AMPERE.

Multi-sized THP Performance Benchmarks and Analysis on ARM64

Linux Plumbers Conference 2024

Yang Shi
Olivier Singla

Agenda

- Huge Page Concept and Brief History
- Multi-sized THP
- Benchmarks and Analysis
- Conclusions
- Discussions

Huge Page Concept and Brief History

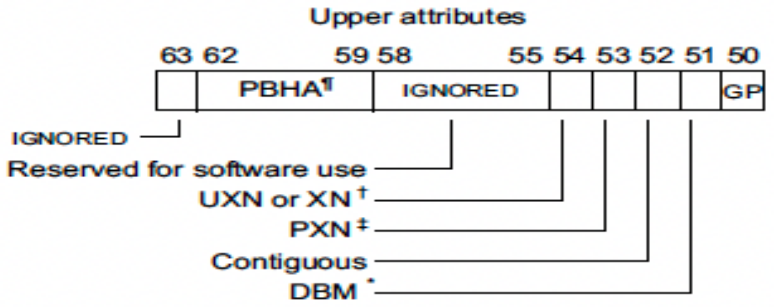
- Huge Page Concept
  - Huge pages are contiguous areas of physical memory
  - The sizes are typically associated with page table level (PMD, PUD)
  - The sizes are also architecture dependent
  - MMU/TLB support
  - Improve TLB coverage
- Huge Page Brief History
  - HugeTLB (hugetlbfs): 2002 (PMD size only, v2.6 era)
  - HugeTLB PUD size: v2.6.27
  - Transparent Huge Page for anonymous (PMD size only): v2.6.38
  - Transparent Huge Page for tmpfs/shmem (PMD size only): v4.8

Multi-sized THP

- Support intermediate sizes instead of PMD size only
  - Some architectures support coalescing multiple PTEs into one entry in TLB (smaller than PMD size)
    - ARM64 contiguous PTE or HPA
    - AMD PTE coalescing
  - Multiple huge page sizes can be supported:
    - 64K/128K/256K/512K/1M (4KPS on ARM64)
- Benefits
  - Improve TLB utilization
  - Easier to allocation than PMD size
  - Possibly unify to one single kernel for distributions
    - RHEL, SuSE and Ubuntu ship both 4K and 64K kernels

Multi-sized THP

- ARM64 supports contiguous bit in page table:



  - The bit identifies a descriptor as belonging to a group of adjacent page table entries
  - Coalesce multiple properly aligned contiguous PTEs into one TLB entry with consistent attributes and permissions:

Base Page Size	Number of adjacent PTEs	Alignment of PTEs	Alignment of Address
4K	16	128 bytes	64KB
16K	128	1KB	2MB
64K	32	256 bytes	2MB

Multi-sized THP

- Multi-sized THP support (anonymous only) was merged in v6.8
  - Supports 16K/32K/64K/128K/256K/512K/1M (4KPS)
  - Added per-size interface at /sys/kernel/mm/transparent_hugepage:

```
[root@fedora ~]# ls /sys/kernel/mm/transparent_hugepage/
defrag            hugepages-128kB   hugepages-32kB   shmem_enabled
enabled           hugepages-16kB    hugepages-512kB  use_zero_page
hpage_pmd_size    hugepages-2048kB  hugepages-64kB
hugepages-1024kB  hugepages-256kB   khugepaged
```

  - Controlled by /sys/kernel/mm/transparent_hugepage/hugepages-<size>/enabled:

```
[root@fedora ~]# cat /sys/kernel/mm/transparent_hugepage/hugepages-64kB/enabled
always inherit madvise [never]
```

  - CONFIG_ARM64_CONTPTE (merged in v6.9):
    - Maintain PTEs contiguous bit
    - TLB maintenance for the region

Benchmarks

- Ran on Ampere® Altra®
  - ARM Neoverse-N1 core
  - 1P/80 cores baremetal machine
- v6.9-rc4 based kernel from mm-unstable
  - Default Fedora config and CONFIG_ARM64_CONTPTE enabled
  - All the optimizations targeted for v6.9 are present
  - Anonymous multi-sized THP only
- Enable target page size only for each run:
  - Set the targe page size knob to always
  - Disable all other page sizes (never)
- 4KPS (4K page size config and huge page disabled) as baseline

Benchmarks

- Workloads
  - Memcached
  - Redis
  - H.264 vbench
  - Kernel compilation
  - Specint 2017
  - MySQL

Benchmarks (Redis and Memcached – Test Setup)

Both Redis and Memcached are an in-memory key-value store for small chunks of arbitrary data. Redis provides additional features such as replication and disk persistency (not used in this benchmark).

Server Side : redis (version 7.2.0) - **memcached** (version 1.6.21)

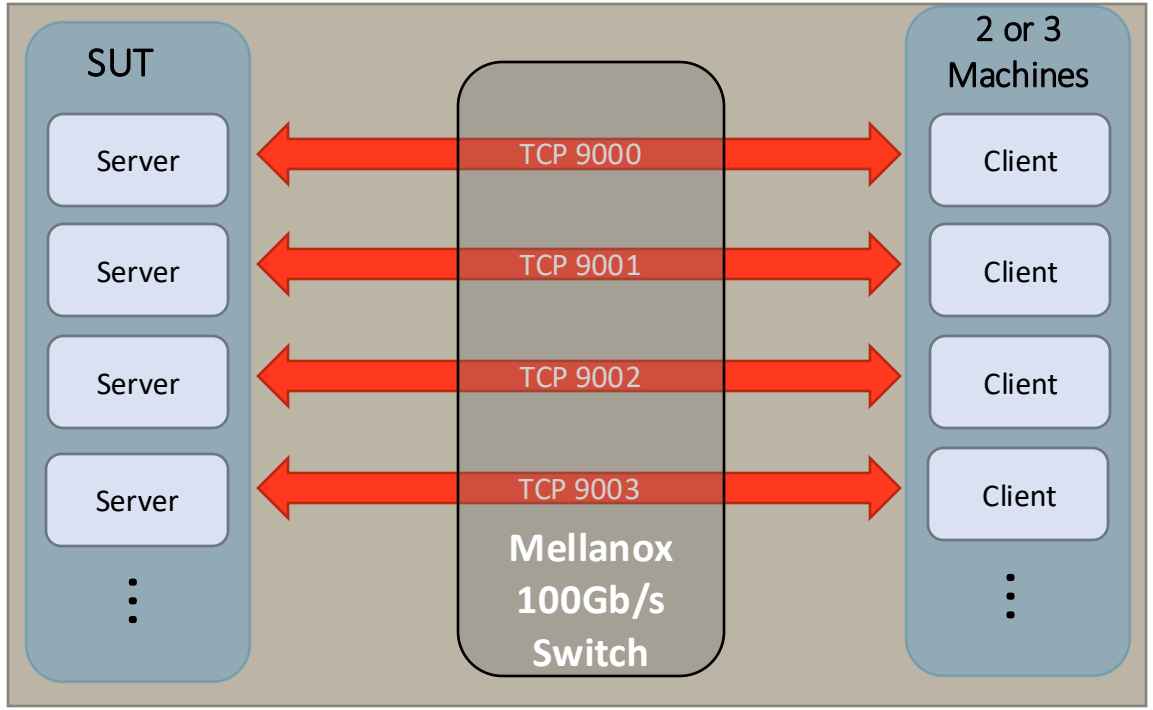
- Compiled natively on each platform with -O3
- Each instance of Redis or Memcached is a single process.
- Redis is single-threaded – Memcached is multi-threaded
- Each instance of either Redis or Memcached is allocated 2.0 GB of memory and configured to hold 13,600,000 keys/data before eviction will happen.

Client Side : memtier_benchmark (version 1.3.0)

- Compiled natively for the clients (Altra) with -O3
- Configuration
  - We first populate the cache, so we get a 95% hits rate
    → 5% of key/value requests will come as not found
  - 1:10 set/get ratio
    → 1 key/value write and 10 key/value read
  - 64 bytes payload (average)
    → 56% 16 Bytes, 30% 64 Bytes, 12% 128 Bytes, 2% 1024 Bytes
  - Data sent to servers is randomized.
  - clients per thread and concurrent pipelined requests:
    → we pick the best configuration for each platform.
- Each run is a 2 minutes-long test (after the cache is populated).
- Some run-to-run variability, so we perform 5 runs and get the median value.

We run Redis and Memcached servers on the SUT and the clients on dedicated machines (2 or 3 depending on the SUT). Each instance of Redis or Memcached is independent and running on a dedicated single CPU.

The Mellanox 100 GbE card is configured to use the default number of interrupts (IRQs), which is 63.



Metrics used:

- Op/Sec – Total number of get/set operations across all instances of either Redis or Memcached in the SUT.
- P99 Latency: must be below 1 msec (SLA)
  → we use the max ("worse") P99 latency across all instances.

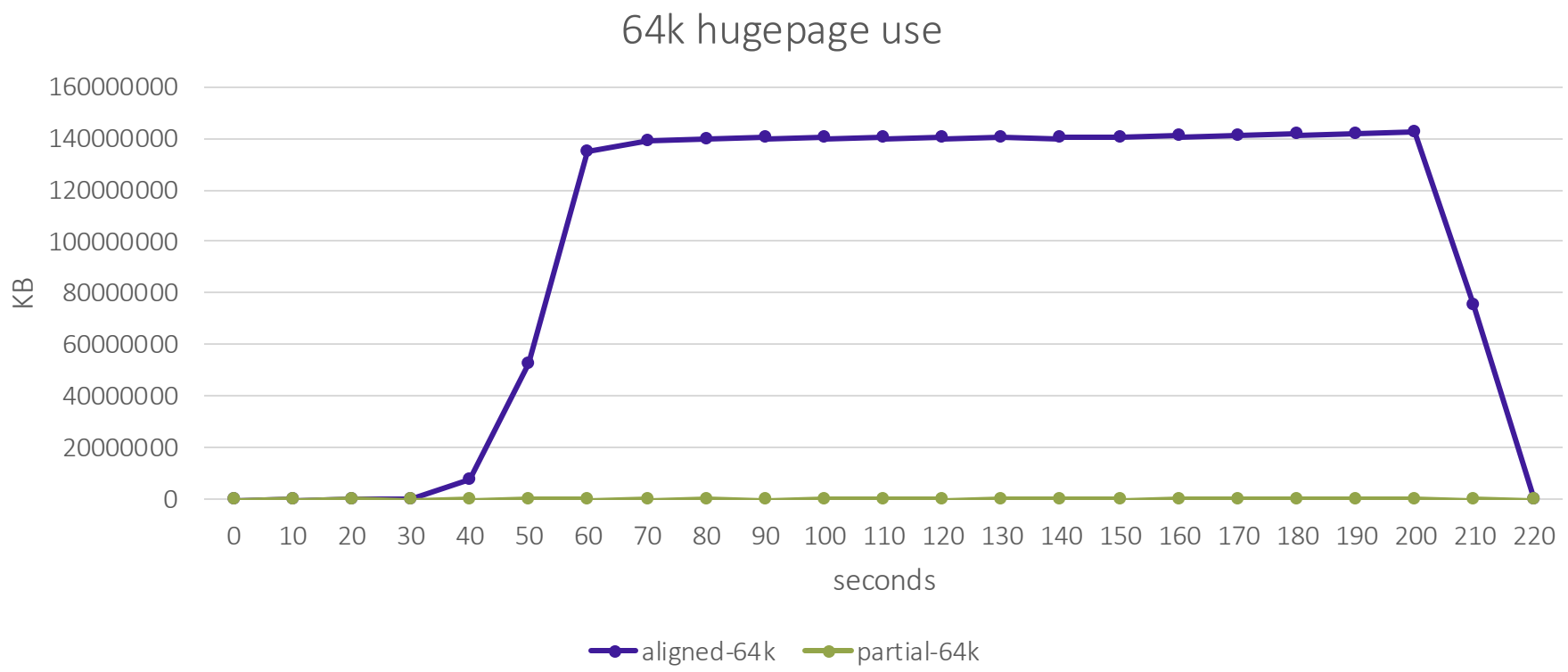
Benchmarks (Memcached)

- No gain with 128K/256K/512K/1M either (4KPS)

	4K(4KPS)	64K (4KPS)	2M (4KPS)	16K (16KPS)	2M (16KPS)	32M (16KPS)	64K (64KPS)	2M (64KPS)
Ops/sec (M)	0%	0%	+17%	+9%	+19%	+23%	+14%	+19%
P99 lat (msec)	0%	0%	-36%	-10%	-29%	-37%	-11%	-29%

Benchmarks (Memcached)

- Memcached huge page (64K 4KPS) distribution



Benchmarks (Memcached)

- PMU metrics
  - TLB misses (DTLB_mpki)
  - TLB walk steps: count the real cost of each page table walk
  - DTLB walk steps pki: DTLB walks steps * 1000 / INST_RETIRED

	4K (4KPS)	64K (4KPS)	2M (4KPS)	16K (16KPS)	2M (16KPS)	32M (16KPS)	64K (64KPS)	2M (64KPS)
IPC:u	0%	+3%	+26%	+13%	+29%	+39%	+27%	+33%
DTLB walk steps	0%	-6%	-54%	-28%	-60%	-78%	-28%	-69%
DTLB walk steps pki	0%	-6%	-62%	-34%	-68%	-84%	-42%	-76%
DTLB_mpki	0%	-5%	-59%	-14%	-67%	-88%	-22%	-75%

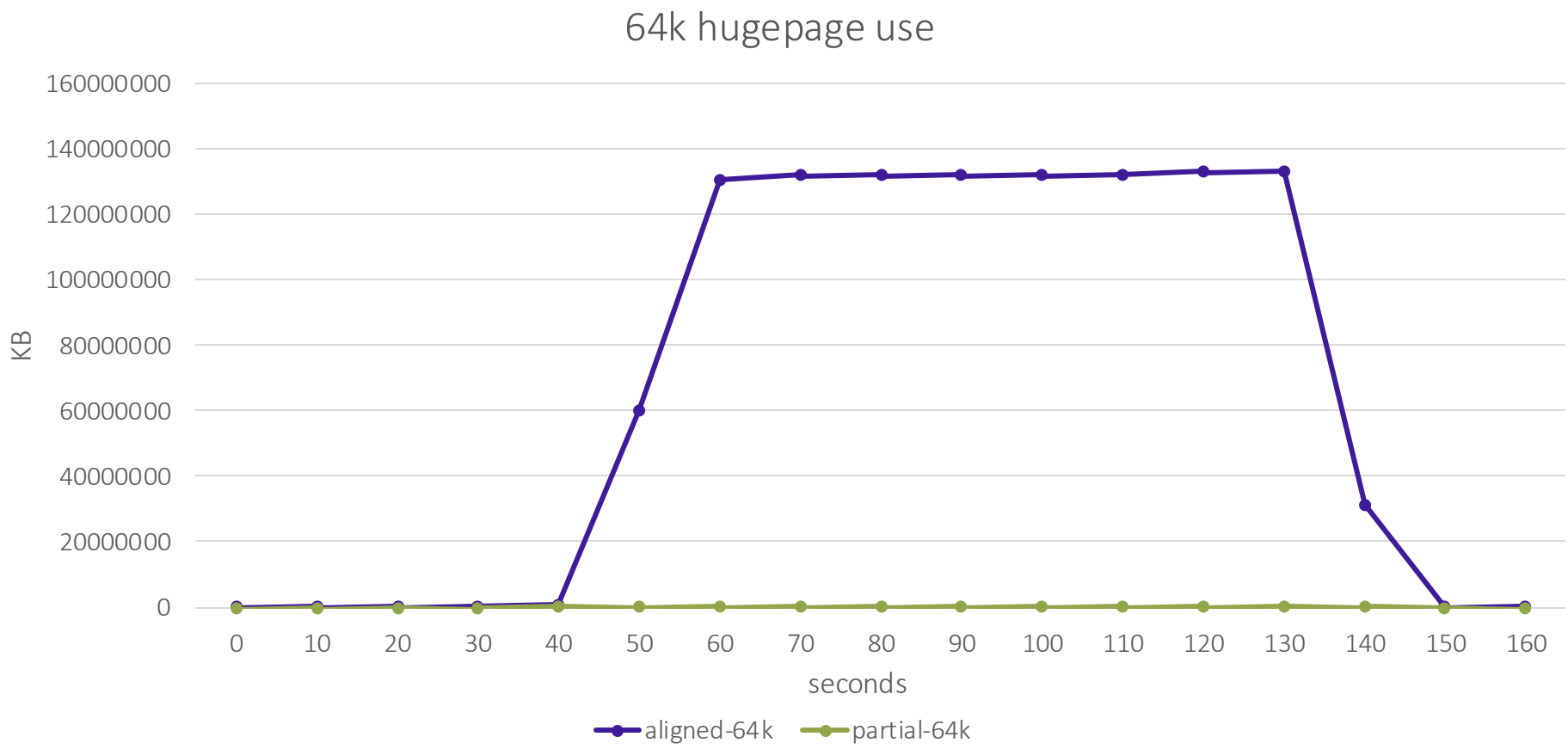
Benchmarks (Redis)

- No gain with 128K/256K/512K/1M either (4KPS)
- Similar pattern with Memcached

	4K(4KPS)	64K (4KPS)	2M (4KPS)	16K (16KPS)	2M (16KPS)	32M (16KPS)	64K (64KPS)	2M (64KPS)
Ops/sec (M)	0%	+3%	+16%	+14%	+19%	+20%	+22%	+19%
P99 lat (msec)	0%	0%	-14%	-11%	-22%	-26%	-23%	-26%

Benchmarks (Redis)

- Redis huge page (64K 4KPS) distribution



Benchmarks (Redis)

- PMU metrics
  - The same metrics as Memcached
    - DTLB_mpki showed weaker correlation
    - DTLB walk steps pki showed much stronger correlation

	4K (4KPS)	64K (4KPS)	2M (4KPS)	16K (16KPS)	2M (16KPS)	32M (16KPS)	64K (64KPS)	2M (64KPS)
IPC:u	0%	+0%	+8%	+3%	+11%	+13%	+10%	+12%
DTLB walk steps	0%	-5%	-56%	-24%	-65%	-78%	-23%	-72%
DTLB walk steps pki	0%	-4%	-59%	-27%	-68%	-81%	-29%	-75%
DTLB_mpki	0%	-9%	-55%	-3%	-60%	-75%	-7%	-68%

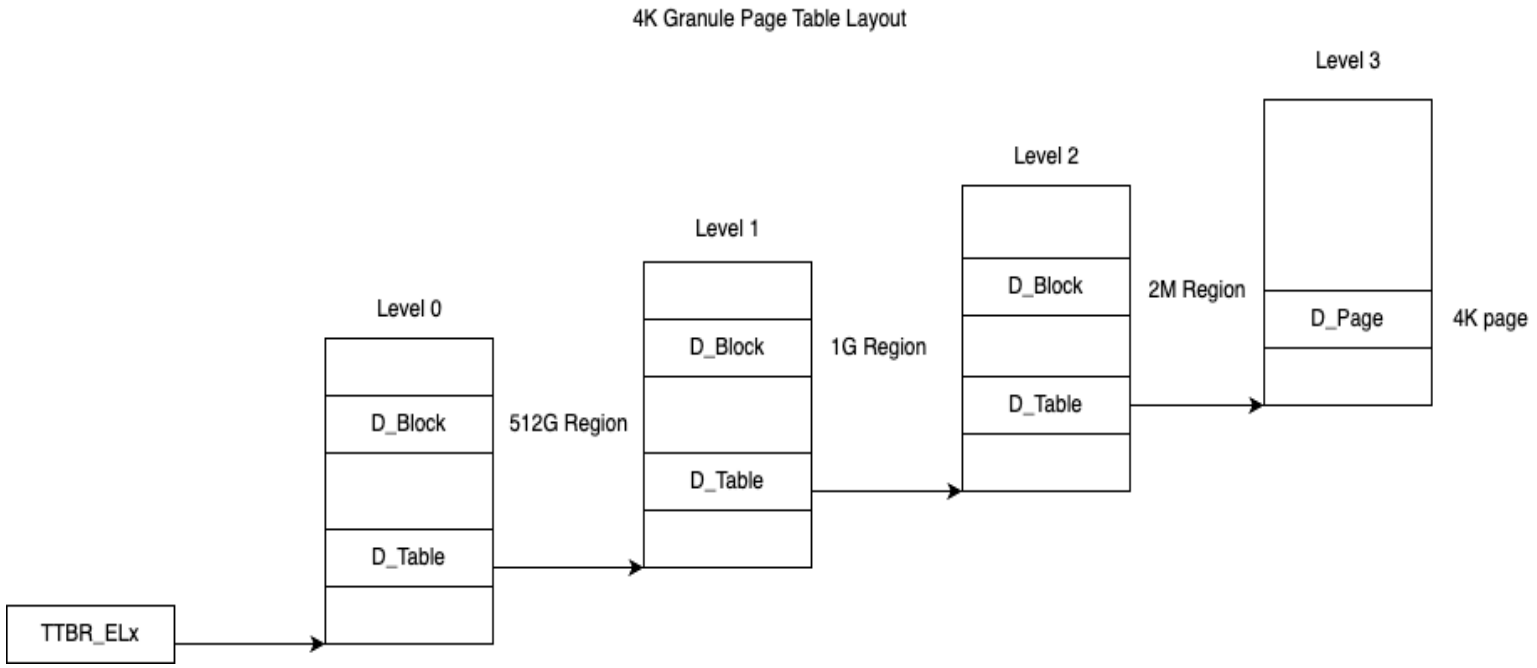
Analysis

- DTLB_mpki doesn't reflect the real cost of page table walk
- The cost of page table walk is determined by:
  - The depth of page table
    - 4KPS 48 VA bit: 4-level
    - 64KPS 48 VA bit: 3-level
  - The region size covered by each intermediate entry

	Level 0	Level 1	Level 2
4KPS	512G	1G	2M
16KPS	128T	64G	32M
64KPS	N/A	4T	512M

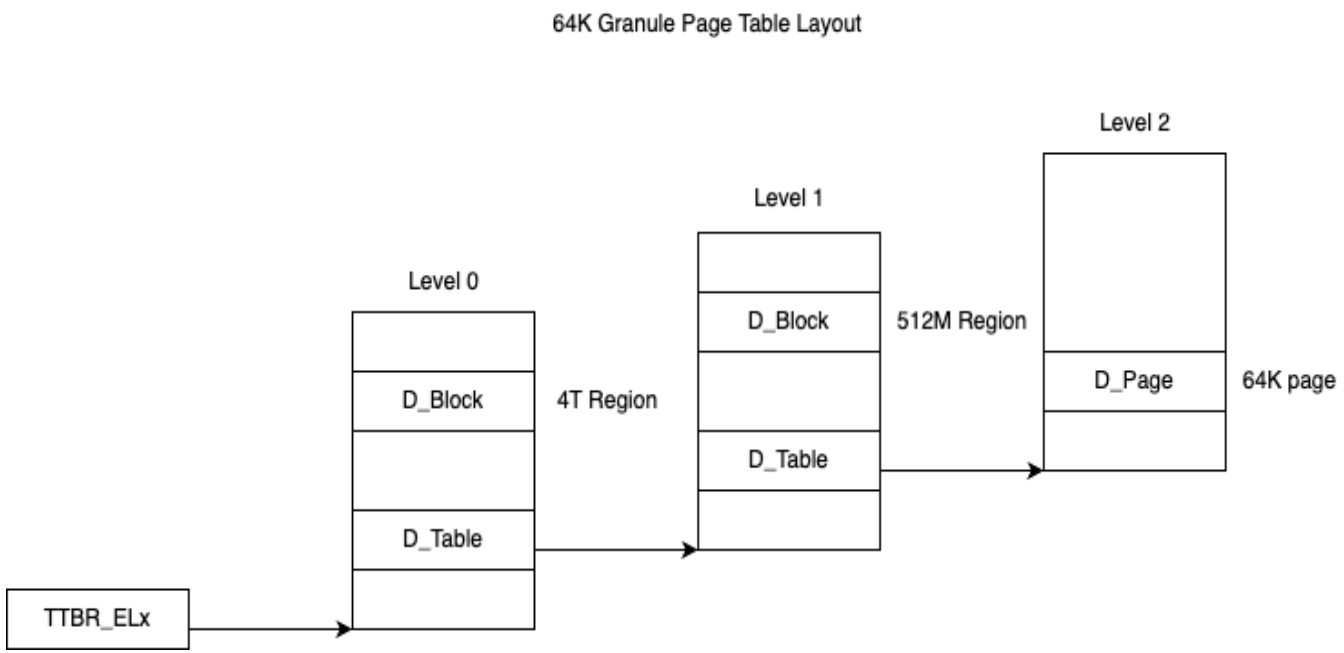
Analysis

- Page table layout
  - 4KPS: 4-level page table (48-bit VA)
    - PMD-sized THP: 3-level page table in fact



Analysis

- Page table layout
  - 16KPS: 4-level page table (48 bit VA), but more entries for each level and each entry cover larger address
  - 64KPS: 3-level page table (48-bit VA)



Benchmarks (H.264 – Test Setup)

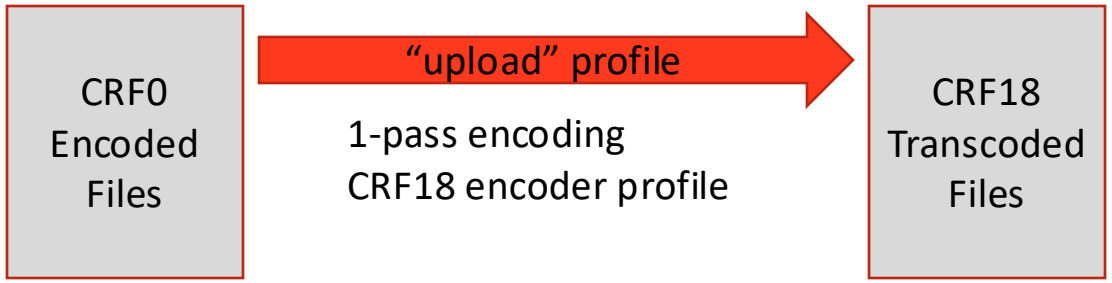
We evaluate x264 using “vbench: a Benchmark for Video Transcoding in the Cloud, a benchmark for the emerging video-as-a-service workload”, available here: http://arcade.cs.columbia.edu/vbench/

15 input files (compressed in H-264), sampled from youtube in 2017, in the public domain:
algorithmically selected to represent a large commercial corpus of millions of videos based on resolution, framerate, and complexity.

Different resolutions (480p, 720p, 1080p, 4k) and different fps (25, 29.97, 30, 50 or 60).

Two different compression are provided:

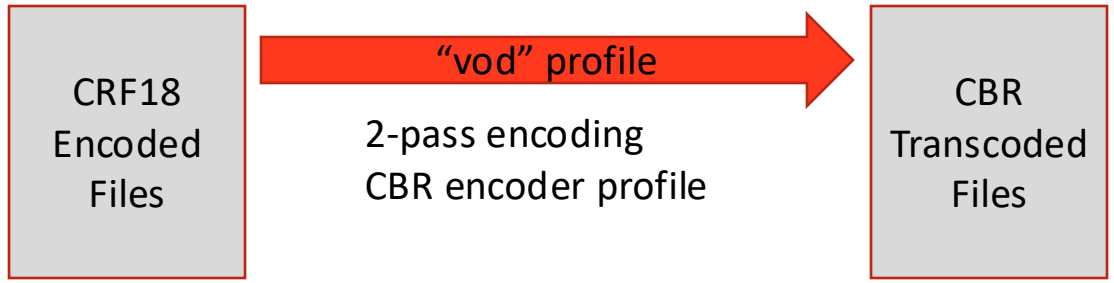
- one (CRF0, lossless) is used for “upload” encoding profile
- and another one (CRF18, visually lossless) is provided for “vod” encoding profile.



File	Input			Ouput		
	Profile	File Size	Bitrate	Profile	File Size	Bitrate
chicken_3840x2160_30.mkv	crf0	285 MB	477.74 Mb/s	crf18	30 MB	49.19 Mb/s

CRF (Constant Rate Factor): a single-pass encoding mode, you choose a quality target, and the encoder adjusts the bitrate to achieve that quality level.

CRF values range from 0 to 51, with lower numbers delivering higher quality scores.



File	Input			Ouput		
	Profile	File Size	Bitrate	Profile	File Size	Bitrate
chicken_3840x2160_30.mkv	crf18	30 MB	477.74 Mb/s	CBR 16,588,800	10 M	16.70 Mb/s

CBR: constant bitrate over the entire file, irrespective of the complexity of the scenes in the video file. You choose a target bitrate and the encoder adjusts quality to meet that bitrate.

In 2-pass encoding: make a first run to collect statistics about the complexity of the video, and then a second pass to achieve the best encoding quality according to these statistics.

Benchmarks (H.264 – Test Setup)

Test Methodology:

- We use a recent version of ffmpeg (git tag 5.1.3) and libx264 (git repo, 2023-01-28), dynamically built with gcc 12 (on Altra for aarch64 – on IceLake for x86_64).
- All the input files and output files are stored in a ramdisk - ffmpeg binary is also stored in a ramdisk
- Limited OS optimizations:

```
tuned-adm profile throughput-performance
echo performance | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

- Each instance of ffmpeg is running on a dedicated CPU (CPU affinity done with numactl).
- Multiple instances of transcoders (single-threaded ffmpeg) over all available CPUs, running in parallel to transcode the set of 15 files.

Metric:

- We use the total number of files transcoded (for instance: 15*80) and divide it by the time it took to transcode all the files.

PSNR (peak signal-to-noise ratios):

- This ratio is used as a quality measurement between the original and a compressed image.
- The higher the PSNR, the better the quality of the compressed, or reconstructed image.
- We used the PSNR to verify that each encoder produced the same visual quality.
- The time it takes to compute PSNR is not part of the test metric.

Benchmarks (H.264)

- Tested with:
  - 64K/128K/256K/512K/1M/2M (4KPS)
  - 16K/2M (16KPS)
  - 64K/2M (64KPS)
- No noticeable gain:
  - 2% gain with 2M vs 4K (4KPS)
  - < 1% gain with 16K vs 2M (16KPS) and 64K vs 2M (64KPS)

Benchmarks (Kernel compilation)

- Using the default Fedora kernel config, gcc 13.2.1 (Fedora 39)
- No more gains with 128K+ (4KPS)
- The more cores, the more memory is used
- Gain mainly comes from reduced sys time due to reduced page faults

	Real-time	Sys-time	Memory use (80 cores)	Page faults
4K (4KPS)	0%	0%	0%	0%
64K (4KPS)	-5%	-33%	+0%	-74%
16K (16KPS)	-15%	-59%	+2%	-72%
64K (16KPS)	-11%	-59%	+2%	-87%
64K (64KPS)	-16%	-80%	+18%	-90%

Benchmarks (specint & MySQL)

- Low single digit gain (< 4%) for some benchmarks of specint 2017 with 64K (4KPS)
- MySQL (read-only): no noticeable gain with 64K (4KPS) an 2M (4KPS) for any metrics
  - Page fault was decreased by 33% with 64K (4KPS). No further reduction with 128K+

Conclusions

- MMU overhead is one of the most significant contributing factors for Memcached and Redis. Roughly 20% - 15% cycles were spent in page table walks for Redis and Memcached
- The cost of page table walk does make significant difference
- 64K (4KPS) can not replace 64K (64KPS)
- 16KPS with mTHP is an optimal solution with most of the performance gain while creating an acceptable memory footprint increase
- Reduced page faults yield performance gain for some type of workloads

Thank You