Strange Kernel Performance Changes?
Cache Alignment Matters

Feng Tang
Intel Linux Kernel Team
Background

• 0Day (kernel test robot) keeps testing kernel performance and reporting regressions and improvements
• Recently, there are many Strange cases, which are hard to explain as bisected culprit commits seem to have nothing to do with the benchmark
• Kernel developers including Linus suspected and even challenged the reports: “What?” “Why it matters?”
• Goal: understand and explain them, try to mitigate (make everyone’s life easier)
• Hints and ideas are welcome and appreciated!
Kernel Sections Layout

0000000000000000 D __per_cpu_start
00000000000002e000 D __per_cpu_end
ffffffff81000000 T _stext
ffffffff81000000 T _text
fffffff81e011b7 T _etext
ffffffff82000000 R __start_rodata
ffffffff8249c000 D __end_rodata
ffffffff82600000 D _sdata
ffffffff82876840 D _edata
ffffffff82cc1000 D __init_begin
ffffffff82cc1000 D __per_cpu_load
ffffffff82cef000 T __inittext
ffffffff82d5663f T __einittext
ffffffff82f03390 T __initcall_start
ffffffff82f03e88 T __initcall_end
ffffffff82f1e000 R __init_end
ffffffff82f2a000 B __bss_start
ffffffff83400000 B __bss_stop
ffffffff8342c000 B __end

System.map
Kernel Section Layout -II

- Text/Data sections layout

<table>
<thead>
<tr>
<th>Text</th>
<th>A.text</th>
<th>B.text</th>
<th>C.text</th>
<th>D.text</th>
<th>E.text</th>
<th>F.text</th>
<th>G.text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>A.data</td>
<td>B.data</td>
<td>C.data</td>
<td>D.data</td>
<td>E.data</td>
<td>F.data</td>
<td>G.data</td>
</tr>
</tbody>
</table>

- Link order matters (from Makefile)

<table>
<thead>
<tr>
<th>init</th>
<th>arch/sub-arch</th>
<th>kernel</th>
<th>mm</th>
<th>fs</th>
<th>security</th>
<th>block</th>
<th>drivers</th>
<th>net</th>
</tr>
</thead>
</table>
Most of them are caused unnoticeably by underlying cache alignment changes:

- Text (function) alignment
- Data alignment (false sharing)
- HW cache prefetchers
  - Adjacent cache lines prefetch (2N, 2N+1)
  - L2 cache prefetcher
Kernel functions are all linked together **compactly**

One line of code change may cause changes to the whole kernel text/function’s alignment

The earlier a .o get linked, the more parts it can affect

Can be explained, but hard to be solved

Kconfig or compiler change can greatly affect the result

Examples
  - [LKP] Re: [mm] fd4d9c7d0c: stress-ng.switch.ops_per_sec -30.5% regression
  - [mm/hugetlb] c77c0a8ac4: will-it-scale.per_process_ops 15.9% improvement
Case Study – Text Alignment

- A one-line mm fix patch cause 30.6% regression for stress-ng.switch case
- change in kmem_cache_alloc_bulk()
c->tid = next_tid(c->tid);
- 16 more bytes in binary for the function
  “49 83 40 08 01 addq $0x1,0x8(%r8)”
- The change is gone with forced function alignment
Force all function start address aligned on 64 bytes (merged)

- A black box check which we are not 100% sure
- Kconfig option CONFIG_DEBUG_FORCE_FUNCTION_ALIGN_64B
- Much less report after 0Day enabled it
- Why not default on? – 10% more kernel size, more TLB usage

```
ifdef CONFIG_DEBUG_FORCE_FUNCTION_ALIGN_64B
KBUILD_CFLAGS += -falign-functions=64
endif
```
Data Alignment

• Key is the **Cache False Sharing**
• Data is more complex than text
  • Static Layout
    - .data section
    - specific sections like (percpu)
  • Dynamic Allocation: kmalloc/slab/vmalloc
• Debug Methods
  • perf-c2c
  • pahole
  • add padding
Cache False Sharing

- Data loaded from memory to cache on cacheline granularity
- Multiple CPUs access data in one cache line
  - all read → Fine
  - one write → Bad
- Try to separate them in hot data structure

Mitigation (Debug) – Data Alignment

• Force data sections of every .o file aligned (patch posted)
• Change in linker script vmlinux.lds.S
• Debug only due to huge size increase

```
/* Data */
- .data : AT(ADDR(.data) - LOAD_OFFSET) {{
+ .data : AT(ADDR(.data) - LOAD_OFFSET)
+#ifdef CONFIG_DEBUG_FORCE_DATA_SECTIONAligned
+ /* Use the biggest alignment of below sections */
+ SUBALIGN(THREAD_SIZE)
+ #endif
+ {{
```

• Per-CPU data - Add debug allocation macros to force all percpu-data address aligned
• Kmalloc/slab - Force alignment (slab has parameter)
HW Cache Prefetcher

- Most platforms have them ON by default as being helpful generally
- Transparent to SW programmer
- Accuracy affects bus BW hugely
- May vary on different generations as the algorithm evolves
- Consider them if SW debugging can’t help
- Real cases related to the first 2 types

Adjacent Cache Lines Prefetch

- When one cache line is accessed and fetched, its adjacent cache line will be fetched too
- 64B cache line extended into 128B ‘fat cacheline’
- Can not be detected by tools like perf-c2c

<table>
<thead>
<tr>
<th>128 * N</th>
<th>2N</th>
<th>2N+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 * (N+1)</td>
<td>2N+2</td>
<td>2N+3</td>
</tr>
<tr>
<td>128 * (N+2)</td>
<td>2N+4</td>
<td>2N+5</td>
</tr>
<tr>
<td>128 * (N+3)</td>
<td>2N+6</td>
<td>2N+7</td>
</tr>
<tr>
<td>128 * (N+4)</td>
<td>2N+8</td>
<td>2N+9</td>
</tr>
</tbody>
</table>
Case Study – HW Prefetcher

- Patch removing a ‘struct page_counter’ from ‘struct mem_cgroup’, causes -22.7% regression for will-it-scale/page_fault2

- Commit does have relation with the test case, looks to be alignment related

- 3 hot members(A, B, C) sit in 2 adjacent cache lines which were not in one 128B trunk, but were pulled into one by the commit.

- “False sharing” of 2 cachelines

- Solution – separate them into different 128B trunks
Mitigation – Selective Isolation

- Goal: Make kernel performance more stable (Less surprise)
- Chose $N(10 \sim 20)$ .o files, add 64/128B alignment to one function and one data of them (modules A/D/I below)
- Divide kernel into $N$ independent capsules like capsules in a big ship - one capsule changed/broken won’t affect others
- Rule: select more in critical and early modules
- It won’t hurt, with minimal increase of kernel size
Todos

• Upstream the mitigation and debug patches
• Extend perf-c2c tool to cover adjacent cache line prefetch
• Explore more about HW prefetcher
• Check cases which are still not explained
Q&A

Thank You!