The TLB

When the kernel unmaps or modified the attributes of a range of memory, it has two choices:

1. Flush the entire TLB with a two-instruction sequence. This is a quick operation, but it causes collateral damage: TLB entries from areas other than the one we are trying to flush will be destroyed and must be refilled later, at some cost.
2. Use the invlpg instruction to invalidate a single page at a time. This could potentially cost many more instructions, but it is a much more precise operation, causing no collateral damage to other TLB entries.

Which method to do depends on a few things:

1. The size of the flush being performed. A flush of the entire address space is obviously better performed by flushing the entire TLB than doing 2^48/PAGE_SIZE individual flushes.
2. The contents of the TLB. If the TLB is empty, then there will be no collateral damage caused by doing the global flush, and all of the individual flush will have ended up being wasted work.
3. The size of the TLB. The larger the TLB, the more collateral damage we do with a full flush. So, the larger the TLB, the more attractive an individual flush looks. Data and instructions have separate TLBs, as do different page sizes.
4. The microarchitecture. The TLB has become a multi-level cache on modern CPUs, and the global flushes have become more expensive relative to single-page flushes.

There is obviously no way the kernel can know all these things, especially the contents of the TLB during a given flush. The sizes of the flush will vary greatly depending on the workload as well. There is essentially no "right" point to choose.

You may be doing too many individual invalidations if you see the invlpg instruction (or instructions _near_ it) show up high in profiles. If you believe that individual invalidations being called too often, you can lower the tunable:

```
/sys/kernel/debug/x86/tlb_single_page_flush_ceiling
```

This will cause us to do the global flush for more cases. Lowering it to 0 will disable the use of the individual flushes. Setting it to 1 is a very conservative setting and it should never need to be 0 under normal circumstances.

Despite the fact that a single individual flush on x86 is guaranteed to flush a full 2MB [1], hugetlbfs always uses the full flushes. THP is treated exactly the same as normal memory.

You might see invlpg inside of flush_tlb_mm_range() show up in profiles, or you can use the trace_tlb_flush() tracepoints. to determine how long the flush operations are taking.

Essentially, you are balancing the cycles you spend doing invlpg with the cycles that you spend refilling the TLB later.

You can measure how expensive TLB refills are by using performance counters and 'perf stat', like this:

```
perf stat -e
  cpu/event=0x8,umask=0x84,name=dtlb_load_misses_walk_duration/,
  cpu/event=0x8,umask=0x82,name=dtlb_load_misses_walk_completed/,
  cpu/event=0x49,umask=0x4,name=dtlb_store_misses_walk_duration/,
  cpu/event=0x49,umask=0x2,name=dtlb_store_misses_walk_completed/,
  cpu/event=0x85,umask=0x4,name=itlb_misses_walk_duration/,
  cpu/event=0x85,umask=0x2,name=itlb_misses_walk_completed/
```

That works on an IvyBridge-era CPU (i5-3320M). Different CPUs may have differently-named counters, but they should at least be there in some form. You can use pmu-tools 'ocperf list' (https://github.com/andikleen/pmu-tools) to find the right counters for a given CPU.

[1] A footnote in Intel's SDM "4.10.4.2 Recommended Invalidation" says: "One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes."