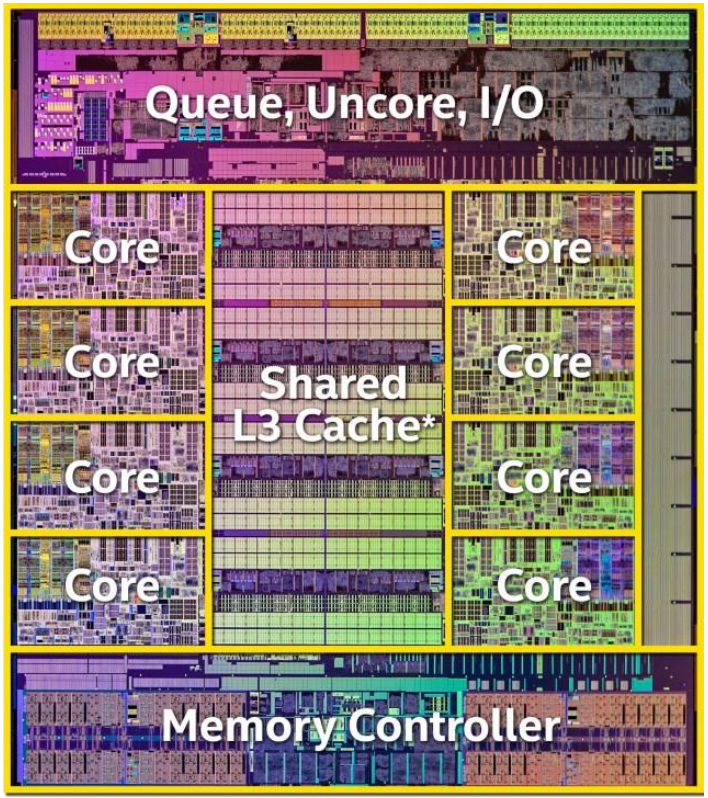
A few experiments with the Cache Allocation Technology

Pawel Szostek
HTCCC,15.02

The caches: a refresher

- Modern computer architectures use caches to **speed-up** memory accesses,
- Caches store data for a later use. **They only “work”, if the data is reused** (for a streaming app. it doesn’t give any advantage)
- Modern x86 machines use **3 levels of cache** - L1D, L1I, L2 and L3
- **L3** cache (also called LLC) is **shared** between all the cores inside a socket
- L1, L2 and L3 caches are **inclusive**
- A process running on a **single core** can use the **whole L3 cache** from all the cores on the same socket



img source: Intel

What is Cache Allocation Technology?

- Long story short: it’s for splitting L3 cache into parts and separating them from each other
- Allows defining **allocation classes** and assigning them to **cores**,
- Provides a way to specify **at runtime** which part of cache can be evicted **when bringing new cache lines from the main memory**
- Together with pinning (or cgroups in the future) **minimizes** cache **pollution**
- Is available on some Haswell SKUs (E5-25x8v3)

What is Cache Monitoring Technology?

- Is bundled with CAT
- Allows monitoring L3 cache allocation per core and process

Possible allocation scenarios

	M1	M2	M3	M4	M5	M6	M7	M8	
COS1									50%
COS2									25%
COS3									12.5%
COS4									12.5%

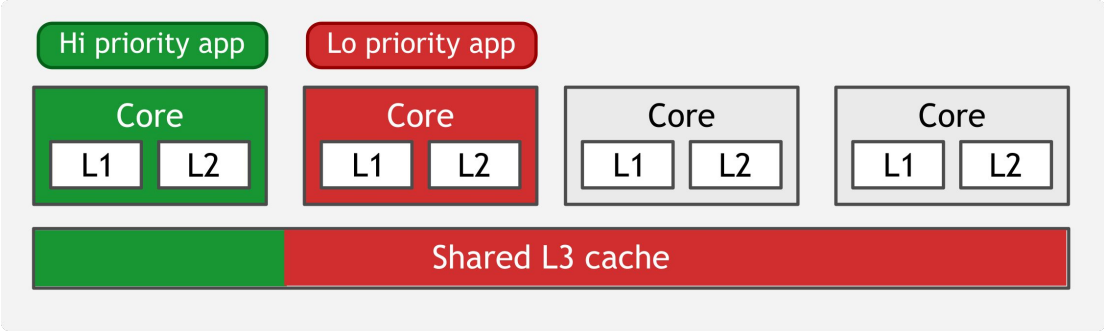
Isolated bitmasks

	M1	M2	M3	M4	M5	M6	M7	M8	
COS1									100%
COS2									50%
COS3									25%
COS4									12.5%

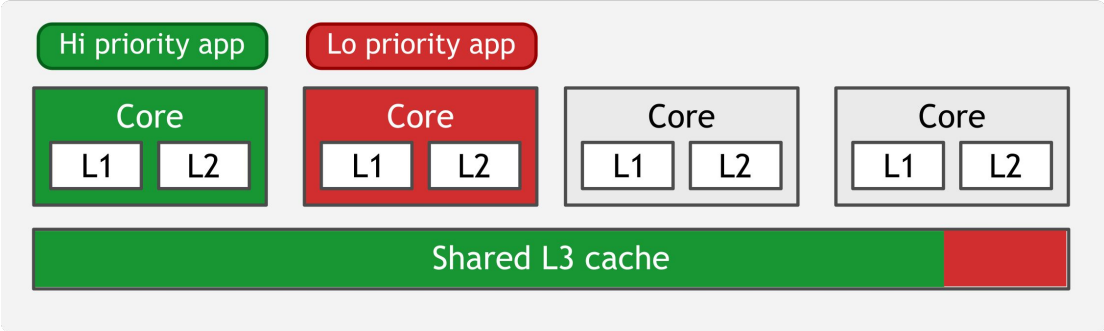
Overlapped bitmasks

Priorities inversion

Without CAT



With CAT



Setting things up

```
[root@olhswep28 working-dir]# pqos -s
BRAND Intel(R) Xeon(R) CPU E5-2658A v3 @ 2.20GHz
L3CA COS definitions for Socket 0:
    L3CA COS0 => MASK 0xfffff
    L3CA COS1 => MASK 0x00fff
    L3CA COS2 => MASK 0x000ff
    L3CA COS3 => MASK 0x0000f
L3CA COS definitions for Socket 1:
    L3CA COS0 => MASK 0xfffff
    L3CA COS1 => MASK 0x00fff
    L3CA COS2 => MASK 0x000ff
    L3CA COS3 => MASK 0x0000f
Core information for socket 0:
    Core 0 => COS0, RMID0
    Core 1 => COS1, RMID0
    Core 2 => COS2, RMID0
….
```

Using top-like cache monitoring

```
TIME 2016-02-15 05:38:42
SOCKET     CORE     RMID    LLC[KB]
     0        0       47      384.0
     0        1       46      240.0
     0        2       45    12960.0
     0        3       44        0.0
     0        4       43        0.0
     0        5       42       48.0
     0        6       41        0.0
     0        7       40        0.0
     0        8       39        0.0
     0        9       38       96.0
     0       10       37       96.0
```

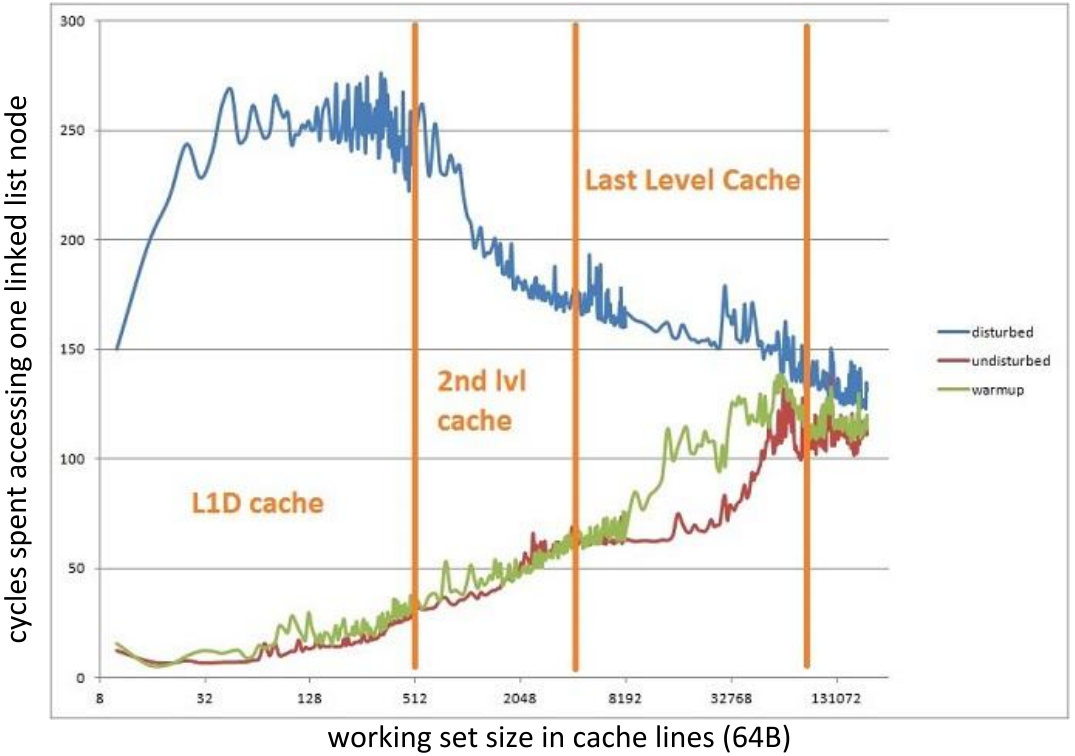
Support for CMT in Linux perf

```
[root@olhswep28 working-dir]# NEVTS=400 perf stat -e
intel_cqm/llc_occupancy/ ParFullCMS4 bench1.g4 1>/dev/null

 Performance counter stats for 'ParFullCMS4 bench1.g4':

       5652480.00 Bytes intel_cqm/llc_occupancy/

    150.010743271 seconds time elapsed
```

Experiments with handcrafted workloads by Intel

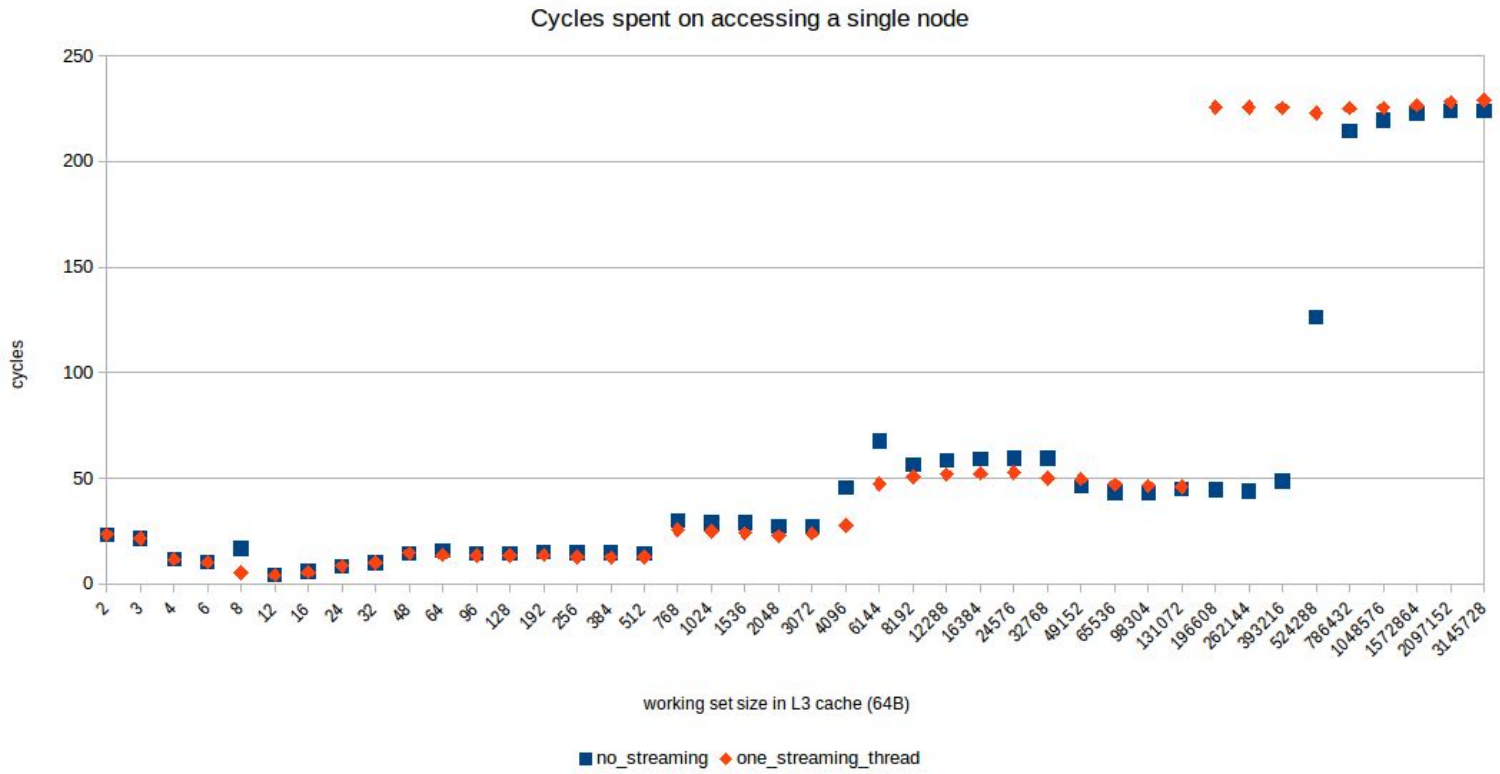
Test scenario:

- Workload #1 traverses a linked list of size N with a random memory layout. It's the hi-priority guy.
- Workload #2 copies 128MB of memory forth and back. It's a lo-priority guy.

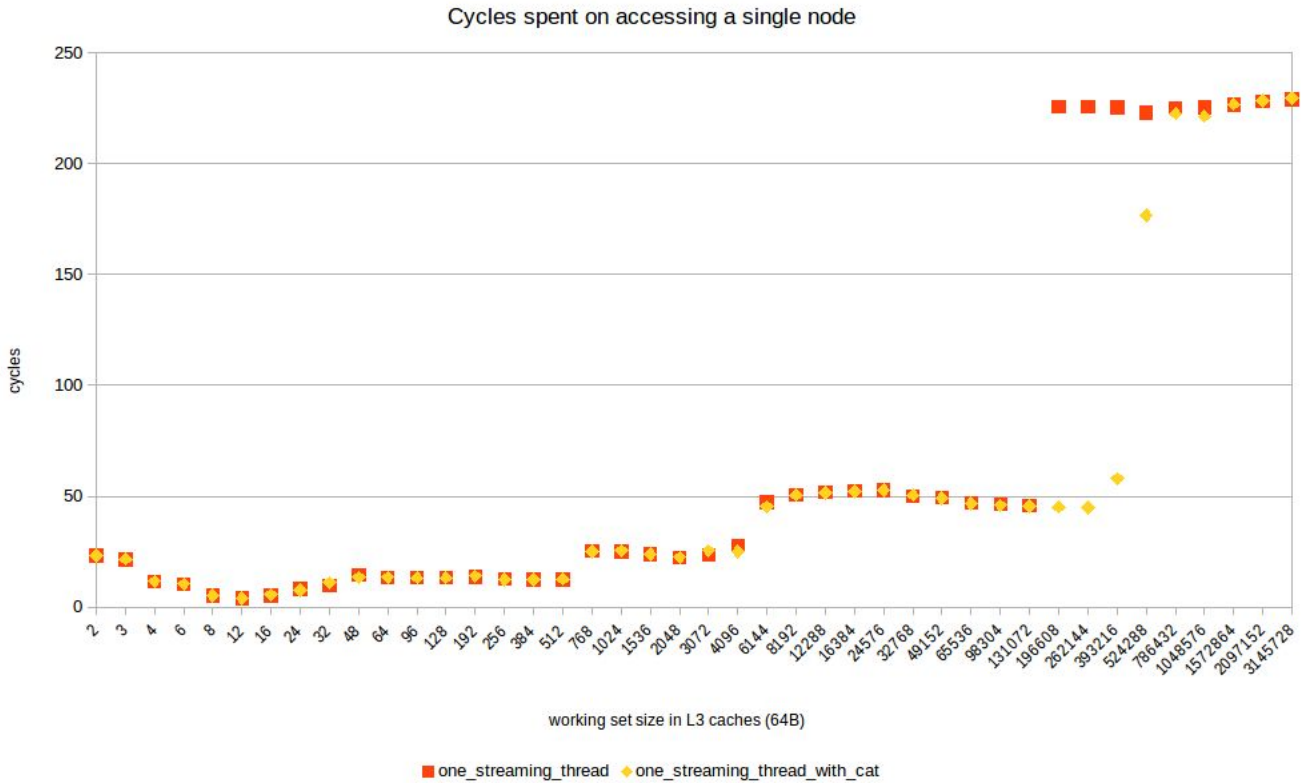


source: software.intel.com/en-us/articles/using-hardware-features-in-intel-architecture-to-achieve-high-performance-in-nfv

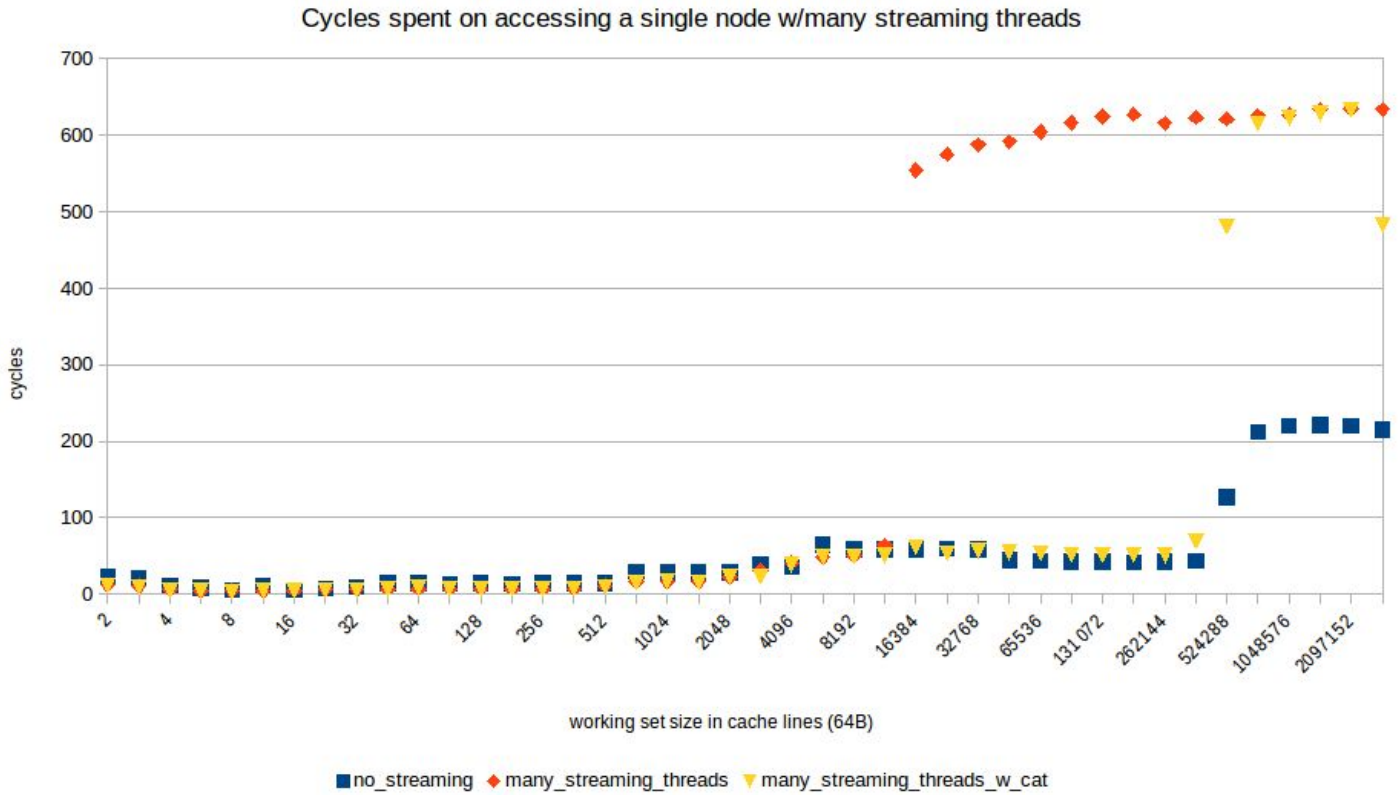
No streaming vs one streaming thread



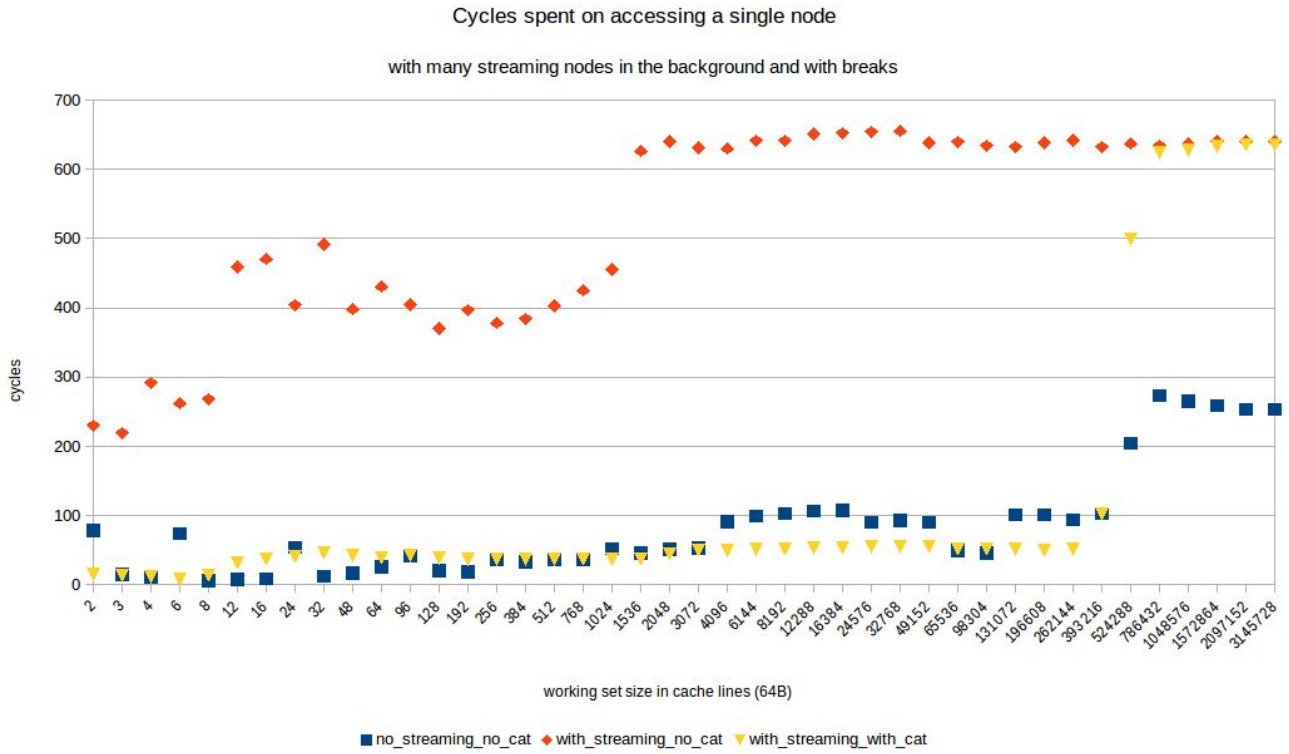
One streaming thread, without and with CAT



Many streaming threads, without and with CAT



Many streaming threads, without and with CAT, interrupted

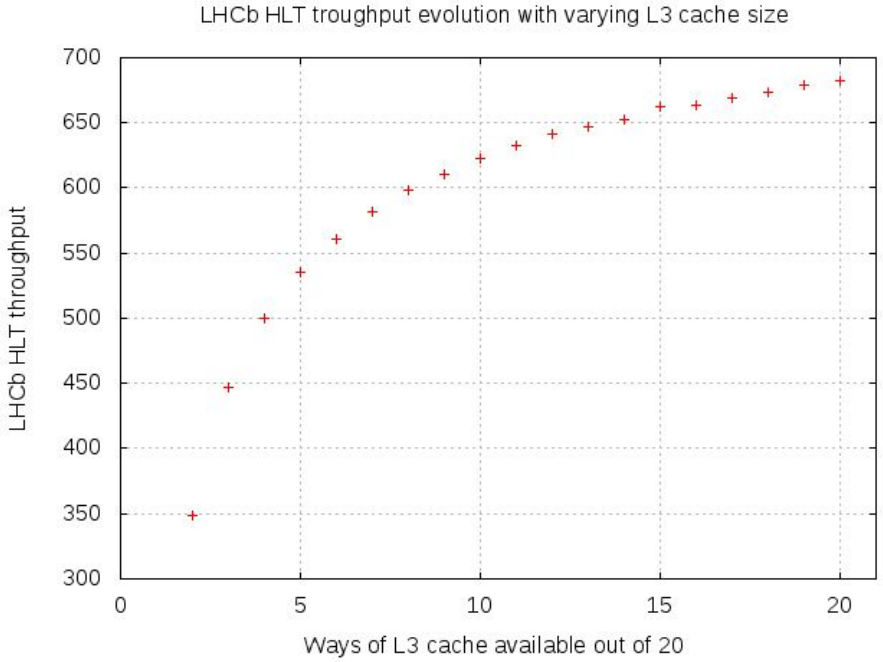


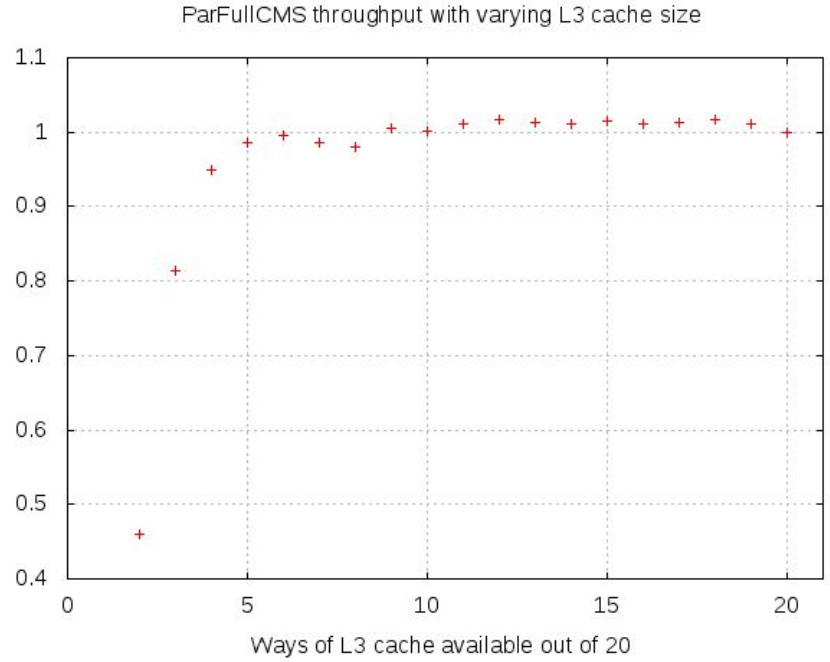
Experiments with the high level trigger software

- I tried to run many HLT instances and split them into groups wrt. the cache allocation, so that they don’t disturb each other
- None of the experiments yielded a better solution than the baseline (no CAT)

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex.. 5	Ex. 6
PQOS #1	0x00FFF	0x003FF	0x0007F	0x07FFF	0x003FF	0x0001F
PQOS #1	0xFFF00	0xFFC00	0x01F80	0xFFFE0	0xFFC00	0x003E0
PQOS #1			0xFE000	0xFFC1F	0xF801F	0x07C00
PQOS #1				0xF83FF	0x07FE0	0xF8000

Side experiment: limiting HLT's cache





A few conclusions

- I tried to mix HLT and simulation (ParFullCMS), but there was no gain from CAT
- Makes a lot of sense to apply it to VMs for a better separation
- Might give extra computing power for opportunistic computation (like it did for Google), but requires finding a proper setup. One have to reconcile with a performance penalty for the hi-priority app
- Might also work for low latency apps to make sure that nothing ever sweeps away its code.