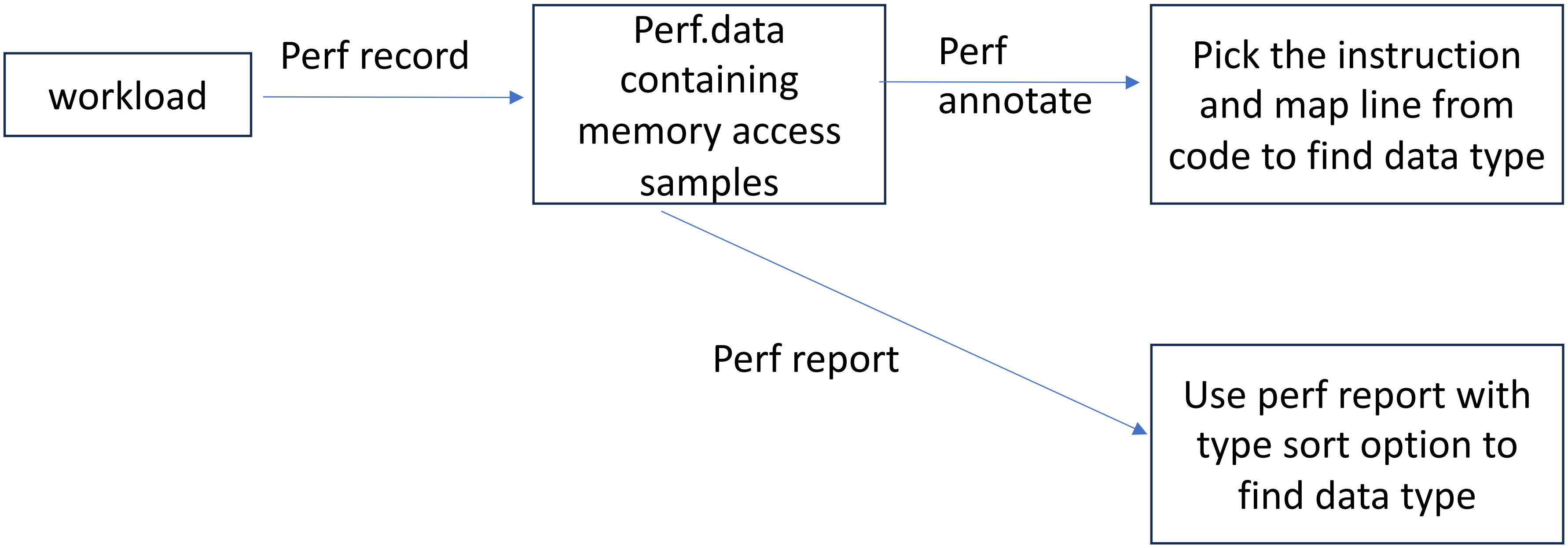
```
       |    * the PELT signals of tasks to converge before taking them
       |    * into account, but that is not straightforward to implement,
       |    * and the following generally works well enough in practice.
       |    */
       |   if (!task_new)
  0.03 |188:  andi.   r27,r27,1
  1.78 |    ↓ beq     238
       |   if (!READ_ONCE(rq->rd->overutilized) && cpu_overutilized(rq->cpu)) {
 91.08 |      ld      r8,2752(r28)
  0.09 |      lwz     r9,540(r8)
```

***Question**: What data type is being accessed here ? ex: basic data types like int or any struct)*

***Solution**: Use data type feature in perf to solve this without manually doing perf annotate and mapping in the code*

Data type profiling enablement in community

- Support in community added by Namhyung Kim
- Associate samples to Data type information
- Uses Dwarf debug information to retrieve the type info
- No change needed in kernel/application workload
- Needs memory access samples in perf.data file



workload

Perf record

Perf.data containing memory access samples

Perf annotate

Pick the instruction and map line from code to find data type

Perf report

Use perf report with type sort option to find data type

How to use

- Get the profile data (perf.data)
  - For more precision, use memory access events
  - If arch doesn't support mem events, use events which gives relevant memory access instructions in samples.
  - Needs kernel with debuginfo since it uses DWARF debug data
  - **$ perf mem record or $ perf record –e <event>**

- Use perf report/annotate to view the result
  - In perf report, use sort keys: type, typeoff
    - type : shows name of the data type
    - typeoff: shows name of the field in the data type
    - **$ perf report –s type,typeoff**
  - In annotate, use data-type option for data field level annotation
    - **$ perf annotate –data-type**

Support for data type profiling on powerpc architecture

- powerpc instruction nmemonic table to associate load/store instructions with move_ops which is use to identify if instruction is a memory access one.
- To get register number and access offset from the given instruction, tool uses fields from "struct arch" -> objump. Add entry for powerpc here.
- Add get_arch_regnum to return register number from the register name string.

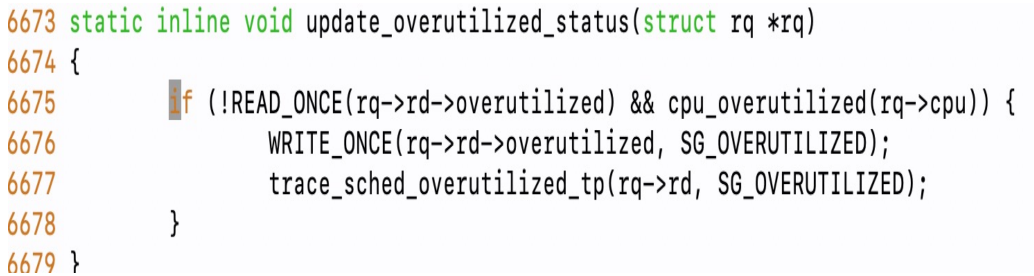
Patch in discussion in mailing list:

https://lore.kernel.org/linux-perf-users/20240309072513.9418-1-atrajeev@linux.vnet.ibm.com/T/#t

Identify data type for **enqueue_task_fair** function

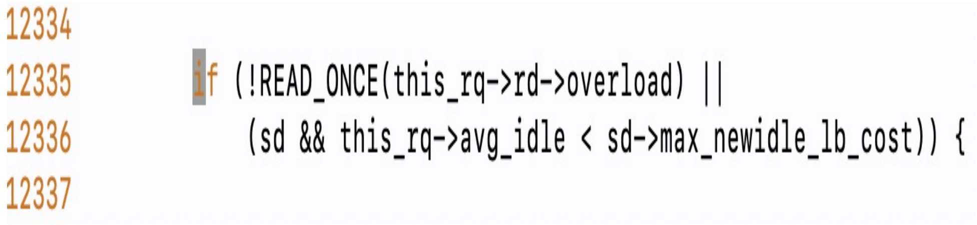
```
$ perf report -v -s symbol,type,typeoff

# Samples: 11M of event 'cycles:P'
# Event count (approx.): 15056187404879
#
# Overhead  Symbol                                   Data Type  Data Type Offset  IPC   [IPC Coverage]
# ........  .......................................  .........  ...............   ...................

  36.07%  0xc0000000000ad8f8 v [k] queued_spin_lock_slowpath        (unknown)  (unknown) +0 (no field)  -     -
  27.51%  0xc000000000020d14 v [k] __ppc64_runlatch_off          struct thread_info  struct thread_info +8 (local_flags)  -
   -
   5.76%  0xc0000000001bfe68 v [k] enqueue_task_fair            struct rq  struct rq +2752 (rd)  -     -
   3.03%  0xc0000000001c432c v [k] newidle_balance              struct rq  struct rq +2752 (rd)  -     -
   2.47%  0xc0000000001c4330 v [k] newidle_balance              struct rq  struct rq +2760 (sd)  -     -
```

```
# addr2line -f -e vmlinux -a 0xc0000000001bfe68
0xc0000000001bfe68
update_overutilized_status
/root/src/linux/kernel/sched/fair.c:6675
```



```
6673 static inline void update_overutilized_status(struct rq *rq)
6674 {
6675         if (!READ_ONCE(rq->rd->overutilized) && cpu_overutilized(rq->cpu)) {
6676                 WRITE_ONCE(rq->rd->overutilized, SG_OVERUTILIZED);
6677                 trace_sched_overutilized_tp(rq->rd, SG_OVERUTILIZED);
6678         }
6679 }
```

```
# addr2line -f -e vmlinux -a 0xc0000000001c432c
0xc0000000001c432c
newidle_balance
/root//src/linux/kernel/sched/fair.c:12335
```



```
12334
12335         if (!READ_ONCE(this_rq->rd->overload) ||
12336             (sd && this_rq->avg_idle < sd->max_newidle_lb_cost)) {
12337
```

Further work

- Complete the basic foundational patches
- Check X form instructions(current patches solves D form)
- Resolve the frame base address type variables in DWARF info
- Understand and Resolve remaining **unresolved ones** in the result
- Explore additional contributions that can be added to the feature

Backup

Perf mem events (Architecture specific)

- Provides information about sampled instruction
  - Useful for memory access analysis
  - Load latency analysis
  - Memory hierarchy (reload source)

Usage: To capture memory access In samples

```
# perf mem record <workload>
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.027 MB perf.data (6 samples) ]
```

Usage: To list memory access events for the specific architecture

```
# perf mem record –e list
```

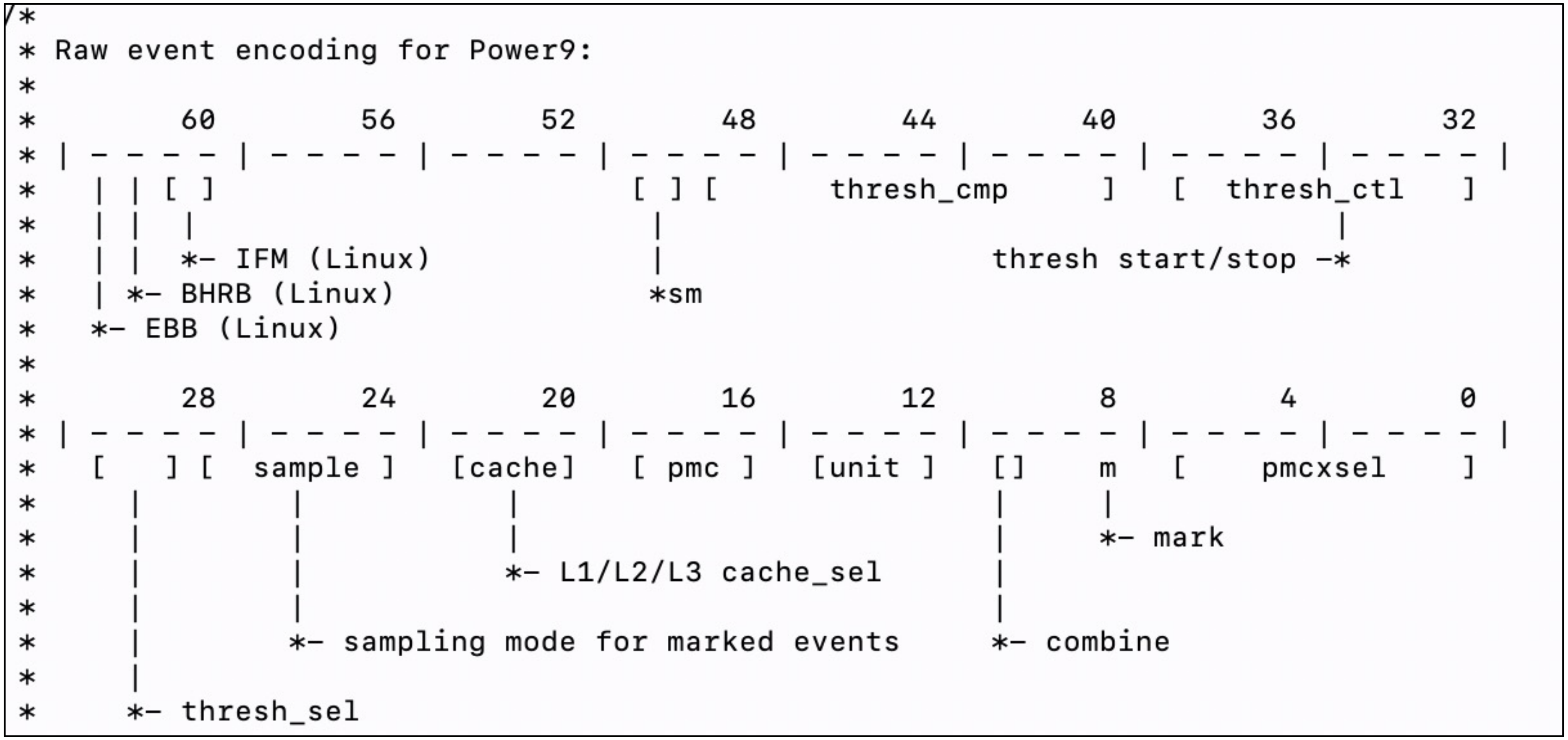
Usage: To capture specifically loads or stores

```
# perf record –e mem-loads –a  -----> capture memory loads
# perf record –e mem-stores –a -----> capture memory stores
```

Raw Hardware Events

- Hardwares typically support several more events than the kernel's generic events.
- Hardware specific events defined in Performance Monitoring Unit (PMU) spec.
- Powerpc supports CPU events like:

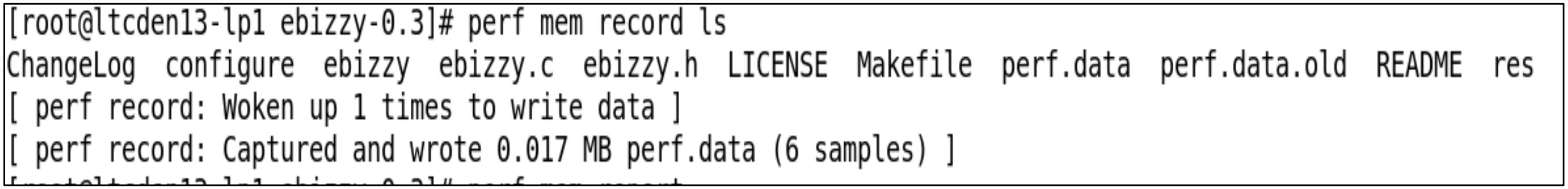
PM_LD_CMPL	Count of Loads completed
PM_DISP_HELD	Dispatch Held
PM_ST_MISS_L1	Store Missed L1
PM_FLOP	Floating Point Operation Finished
PM_BR_MPRED_CMPL	Number of Branch mispredicts



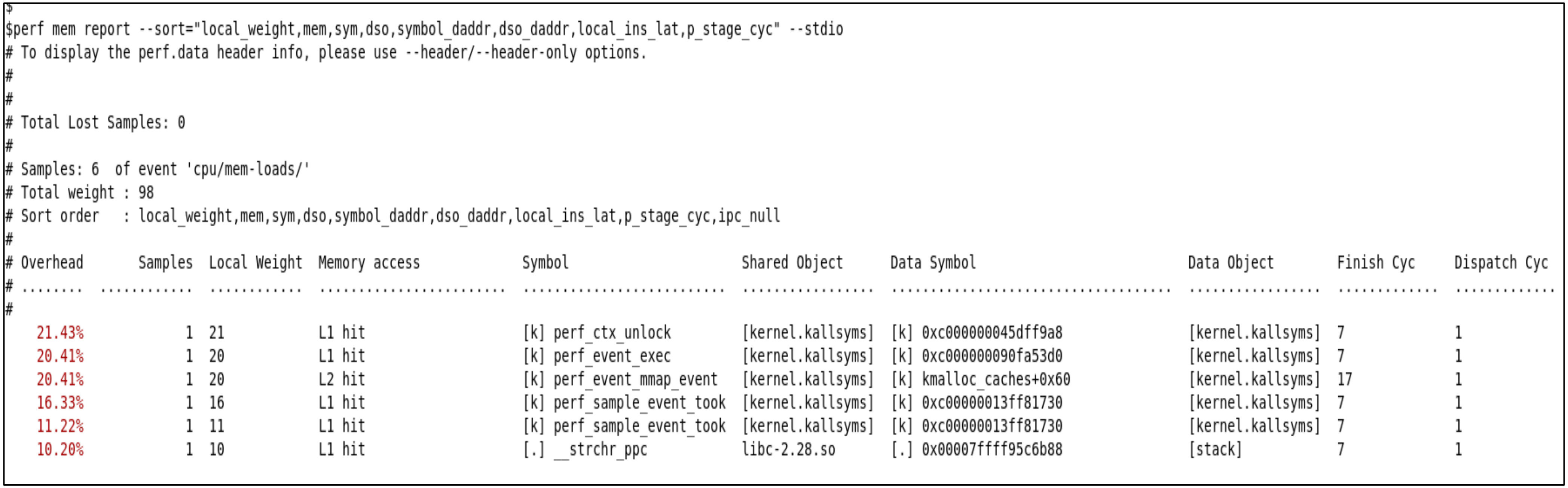
```
/*
 * Raw event encoding for Power9:
 *
 *        60        56        52        48        44        40        36        32
 * | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - |
 *   | | [ ]                       [ ] [      thresh_cmp     ]   [  thresh_ctl   ]
 *   | |  |                         |                                     |
 *   | |  *- IFM (Linux)            |                  thresh start/stop -*
 *   | *- BHRB (Linux)              *sm
 *   *- EBB (Linux)
 *
 *        28        24        20        16        12         8         4         0
 * | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - | - - - - |
 *   [   ] [  sample ]   [cache]   [ pmc ]   [unit ]   []    m   [    pmcxsel    ]
 *     |        |           |                          |     |
 *     |        |           |                          |     *- mark
 *     |        |           *- L1/L2/L3 cache_sel      |
 *     |        |                                      |
 *     |        *- sampling mode for marked events     *- combine
 *     |
 *     *- thresh_sel
```

Perf mem – Memory Access analysis

- Provides information about sampled instruction
  - Useful for memory access analysis
  - Load latency analysis
  - Memory hierarchy (reload source)
  - ./perf mem record/report



```
[root@ltcden13-lp1 ebizzy-0.3]# perf mem record ls
ChangeLog  configure  ebizzy  ebizzy.c  ebizzy.h  LICENSE  Makefile  perf.data  perf.data.old  README  res
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.017 MB perf.data (6 samples) ]
[root@ltcden13-lp1 ebizzy-0.3]# perf mem report
```



```
$
$perf mem report --sort="local_weight,mem,sym,dso,symbol_daddr,dso_daddr,local_ins_lat,p_stage_cyc" --stdio
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 6  of event 'cpu/mem-loads/'
# Total weight : 98
# Sort order   : local_weight,mem,sym,dso,symbol_daddr,dso_daddr,local_ins_lat,p_stage_cyc,ipc_null
#
# Overhead       Samples  Local Weight  Memory access             Symbol                      Shared Object      Data Symbol                           Data Object        Finish Cyc    Dispatch Cyc
# ........  ............  ............  ........................  ..........................  .................  ....................................  .................  ............  .............
#
    21.43%              1  21            L1 hit                    [k] perf_ctx_unlock         [kernel.kallsyms]  [k] 0xc000000045dff9a8                [kernel.kallsyms]  7              1
    20.41%              1  20            L1 hit                    [k] perf_event_exec         [kernel.kallsyms]  [k] 0xc000000090fa53d0                [kernel.kallsyms]  7              1
    20.41%              1  20            L2 hit                    [k] perf_event_mmap_event   [kernel.kallsyms]  [k] kmalloc_caches+0x60               [kernel.kallsyms]  17             1
    16.33%              1  16            L1 hit                    [k] perf_sample_event_took  [kernel.kallsyms]  [k] 0xc00000013ff81730                [kernel.kallsyms]  7              1
    11.22%              1  11            L1 hit                    [k] perf_sample_event_took  [kernel.kallsyms]  [k] 0xc00000013ff81730                [kernel.kallsyms]  7              1
    10.20%              1  10            L1 hit                    [.] __strchr_ppc            libc-2.28.so       [.] 0x00007ffff95c6b88                [stack]            7              1
```