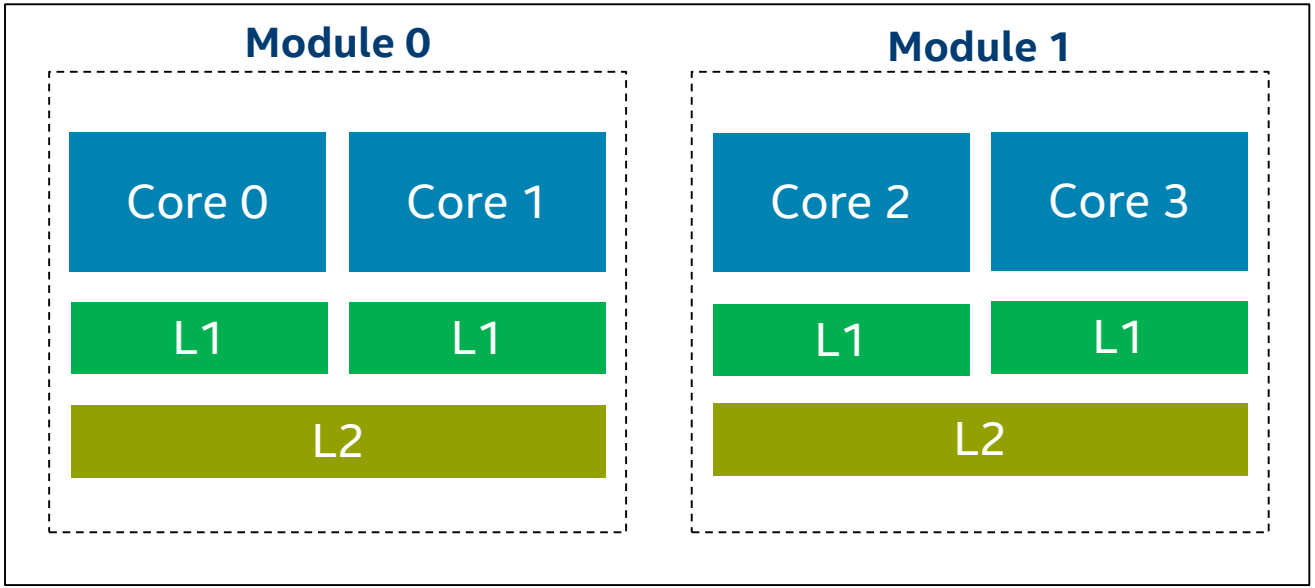
Handling cache-related latency issues in real-time applications

Kirill Uhanov, Vitaly Slobodskoy

Real Time Application (RTA)

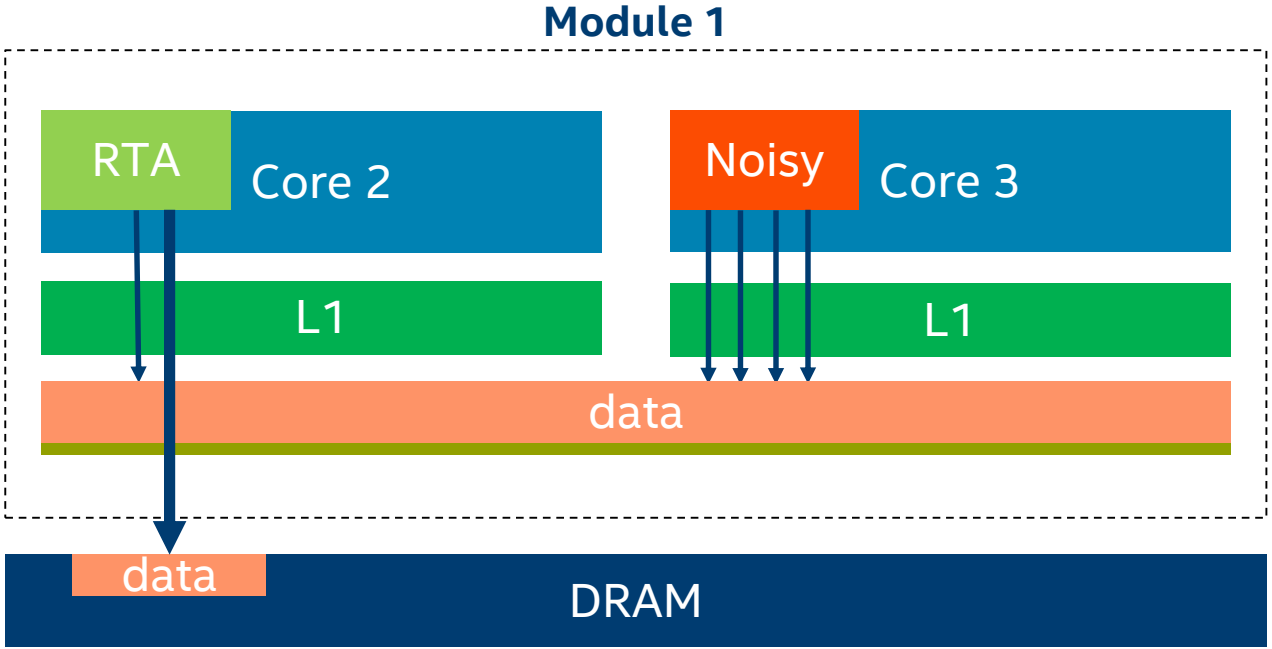
- What is execution success?
  - Bug-free and efficient code
  - Ensure response at the correct time – ‘deadlines’
  - Failing to meet ‘deadlines’ leads to an error
- What are the deadline misses reasons?
  - preemptions
  - interrupts
  - unexpected latency in the critical code execution

CPU Microarchitecture issues - CPU cache miss penalty



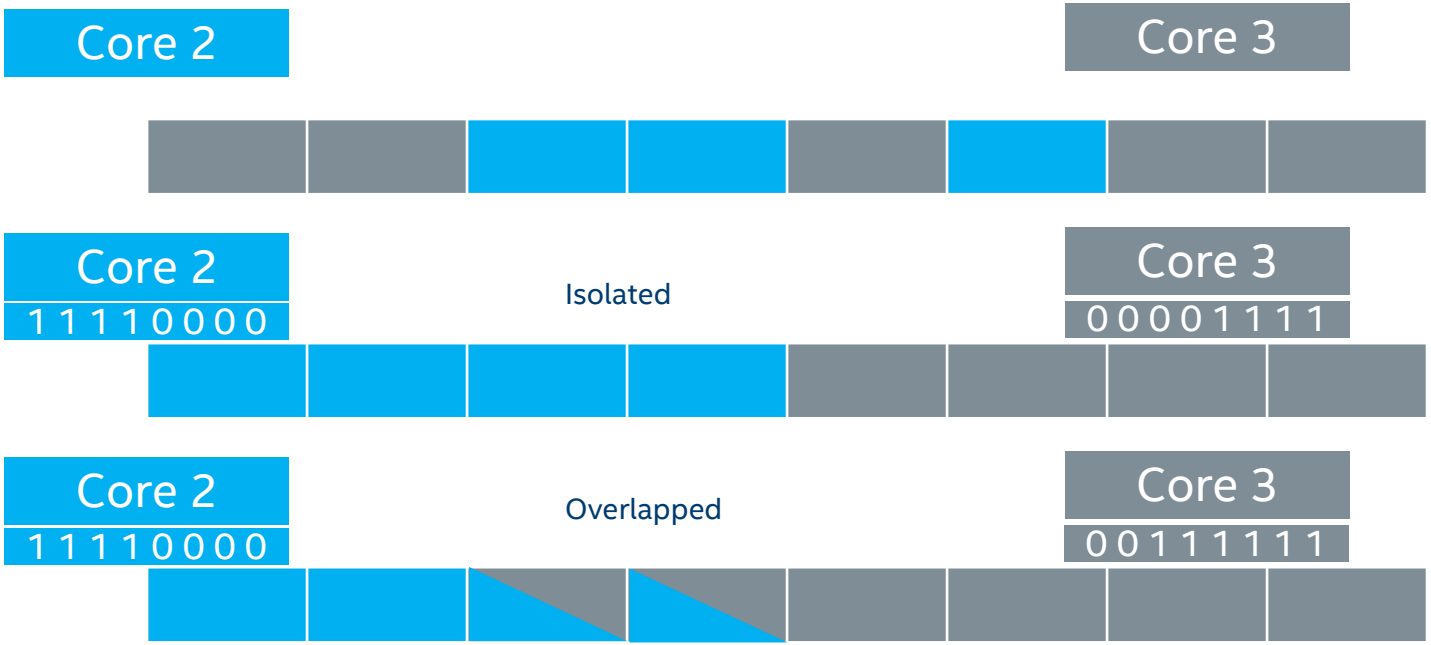
Leaf Hill (Apollo Lake-I)

CPU Microarchitecture issues - CPU cache miss penalty



Cache Allocation Technology (CAT)

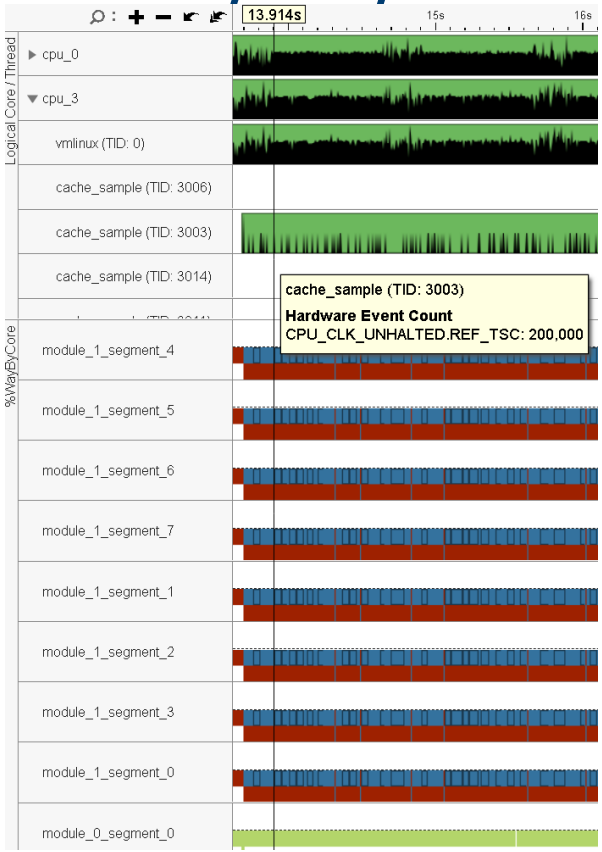
CAT helps to address shared resource concerns by providing software control of where data is allocated into the last-level cache (LLC), enabling isolation of cache segments

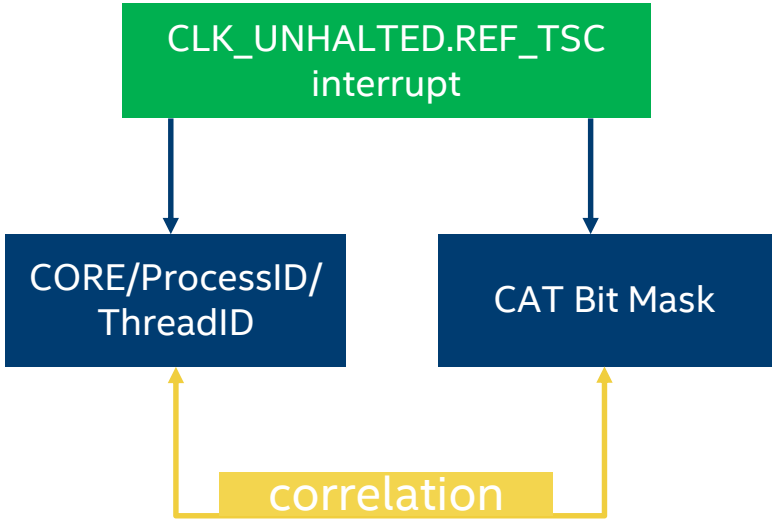


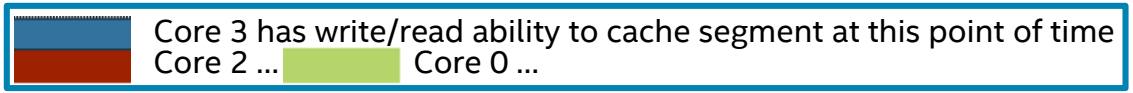
CAT management

- Configure MSRs directly
  - Software Developer's Manual Volume 3B, Section 17.19
    https://software.intel.com/en-us/articles/intel-sdm
- Kernel interface - Resource Control (resctrl) (linux kernel 4.10+)
    https://github.com/intel/intel-cmt-cat/wiki/resctrl

CORE/PID/TID correlation with CBM







Applications

RTA

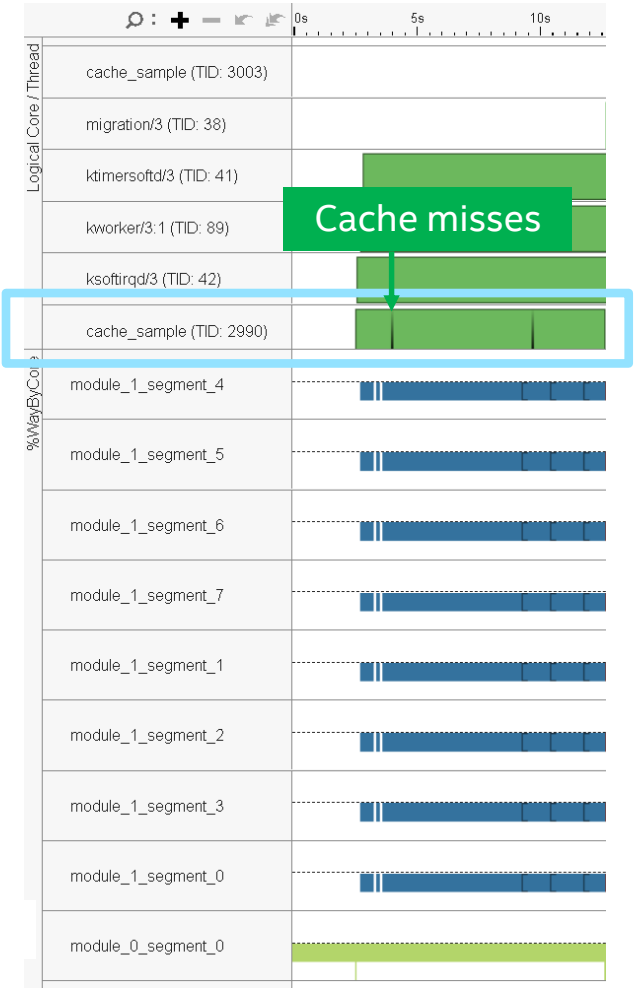
```
struct timespec sleep_timeout =
    (struct timespec) { .tv_sec = 0, .tv_nsec = 10000000 };
...
buffer = malloc(128 * 1024);
...
run_workload(buffer, 128 * 1024);

void run_workload(void *start_addr, size_t size) {
    unsigned long long i, j;
    for (i = 0; i < 1000; i++)
    {
        nanosleep(&sleep_timeout, NULL);
        for (j = 0; j < size; j += 32)
        {
            asm volatile("mov (%0,%1,1), %%eax"
                :
            : "r" (start_addr), "r" (i)
                : "%eax", "memory");
        }
    }
}
```

Noisy neighbor

stress-ng -C 10 --cache-level 2 --taskset 2 --aggressive

Clear run

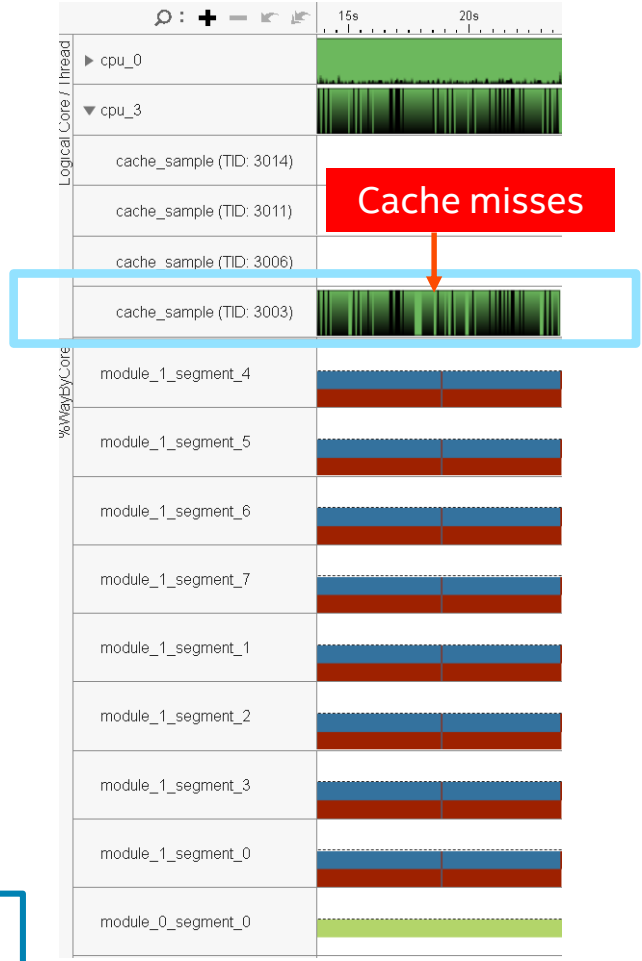
- There are almost no cache misses
- Maximum measured latency: **53587** CPU cycles



Core 3 occupied all L2 cache for module 1

Run with neighbors

- Full cache contention between Core 2 and Core 3, huge count of cache misses.
- Maximum measured latency: **175665 (~3x)** CPU cycles

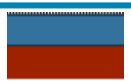


Core 3 occupied L2 cache for module 1
Core 2 occupied L2 cache for module 1

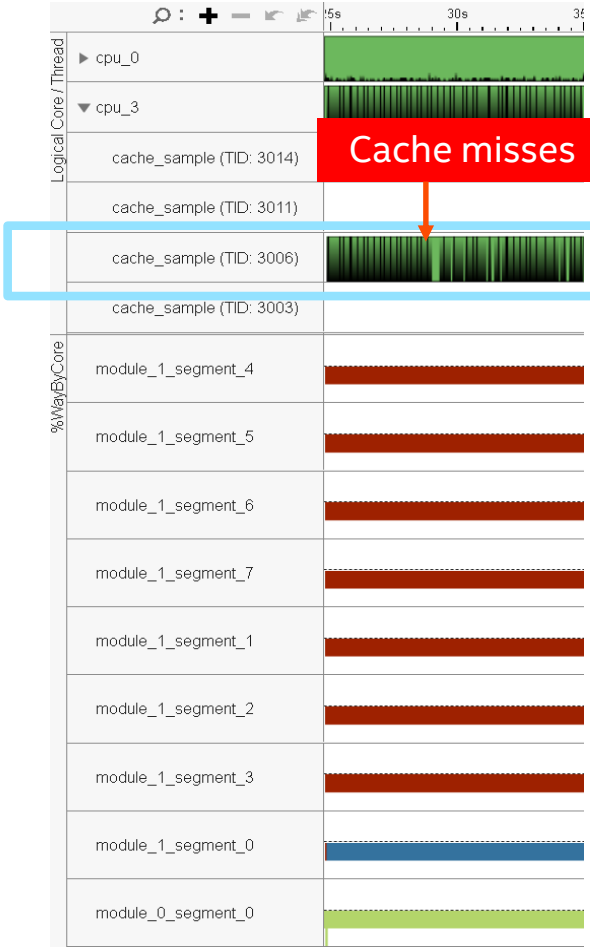
Run with neighbors and 1 cache segment.

```
# set '00000001' Capacity Bit Mask for CORE 3
mkdir /sys/fs/resctrl/clos0
echo 8 >/sys/fs/resctrl/clos0/cpus
echo 'L2:1=1' >/sys/fs/resctrl/clos0/schemata
# set '11111110' CBM for rest CORE
echo 'L2:1=fe' >/sys/fs/resctrl/schemata
```

- The worst case, enormous count of cache misses. Not enough cache for application.
- Maximum measured latency: **216163(~4x)** CPU cycles



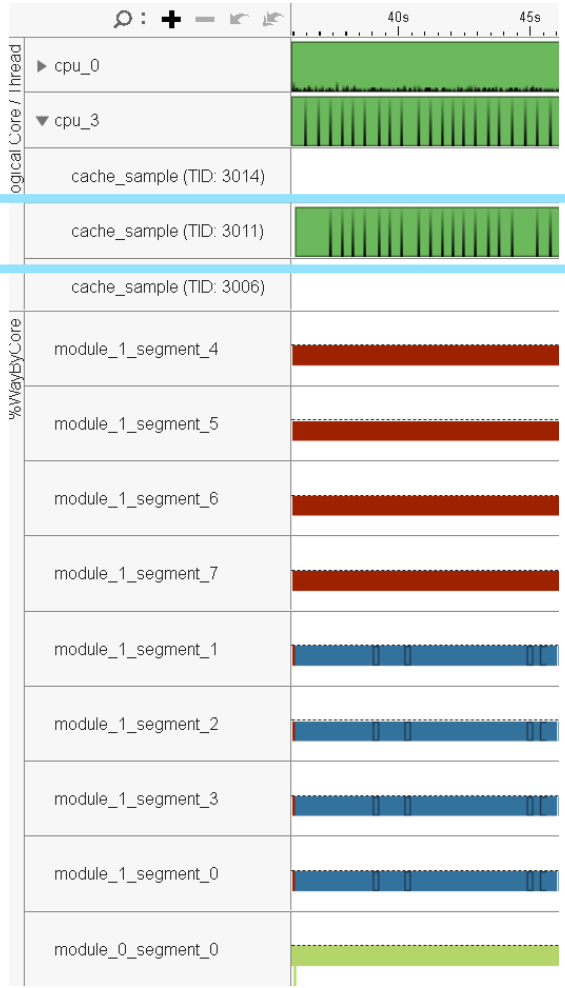
Core 3 occupied just 1 segments L2 cache for module 1
Core 2 occupied 7 segments L2 cache for module 1



Run with neighbors and 4 cache segments.

```
# set '00001111' Capacity Bit Mask for CORE 3
mkdir /sys/fs/resctrl/clos0
echo 8 >/sys/fs/resctrl/clos0/cpus
echo 'L2:1=f' >/sys/fs/resctrl/clos0/schemata
# set '11110000' CBM for rest CORE
echo 'L2:1=f0' >/sys/fs/resctrl/schemata
```

- Insignificant count of cache misses
- Maximum measured latency: **53418** CPU cycles
- App occupies big portion of cache



Core 3 occupied 4 segments L2 cache for module 1
Core 2 occupied 4 segments L2 cache for module 1

Cache pseudo-locking

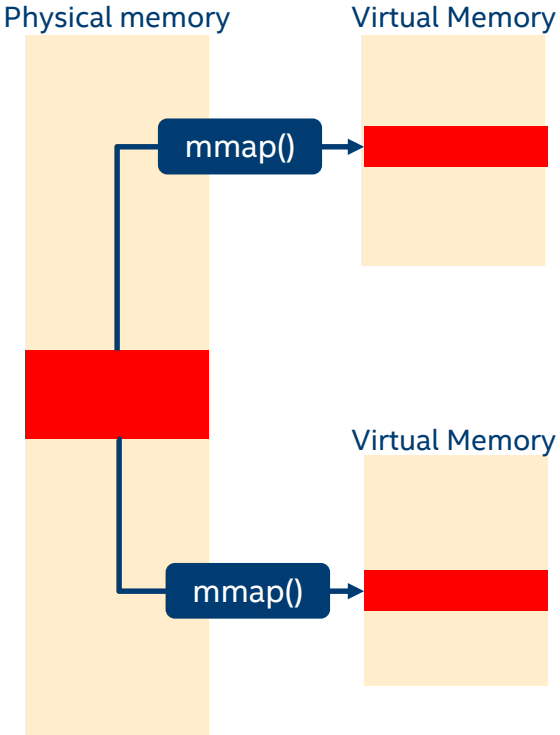
Pseudo-locking helps protect data from being evicted from Level 2 (L2) and Level 3 (L3) cache by other processes attempting to use the same cache.


```
# Create the pseudo-locked region with 1 cache segment
mkdir /sys/fs/resctrl/demolock
echo pseudo-locksetup > /sys/fs/resctrl/demolock/mode
echo 'L2:1=0x1' > /sys/fs/resctrl/demolock/schemata
cat /sys/fs/resctrl/demolock/mode
pseudo-locked
```

Map pseudo-locked memory to user space

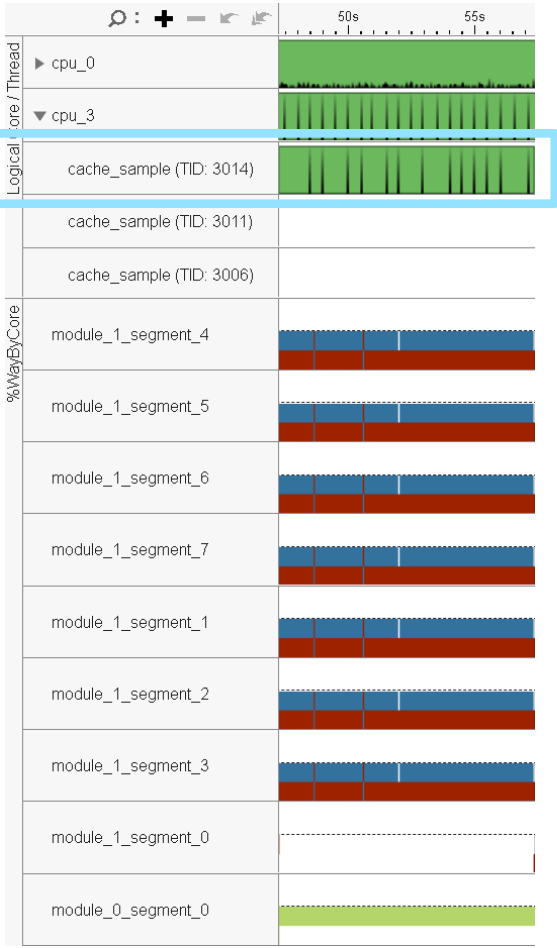
```
struct timespec sleep_timeout =
    (struct timespec) { .tv_sec = 0, .tv_nsec = 10000000 };
...

/ * buffer = malloc(128 * 1024); */
open("/dev/pseudo_lock/demolock", O_RDWR);
buffer = mmap(0, 128 * 1024, PROT_READ | PROT_WRITE, MAP_SHARED,
    dev_fd, 0);
...
run_workload(buffer, 128 * 1024);

void run_workload(void *start_addr, size_t size) {
    unsigned long long i, j;
    for (i = 0; i < 1000; i++)
    {
        nanosleep(&sleep_timeout, NULL);
        for (j = 0; j < size; j += 32)
        {
            asm volatile("mov (%0,%1,1), %%eax"
                :
            : "r" (start_addr), "r" (i)
                : "%eax", "memory");
        }
    }
}
```

User space maps (mmap()) pages of pseudo-locked memory into its own address space.



Run with pseudo-locking

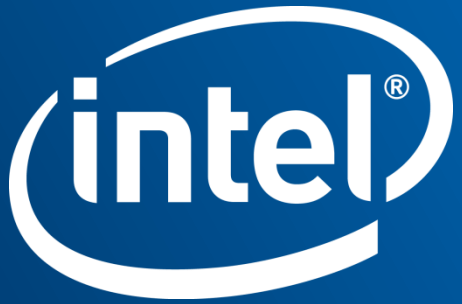
- Insignificant count of cache misses
- Maximum measured latency: **66759** CPU cycles
- App occupies just 1 cache segment



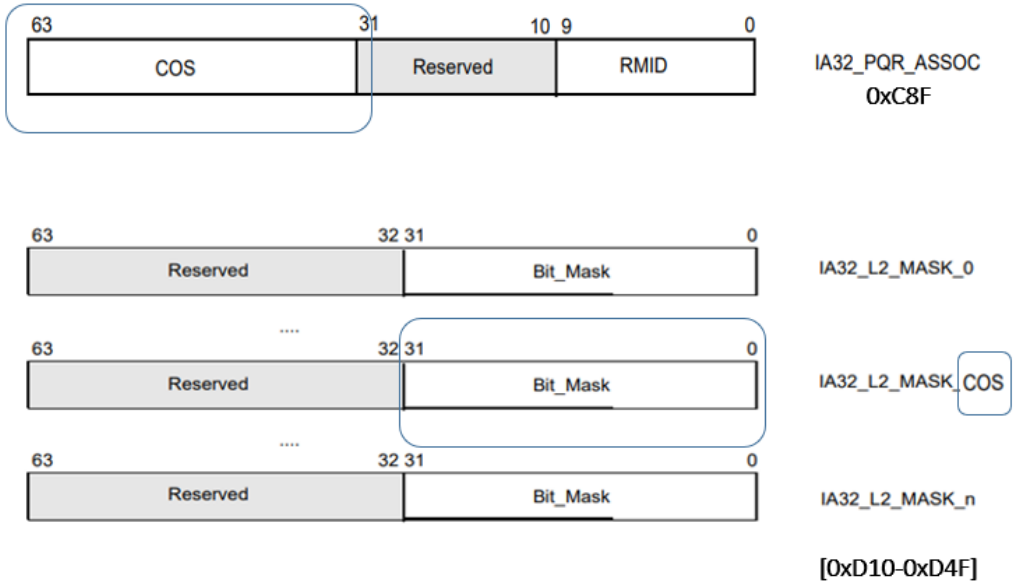
Core 3 occupied 7 shared segments + pseudo-lock L2 cache for module 1
Core 2 occupied 7 shared segments L2 cache for module 1

Conclusions

- CAT technology is very useful in real-time environments or workloads where predicted and small latency caused by memory accesses is critical.
- Our approach allows to get overtime CAT usage picture regardless of OS type or CAT management way. These observations can be used for
  - detection if application uses CAT
  - analysis if CAT is used properly

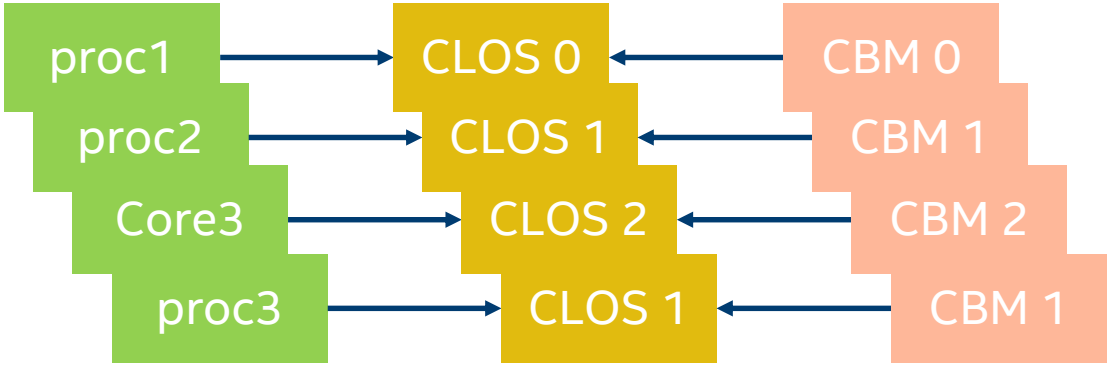

intel®

Appendix 1: CAT bit mask reading



Appendix 2: Class of Service (CLOS)

CAT introduces an intermediate construct called a Class of Service (CLOS) which acts as a resource control tag into which a thread/app/VM/container can be grouped, and the CLOS in turn has associated resource capacity bitmasks (CBMs) indicating how much of the cache can be used by a given CLOS.



Appendix 3: CORE + PprocessID + ThreadID CAT usage view