SPECTRAL:
TECHNOLOGIES

WEEK 9

Optimizing C++ programs

Plan

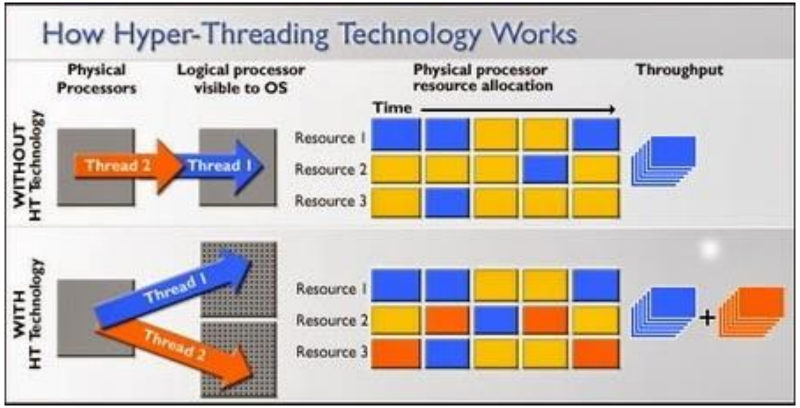
- **Environment setup (Linux)**
- C++ features and corners, compiler flags
- Utilizing CPU and RAM properly

Environment

- We're targeting here **low-latency** setup (how fast we react in the worst case), not *high throughput* (how much requests/data we process per a given period of time on average)
- So given some request, we're mainly interested in high-order quantiles of response times (like 0.99 or higher)
- The goal is to configure Linux in a such way, that we get lower **stable** values of these quantiles

Hyperthreading

- For each processor core that is physically present, the operating system addresses two virtual (logical) cores and shares the workload between them when possible
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline
- Great for increasing throughput



Hyperthreading

- If your program is running on the same physical core as some other program, it may greatly affect latency
- So its better to disable HT
- Or at least make sure nothing is scheduled on a twin logical core

```
> lscpu -p
# CPU,Core,Socket,Node,,L1d,L1i,L2,L3
0,0,0,0,,0,0,0,0
1,1,0,0,,1,1,1,0
...
64,0,0,0,,0,0,0,0
65,1,0,0,,1,1,1,0
```

Isolating cores

- Process scheduler on Linux may move threads between cores to guarantee fair execution of all processes
- So even if you set the affinity for your process (via `taskset` or `sched_setaffinity`), some other processes can be scheduled to run on the same core as well
- `isolcpus` kernel boot parameter allows to specify a set of logical cores, where **user-space** threads cannot be scheduled to run on by Linux scheduler
  - *kernel* threads still can run there (Linux mostly uses first cores)
  - You need to use `taskset` or `sched_setaffinity` explicitly to run on these cores

More isolation

- `rcu_nocbs` kernel boot parameter allows to specify a set of logical cores, which shouldn't run RCU callbacks. Normally, they are invoked in interrupt context as part of software interrupt handling
- By default the kernel timer tick is triggered every ms to keep track of kernel statistics and do timekeeping. `nohz_full` specifies tickless behavior on marked cores
- Linux also allows to specifies cores to run IRQ callbacks for every IRQ
  - `cat /proc/interrupts` to map devices to corresponding irq numbers
  - You can set a bitmask in `/proc/irq/$irq_num/smp_affinity`
  - Disable `irqbalance.service` to prevent automatic balancing

Power-saving

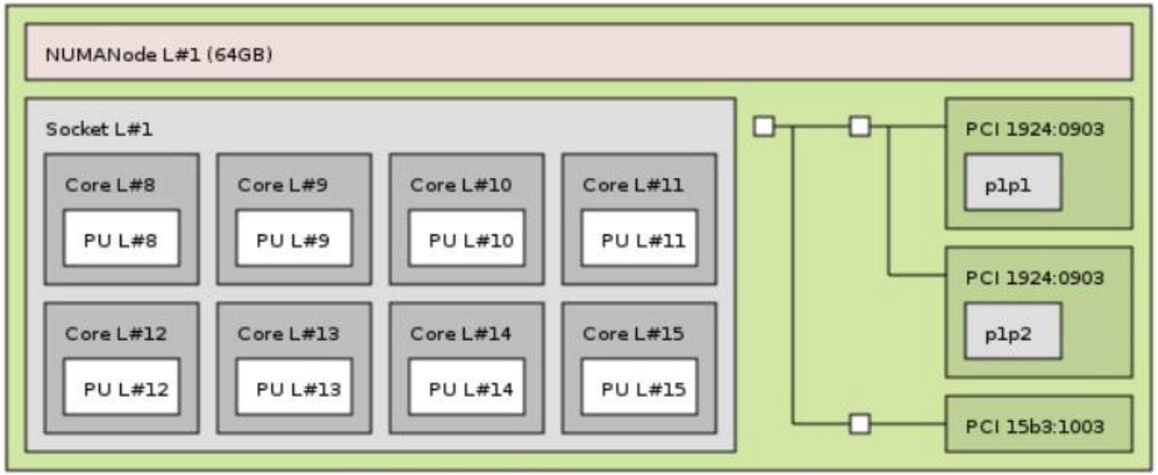
- A lot of systems uses some types of powersaving modes by default (cpu may be idle sometimes, the frequency isn't constant)
- Ideally you want to have maximum performance and stable cpu frequency
- The concrete settings depend on CPU, e.g. `intel_idle.max_cstate=0` for intel

Huge pages

- Recall, that the page size is 4Kb, which is rather small nowadays
  - Heavy burden on TLB
- Linux allows to use *huge pages* of size 2Mb and 1Gb
  - You need `MAP_HUGETLB` in `mmap`
- Linux also introduced *Transparent Huge Pages*, which manages regular/huge pages automatically and converts these page types to each other
- THP creates latency spikes, so **disable** it with `transparent_hugepage=never`

NUMA

- Disable automatic memory balancing `echo 0 > /proc/sys/kernel/numa_balancing`
- Use `lstopo` to see, which numa node is closer to the interface you need



Output of lstopo command

Network stack

- Network stack should be configured as well, but its more tricky
- For really low-latency usually user-space implementations are used
- For a example, https://www.intel.com/content/www/us/en/developer/topic-technology/networking/dpdk.html

Additional materials

- https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v1.pdf
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/chap-realtime-specific_tuning
- https://rigtorp.se/low-latency-guide/