PROFILING DATA STRUCTURES

perf + pahole

Arnaldo Carvalho de Melo
acme@redhat.com
WHAT IS THIS ABOUT?

- Data structures
- Optimal field ordering
- CPU Cache usage
- False Sharing
- Avoiding unintended outcomes
WHY DO WE CARE?

- Cachelines
- Cache hierarchy
- Cache coherency
- L1, L2, LLC
- NUMA: Remote memory
- Load latencies
Machine (23GB total)

Package L#0
- NUMANode L#0 P#0 (23GB)
- L3 (8192KB)
- L2 (256KB)
- L1d (32KB)
- L1i (32KB)
- Core L#0
  - PU L#0 P#0
  - PU L#1 P#4
- Core L#1
  - PU L#2 P#1
  - PU L#3 P#5
- Core L#2
  - PU L#4 P#2
  - PU L#5 P#6
- Core L#3
  - PU L#6 P#3
  - PU L#7 P#7

PCI 00:02.0
- PCI 3d:00.0
  - Net wlp61s0
- PCI 3e:00.0
  - Block nvme0n1
    - 931 GB
  - PCI 00:1f.6
    - Net enp0s31f6

Host: quaco
Date: Mon 12 Sep 2022 04:58:26 AM -03
PROBLEMS?

- Non cache aligned arrays
- Close by read-mostly and write-mostly variables
- Lots fixed over the years
- Learn from it!
TEDIOUS

- Being done already
- Manually
- Tinkering
- Move, rebuild, perf stat it
MANUALLY DOING IT

- Look for "false sharing" on git log
- Eric Dumazet does it a lot
FIRST EXAMPLE

commit 91b6d325635617540b6a1646d0b138bb17cbd569
Author: Eric Dumazet
Date:   Mon Nov 15 11:02:39 2021 -0800

net: cache align tcp_memory_allocated, tcp_sockets_allocated

tcp_memory_allocated and tcp_sockets_allocated often share a common cache line, source of false sharing.
THE CHANGE

+++ b/net/ipv4/tcp.c
-atomic_long_t tcp_memory_allocated; // Current allocated memory
+atomic_long_t tcp_memory_allocated ____cacheline_aligned_in_smp; /

-struct percpu_counter tcpsockets_allocated;
+struct percpu_counter tcpsockets_allocated ____cacheline_aligned_in_smp; /

BCACHE_DEV_SECTORS_DIRTY_ADD()

- Always set_bit(stripe, full_dirty_stripes)
- Cacheline being invalidated constantly
- 100G dirty data with 20 threads
- 50 times slower
- Change to test_bit() first
- Mingzhe Zou
- 7b1002f7cfe581930f63787a0b3de0144e61ed55
cgroup perf regression
'usage' and 'parent' on the same cacheline
'parent' is mostly read
'usage' is mostly written
False sharing
Feng Tang used 'perf c2c' to detect it
https://lore.kernel.org/lkml/20201102091543.GM3109
debian
802f1d522d5fdaefc2b935141bc8fe03d43a99ab
THE CHANGE

+++ b/include/linux/page_counter.h
@@ -12,7 +12,6 @@ struct page_counter {
       unsigned long low;
       unsigned long high;
       unsigned long max;
-      struct page_counter *parent;

       /* effective memory.min and memory.min usage tracking */
       unsigned long emin;
@@ -26,6 +25,12 @@ struct page_counter {
       unsigned long watermark;
       unsigned long failcnt;
       /*
+       * 'parent' is placed here to be far from 'usage' to reduce
+       * false sharing, as 'usage' is written mostly while parent
+       * frequently read for cgroup's hierarchical counting nature.
+       */
+      struct page_counter *parent;
    };

$ pahole page_counter
struct page_counter {
    atomic_long_t        usage;                /*   0   8 */
    long unsigned int    min;                  /*   8   8 */
    long unsigned int    low;                  /*  16   8 */
    long unsigned int    high;                 /*  24   8 */
    long unsigned int    max;                  /*  32   8 */
    long unsigned int    emin;                 /*  40   8 */
    atomic_long_t        min_usage;            /*  48   8 */
    atomic_long_t        children_min_usage;   /*  56   8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    long unsigned int    elow;                 /*  64   8 */
    atomic_long_t        low_usage;            /*  72   8 */
    atomic_long_t        children_low_usage;   /*  80   8 */
    long unsigned int    watermark;            /*  88   8 */
    long unsigned int    failcnt;              /*  96   8 */
    struct page_counter *parent;               /* 104   8 */
} /* size: 112, cachelines: 2, members: 14 */
/* last cacheline: 48 bytes */
$
LAYOUT

- Type info: DWARF, BTF
- BTF now always available
- per-cpu variable types
BTF

- Compact
- Yeah, BPF stuff
- Used with BPF's CO-RE
- And other BPF features
- /sys/kernel/btf/
$ cd /sys/kernel/btf
$ ls -lh vmlinux
-r--r--r--. 1 root root 5.1M Sep  8 20:38 vmlinux
$
$ ls -lh i915
-r--r--r--. 1 root root 556K Sep 12 09:29 i915
$
$ ls -l | wc -l
204
$ lsmod | wc -l
204
$ lsmod | head -2
Module                  Size  Used by
sctp                  434176  28
$
void (*disconnect)(void *); /* 144  8 */
long unsigned int defer_start; /* 152  8 */
long unsigned int defer_warn; /* 160  8 */
u32 pages_state_hold_cnt; /* 168  4 */
unsigned int frag_offset; /* 172  4 */
struct page * frag_page; /* 176  8 */
long int frag_users; /* 184  8 */
/* --- cacheline 3 boundary (192 bytes) --- */
u32 xdp_mem_id; /* 192  4 */
/* XXX 60 bytes hole, try to pack */
/* --- cacheline 4 boundary (256 bytes) --- */
struct pp_alloc_cache alloc __attribute__((__aligned__(64))); /* 256  1032 */
/* XXX 56 bytes hole, try to pack */
/* --- cacheline 21 boundary (1344 bytes) --- */
struct ptr_ring ring __attribute__((__aligned__(64))); /* 1344  192 */
/* XXX last struct has 48 bytes of padding */
/* --- cacheline 24 boundary (1536 bytes) --- */
atomic_t pages_state_release_cnt; /* 1536  4 */
refcount_t user_cnt; /* 1540  4 */
u64 destroy_cnt; /* 1544  8 */
/* size: 1600, cachelines: 25, members: 15 */
/* sum members: 1436, holes: 2, sum holes: 116 */
/* padding: 48 paddings: 2, sum paddings: 52 */
/* forced alignments: 2, forced holes: 2, sum forced holes: 116 */
} __attribute__((__aligned__(64)));
CPU HELPS

- Intel PEBS
- AMD IBS
- ARM CoreSight
18.8.1.2 Load Latency Performance Monitoring Facility

The load latency facility provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table 18-23.

This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).
WHAT DO WE HAVE IN PERF?

- perf stat
- perf mem
- perf c2c
- Several articles about using it
- Improve upon this foundation
PERF MEM

- loads, stores
- load latency
- Stephane Eranian
- Jiri Olsa, Kan Liang, others
# perf mem record -a sleep 1
#
# perf mem -t load report --sort=mem --stdio
# Total Lost Samples: 0
#
# Samples: 51K of event 'cpu/mem-loads,ldlat=30/P'
# Total weight : 4819902
# Sort order : mem
#
# Overhead       Samples  Memory access
# ........  ............  ........................
# 44.87%         20217  LFB or LFB hit
# 27.30%         18618  L3 or L3 hit
# 22.53%         11712  L1 or L1 hit
# 4.85%          637    Local RAM or RAM hit
# 0.25%           1     Uncached or N/A hit
# 0.20%          188    L2 or L2 hit
# 0.00%           35    L3 miss
# perf mem record sleep 1
#  
# perf mem -t load report --sort=mem --stdio
# Total Lost Samples: 0
#
# Samples: 16 of event 'cpu/mem-loads,ldlat=30/P'
# Total weight : 1556
# Sort order : mem
# Overhead   Samples Memory access
# ........  ............  ........................
# 64.52%  8  LFB or LFB hit
# 14.07%  4  L1 or L1 hit
# 11.05%  3  L3 or L3 hit
# 10.35%  1  Local RAM or RAM hit
A FAMILIAR WORKLOAD

$ make -j8 O=../build/allmodconfig/
made[1]: Entering directory 'home/acme/git/build/allmodconfig'

# perf mem record sleep 1m
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.037 MB perf.data (20 samples) ]
#
```
# perf mem report --stdio
# Total Lost Samples: 0
#
# Samples: 11 of event 'cpu/mem-loads,ldlat=30/P'
# Total weight: 2155
# Sort order: local_weight,mem,sym,dso,symbol_daddr,dso_daddr
#
# Local Mem
# Overhead Weig Access Symbol Sh Object Data Symbol Data Obj
#........ .... ........ ................... .........   ................   ...........
23.94% 516 LocalRAM copy_page [kernel] 0xffff8d42228ea900 [unknown]
15.31% 330 LFB flush_signal_handle [kernel] 0xffff8d3f976020a0 [unknown]
14.66% 316 LFB strlen [kernel] 0xffffffff9b5f4cd3 [kernel].ro
13.36% 288 LFB _dl_relocate_object ld-linux.so 0x00007f6ccdc23068 libc.so.6
11.46% 247 LFB next_uptodate_page [kernel] 0xffffe401957e4df4 [unknown]
 7.33% 158 LFB copy_page [kernel] 0xfffff8d41f2dae920 [unknown]
 4.04%  87 LFB unlock_page_memcg [kernel] 0xffffe4019333d8b8 [unknown]
 3.06%  66 L1 check_preemption_di [kernel] 0xfffffa8e8622ffcc8 [unknown]
 2.69%  58 LFB perf_output_begin [kernel] 0xfffff8d3f52a1b01c [unknown]
 2.13%  46 L3 task_work_run [kernel] 0xfffff8d3f4a9c802c [unknown]
 2.00%  43 L1 kmem_cache_alloc_tr [kernel] 0xfffffa8e8622ffbc8 [unknown]
```
PERF C2C

- Cache to Cache
- False Sharing
- Cachelines tugged
- Resolves global variables
- Should resolve its type and field
- Using BTF
ORIGINS

- Dick Fowles
- HP-UX had something similar
- Joe Mario
- Don Zickus
- Jiri Olsa
C2C OUTPUT

- Cachelines where false sharing was detected
- Readers and writers and cacheline offsets
- pid, tid, instruction addr, function name, binary object names
- Source file and line number
- Average load latency
- NUMA nodes and CPUs involved
# perf c2c record -a sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 7.787 MB perf.data (2450 samples) ]
# perf evlist
cpu/mem-loads,ldlat=30/P
cpu/mem-stores/P
dummy:HG
#
### WHATS IN THERE?

```bash
# perf script --cpu 4 --pid 0 | head
swapper 0 [4] 319242.843904: 58 cpu/mem-loads, ldlat=30/P: ffff8d3e49c82e688 11868100242 OP LOAD|LVL LFB or LFB hit|SNP None|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.142295: 39 cpu/mem-loads, ldlat=30/P: ffff8d44865f2408 10268100142 OP LOAD|LVL L1 or L1 hit|SNP None|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.143587: 19614 cpu/mem-stores/P: ffff8d4486500028 5080184 OP STORE|LVL L1 miss|SNP N/A|TLB N/A|LCK N/A
swapper 0 [4] 319242.174494: 33 cpu/mem-loads, ldlat=30/P: ffff8d3f595ddc38 11a68201042 OP LOAD|LVL Local RAM or RAM hit|SNP Hit|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.178002: 27 cpu/mem-loads, ldlat=30/P: ffff8d44865312c0 10668100842 OP LOAD|LVL L3 or L3 hit|SNP None|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.217357: 18 cpu/mem-loads, ldlat=30/P: ffff8d4486532490 10268100142 OP LOAD|LVL L1 or L1 hit|SNP None|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.220573: 15 cpu/mem-loads, ldlat=30/P: ffff8d3f4f35f218 11868100242 OP LOAD|LVL LFB or LFB hit|SNP None|TLB L1 or L2 hit|LCK No
swapper 0 [4] 319242.240176: 15 cpu/mem-loads, ldlat=30/P: ffff8d3f6b617be0 10650100842 OP LOAD|LVL L3 or L3 hit|SNP None|TLB L2 miss|LCK No
swapper 0 [4] 319242.243441: 8849 cpu/mem-stores/P: ffff8d3f40c2b1a4 5080144 OP STORE|LVL L1 hit|SNP N/A|TLB N/A|LCK N/A
```

WHAT WAS REALLY ASKED?

# perf evlist -v | head -1
cpu/mem-loads,ldlat=30/P: type: 4, size: 128, config: 0x1cd, \n{ sample_period, sample_freq }: 4000, \nsample_type: IP|TID|TIME|ADDR|ID|CPU|PERIOD|DATA_SRC|PHYS_ADDR| \n    WEIGHT_STRUCT, \nread_format: ID, disabled: 1, inherit: 1, freq: 1, precise_ip: 3, \nsample_id_all: 1, { bp_addr, config1 }: 0x1f
#
PERF_SAMPLE_DATA_SRC

- PEBS Load Latency
- opcode: load, store, prefetch, exec
- mem level: l1, l2, l3, l4, lfb, ram, pmem, etc
- tlb: not available, hit, miss, l1, l2, hw walker, OS fault handler
- snoop: no snoop, hit, miss, hit modified
- lock: not available, locked instruction
PERF_SAMPLE_WEIGHT_STRUCT

- hardware provided number
- How expensive the sampled action represents
- For profiler to scale samples
# perf c2c report --stats

<table>
<thead>
<tr>
<th>Metric</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total records</td>
<td>3223429</td>
</tr>
<tr>
<td>Locked Load/Store Operations</td>
<td>112673</td>
</tr>
<tr>
<td>Load Operations</td>
<td>1387118</td>
</tr>
<tr>
<td>Loads - uncacheable</td>
<td>1</td>
</tr>
<tr>
<td>Loads - IO</td>
<td>4</td>
</tr>
<tr>
<td>Loads - Miss</td>
<td>142</td>
</tr>
<tr>
<td>Loads - no mapping</td>
<td>2350</td>
</tr>
<tr>
<td>Load Fill Buffer Hit</td>
<td>455747</td>
</tr>
<tr>
<td>Load L1D hit</td>
<td>264355</td>
</tr>
<tr>
<td>Load L2D hit</td>
<td>29304</td>
</tr>
<tr>
<td>Load LLC hit</td>
<td>534642</td>
</tr>
<tr>
<td>Load Local HITM</td>
<td>629</td>
</tr>
<tr>
<td>Load Remote HITM</td>
<td>0</td>
</tr>
<tr>
<td>Load Remote HIT</td>
<td>0</td>
</tr>
</tbody>
</table>
# perf c2c report --stdio

--------------------------------------------------
Shared Data Cache Line Table
--------------------------------------------------

<table>
<thead>
<tr>
<th>Idx</th>
<th>Address</th>
<th>Hitm</th>
<th>Tot</th>
<th>Total</th>
<th>Total</th>
<th>Total</th>
<th>Stores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>LclHitm</td>
<td>rec</td>
<td>Loads</td>
<td>Stores</td>
<td>L1Hit</td>
</tr>
<tr>
<td>0</td>
<td>ffff8d449e7d6380</td>
<td>8.43%</td>
<td>53</td>
<td>53</td>
<td>510</td>
<td>499</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>ffff8d4058209340</td>
<td>6.20%</td>
<td>39</td>
<td>39</td>
<td>371</td>
<td>135</td>
<td>236</td>
</tr>
<tr>
<td>2</td>
<td>ffff8d449e7ff400</td>
<td>5.88%</td>
<td>37</td>
<td>37</td>
<td>501</td>
<td>479</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>ffffffff9bf53980</td>
<td>4.93%</td>
<td>31</td>
<td>31</td>
<td>233</td>
<td>208</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>ffff8d3f49ebd280</td>
<td>3.18%</td>
<td>20</td>
<td>20</td>
<td>162</td>
<td>153</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>ffff8d3f420d4880</td>
<td>2.86%</td>
<td>18</td>
<td>18</td>
<td>126</td>
<td>121</td>
<td>5</td>
</tr>
</tbody>
</table>
TUGGED CACHELINE

<table>
<thead>
<tr>
<th>LclHitm</th>
<th>Off</th>
<th>Code address</th>
<th>lcl hitm</th>
<th>load</th>
<th>rec</th>
<th>cnt</th>
<th>Symbol Object</th>
<th>Source:Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>97.30%</td>
<td>0x0</td>
<td>0xffffffff9a2d293b</td>
<td>113</td>
<td>44</td>
<td>454</td>
<td>8</td>
<td>__mod_node_page_state [kernel] vmstat.c:379</td>
<td></td>
</tr>
<tr>
<td>0.00%</td>
<td>0x8</td>
<td>0xffffffff9a2d29bb</td>
<td>0</td>
<td>112</td>
<td>40</td>
<td>8</td>
<td>__mod_node_page_state [kernel] atomic64_64.h:46</td>
<td></td>
</tr>
<tr>
<td>2.70%</td>
<td>0x18</td>
<td>0xffffffff9a2d2be5</td>
<td>959</td>
<td>103</td>
<td>2</td>
<td>2 refresh_cpu_vm_stats [kernel] atomic64_64.h:46</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NEXT STEPS

• Look at the CL Off (Cacheline Offset)
• Three fields being accessed
• Two are with local HITM
• Find out what is the data structure
• By looking at the functions, source:line
$ perf probe -L vmstat.c:379 | head

379   struct per_cpu_nodestat __percpu *pcp = pgdat->per_cpu_nodestats;
380   s8 __percpu *p = pcp->vm_node_stat_diff + item;
381   long x;
   long t;

   if (vmstat_item_in_bytes(item)) {
      /*
       * Only cgroups use subpage accounting right now; at
       * the global level, these items still change in
   $
LOOKING AT PGDAT

```
$ pfunct __mod_node_page_state
void __mod_node_page_state(struct pglist_data * pgdat,
    enum node_stat_item item, long int delta);
$

$ pahole pglist_data | grep -B2 -A6 per_cpu_nodestats
/* --- cacheline 2704 boundary (173056 bytes) --- */
struct zone_padding _pad2_;              /* 173056   0 */
struct per_cpu_nodestat *per_cpu_nodestats
    __attribute__((__aligned__(64))); /* 173056   8 */
atomic_long_t            vm_stat[41];         /* 173064 328 */
/* size: 173440, cachelines: 2710, members: 32 */
/* sum members: 173309, holes: 6, sum holes: 83 */
/* padding: 48 */
/* forced alignments: 2 */
$```
CODE INSPECTION

- Manual steps
- Streamlining the current UI:
  - Should go from the c2c TUI
  - To the 'perf annotate' browser
  - Positioning at that source:line
- ENTER on type: go to a pahole browser
# perf annotate --stdio2 refresh_cpu_vm_stats

refresh_cpu_vm_stats() /usr/lib/debug/lib/modules/5.18.17-200.fc36.x86_64/vmlinux

    fffffffff812d29e0 :
    static int refresh_cpu_vm_stats(bool do_pagesets)
    struct pglist_data *pgdat;
    struct zone *zone;

    for_each_online_pgdat(pgdat) {
        1f3:   cmpxchg %r8b,%gs:(%rdx)
        ↑  jne    1f3
        movsbl  %al,%edi
        if (v) {
            test    %edi,%edi
            ↓ je     20b
            atomic_long_add(v, &pgdat->vm_stat[i])
        }
Update the zone counters for the current CPU.

Note that refresh_cpu_vm_stats strives to only access node local memory. The per CPU pagesets on remote zones are placed in the memory local to the processor using that pageset. So the loop over all zones will access a series of cachelines local to the processor.

The call to zone_page_state_add updates the cachelines with the stats in the remote zone struct as well as the global cachelines with the global counters. These could cause remote node cache line bouncing and will have to be only done when necessary.

The function returns the number of global counters updated.
PAHOLE TODO

- We have --hex, from struct start
- Implement per-cacheline offsets in pahole output
- For multi cacheline structs
- Like pgdat in the previous example
WHAT DO WE SEE?

- Code
- Source code/line
- All focused where things happened
- Not on what data
- But looking at where it happens
- Helps figuring out the data structure accessed
NO RMAP FOR DATA SYMBOLS?

- Global vars: ok
- SLAB: can we do it?
- From code?
- Going back to args
- DWARF location expressions
- LBR?
PERF LOCK CONTENTION -B

- Gets some backtraces
- Identifies locks by backtrace sigs
- Good enough?
A DETOUR

- perf record + report is too costly
- In kernel aggregation should be supported
- perf kmem, kwork, bcounters do it
- BPF way of doing things
- perf using it
$ wc -l tools/perf/util/bpf_skel/*.bpf.c
  191 tools/perf/util/bpf_skel/bperf_cgroup.bpf.c
   78 tools/perf/util/bpf_skel/bperf_follower.bpf.c
   55 tools/perf/util/bpf_skel/bperf_leader.bpf.c
   92 tools/perf/util/bpf_skel/bpf_prog_profiler.bpf.c
  116 tools/perf/util/bpf_skel/func_latency.bpf.c
  383 tools/perf/util/bpf_skel/kwork_trace.bpf.c
  175 tools/perf/util/bpf_skel/lock_contention.bpf.c
  273 tools/perf/util/bpf_skel/off_cpu.bpf.c
 1363 total
$

PERF BPF SKELS
$ sudo perf lock contention -b
^C
contended  total wait  max wait  avg wait  type    caller

<table>
<thead>
<tr>
<th>Contended</th>
<th>Total Wait (us)</th>
<th>Max Wait (us)</th>
<th>Avg Wait (us)</th>
<th>Type</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>192.67</td>
<td>13.64</td>
<td>4.59</td>
<td>spinlock</td>
<td>queue_work_on+0x20</td>
</tr>
<tr>
<td>23</td>
<td>85.54</td>
<td>10.28</td>
<td>3.72</td>
<td>spinlock</td>
<td>worker_thread+0x14a</td>
</tr>
<tr>
<td>6</td>
<td>13.92</td>
<td>6.51</td>
<td>2.32</td>
<td>mutex</td>
<td>kernfs_iop_permission+0x30</td>
</tr>
<tr>
<td>3</td>
<td>11.59</td>
<td>10.04</td>
<td>3.86</td>
<td>mutex</td>
<td>kernfs_dop_revalidate+0x3c</td>
</tr>
<tr>
<td>1</td>
<td>7.52</td>
<td>7.52</td>
<td>7.52</td>
<td>spinlock</td>
<td>kthread+0x115</td>
</tr>
<tr>
<td>1</td>
<td>7.24</td>
<td>7.24</td>
<td>7.24</td>
<td>spinlock</td>
<td>sys_epoll_wait+0x148</td>
</tr>
<tr>
<td>1</td>
<td>7.08</td>
<td>3.99</td>
<td>3.54</td>
<td>spinlock</td>
<td>delayed_work_timer_fn+0x1b</td>
</tr>
<tr>
<td>1</td>
<td>6.41</td>
<td>6.41</td>
<td>6.41</td>
<td>spinlock</td>
<td>idle_balance+0xa06</td>
</tr>
<tr>
<td>2</td>
<td>2.50</td>
<td>1.83</td>
<td>1.25</td>
<td>mutex</td>
<td>kernfs_iop_lookup+0x2f</td>
</tr>
<tr>
<td>1</td>
<td>1.71</td>
<td>1.71</td>
<td>1.71</td>
<td>mutex</td>
<td>kernfs_iop_getattr+0x2c</td>
</tr>
</tbody>
</table>
PERF SAMPLING IN BPF

- We have BPF programs counting events
- That perf then treats as a counter (evsel)
- Now for sampling
- Do the c2c aggregation in a BPF program
- Until control+C
- Or some other existing perf stop method
- End of a control workload (sleep), etc
- Recent discussion: Namhyung Kim and Song Liu
BTF

- per-cpu variables
- patch to record all variables
- And have it in a kernel module
- Load it and get access to type info
- For all variables
CODE ANNOTATION

• perf annotate
• Source code
• Machine code
• Where events take place
### perf annotate

```
Samples: 6M of events 'anon group { cycles:P, L1-icache-load-misses }', 4000 Hz, Event count (approx.): 42093638433
bpf prog 0bc3fc9d11754ba1 sys enter  bpf prog 0bc3fc9d11754ba1 sys enter [Percent: local period]
```

<table>
<thead>
<tr>
<th>Percent</th>
<th>int sys_enter(struct syscall_enter_args *args)</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.68</td>
<td>push %rbp</td>
</tr>
<tr>
<td>0.71</td>
<td>mov %rsp,%rbp</td>
</tr>
<tr>
<td>0.23</td>
<td>sub $0x200,%rsp</td>
</tr>
<tr>
<td>6.46</td>
<td>push %rbx</td>
</tr>
<tr>
<td>0.55</td>
<td>push %r13</td>
</tr>
<tr>
<td>0.46</td>
<td>push %r14</td>
</tr>
<tr>
<td>0.33</td>
<td>push %r15</td>
</tr>
<tr>
<td>0.55</td>
<td>pushq $0x0</td>
</tr>
<tr>
<td>0.03</td>
<td>mov %rdi,%rbx</td>
</tr>
<tr>
<td></td>
<td>return bpf_get_current_pid_tgid();</td>
</tr>
<tr>
<td>0.54</td>
<td>→ callq *ffffffffffe017a907</td>
</tr>
<tr>
<td>1.41</td>
<td>mov %eax,-0x8(%rbp)</td>
</tr>
<tr>
<td>0.02</td>
<td>mov %rbp,%rsi</td>
</tr>
<tr>
<td></td>
<td>add $0xffffffffffffff8,%rsi</td>
</tr>
<tr>
<td></td>
<td>return bpf_map_lookup_elem(pids, &amp;pid) != NULL;</td>
</tr>
<tr>
<td>0.11</td>
<td>movabs $0x0f8b6b00c000,%rdi</td>
</tr>
<tr>
<td>1.28</td>
<td>→ callq *ffffffffffe017c2a7</td>
</tr>
<tr>
<td>0.02</td>
<td>cmp $0x0,%rax</td>
</tr>
<tr>
<td>0.25</td>
<td>↓ je 3f</td>
</tr>
<tr>
<td>0.10</td>
<td>add $0x38,%rax</td>
</tr>
<tr>
<td>0.54</td>
<td>3f: mov %rax,%rdi</td>
</tr>
<tr>
<td>0.02</td>
<td>xor %eax,%eax</td>
</tr>
<tr>
<td></td>
<td>if (pid_filter_has(&amp;pids_filtered, getpid()))</td>
</tr>
<tr>
<td>0.05</td>
<td>cmp $0x0,%rdi</td>
</tr>
<tr>
<td></td>
<td>↓ jne db</td>
</tr>
<tr>
<td>0.55</td>
<td>xor %edi,%edi</td>
</tr>
</tbody>
</table>

Press 'h' for help on key bindings
nospectre_v1 + nospectre_v2
PERF + PAHOLE

- Data structure annotation
- Show HITM causing fields
- Mapping data memory to a variable
- Finding its type
- BTF info for variables: types
- Function signatures: arg types
THE END

- https://joemario.github.io/blog/2016/09/01/c2c-blog/
- https://frankdenneman.nl/2016/07/11/numa-deep-dive-part-3-cache-coherency/
- acme@kernel.org
- https://twitter.com/acmel