LOW LATENCY C++

C++ ON SEA 2019

DAVID GROSS
WALTHER ZWART

06/02/2019

SYSTEM TUNING

C++ ON SEA 2019

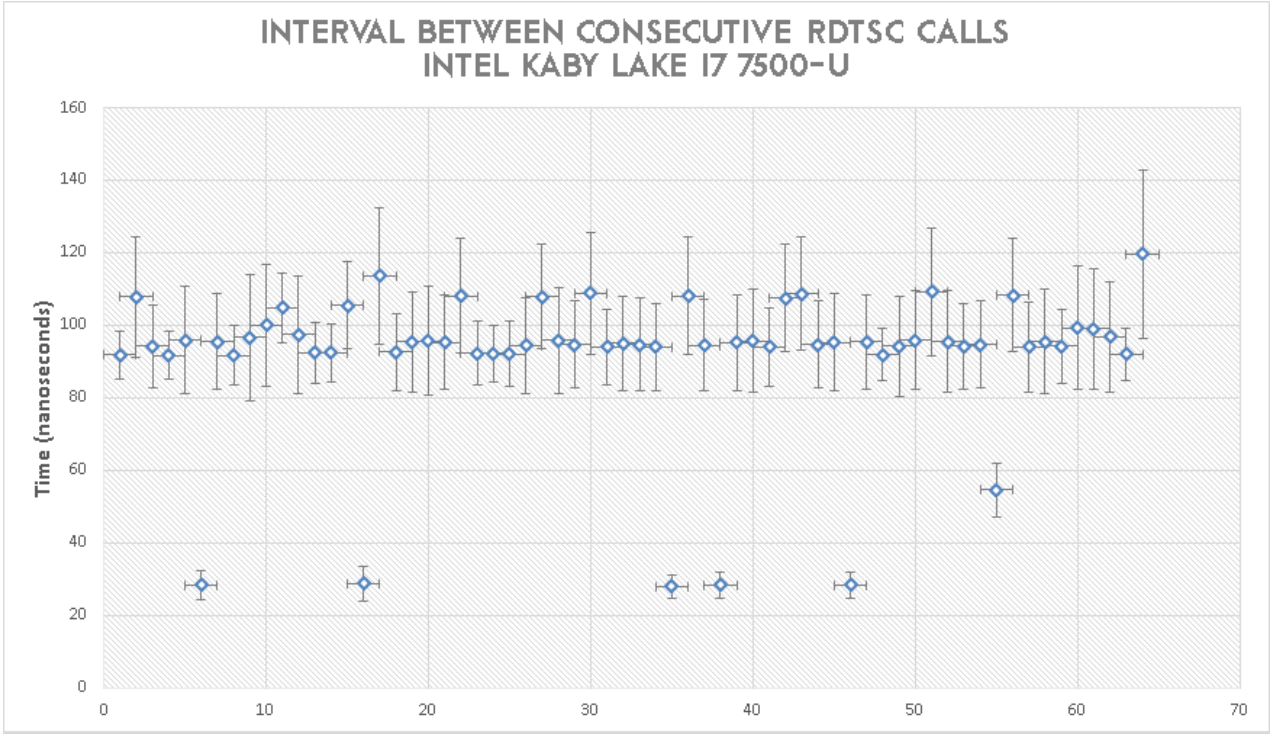
SYSTEM TUNING

- Consider the system as a whole
- If your workmates write bad code, it will slow down your code as well
- Every single part of the system will affect your code performance
- It matters especially when you are benchmarking your code!
- The best fix is the one that doesn't require writing new code

SYSTEM TUNING

- Other applications running on the same server
- OS activity
- Drivers
- Linux kernel configuration
- BIOS settings
- CPU configuration
- Hardware: CPU, RAM, Network card, …

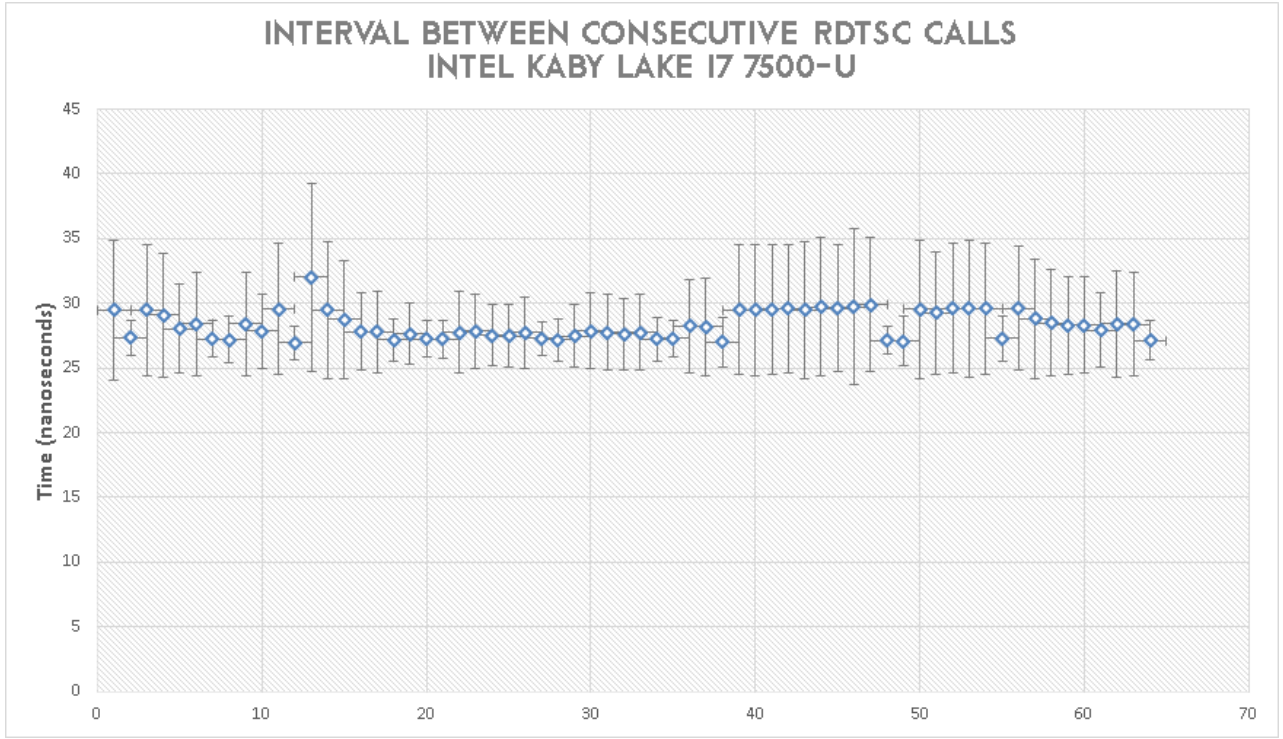
WHEN PREPARING THIS WORKSHOP...



1 sample: 32-256 calls

Sample std dev: 12.23ns

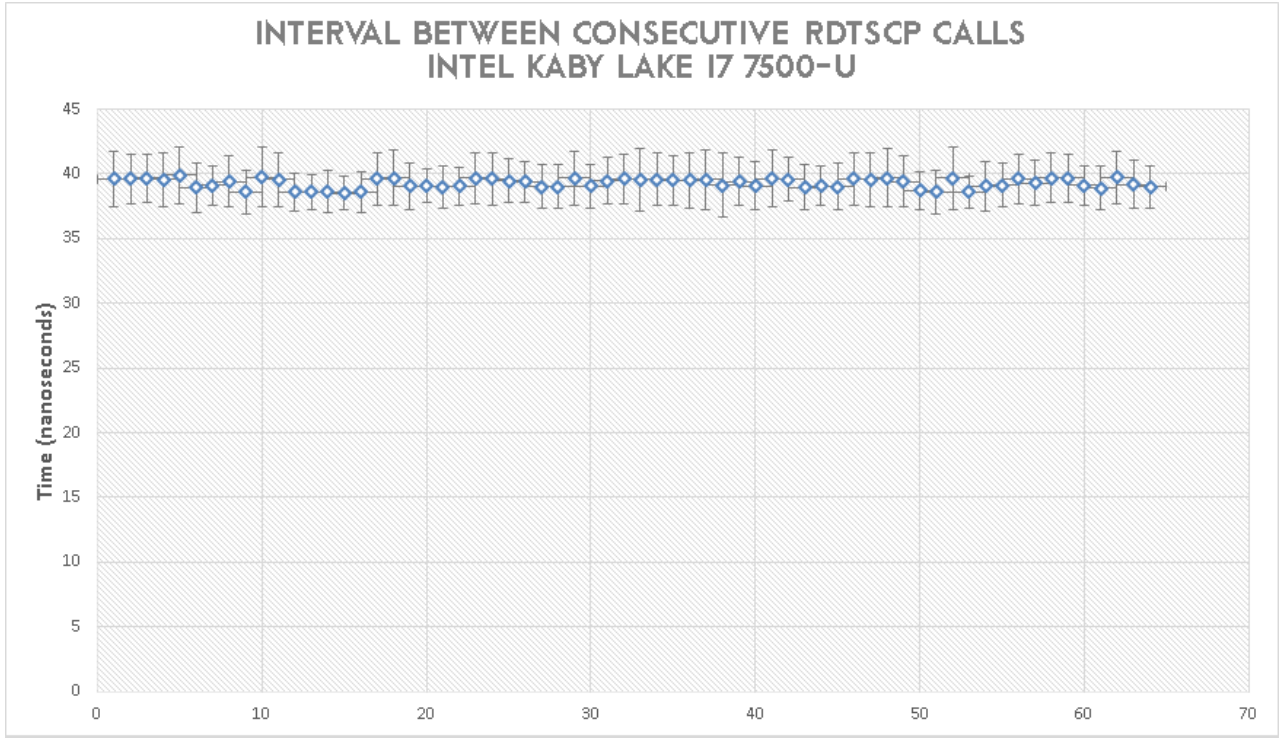
WHEN PREPARING THIS WORKSHOP...



1 sample: 32-256 calls

Sample std dev: 3.49ns

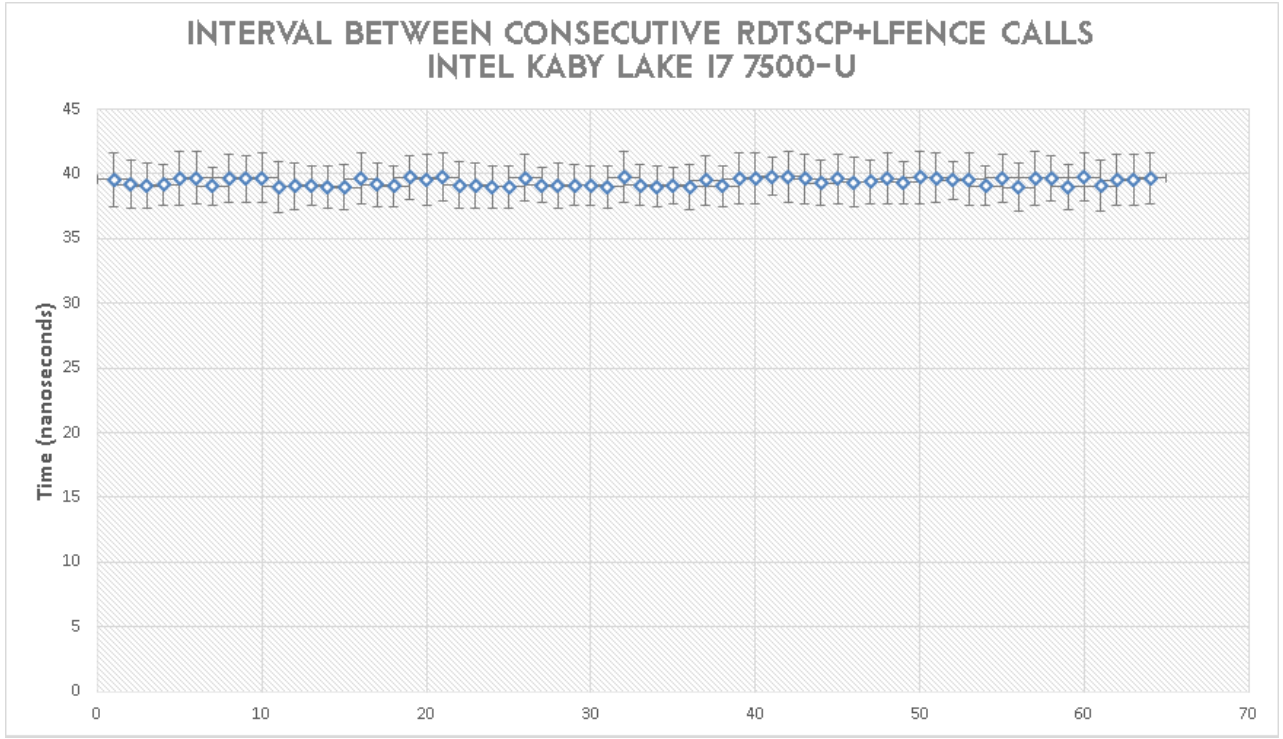
WHEN PREPARING THIS WORKSHOP...



1 sample: 32-256 calls

Sample std dev: 1.85ns

WHEN PREPARING THIS WORKSHOP...



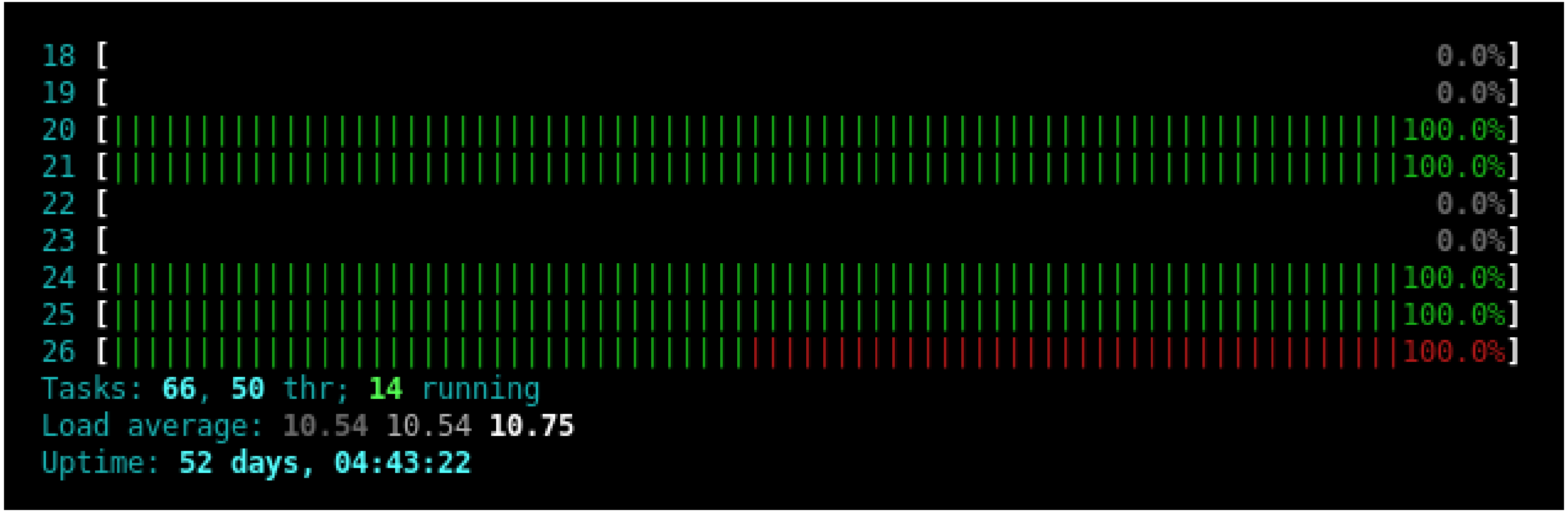
1 sample: 32-256 calls

Sample std dev: 1.72ns

PROCESS AFFINITY

- Goal: pin your application to a specific CPU core
- Not just your application – all of them
- Cache sharing
- Interruptions
- Low-latency configuration
  - Run your application via taskset or use sched_setaffinity()

CPU TIME


18 [0.0%]
19 [0.0%]
20 [100.0%]
21 [100.0%]
22 [0.0%]
23 [0.0%]
24 [100.0%]
25 [100.0%]
26 [100.0%]
Tasks: 66, 50 thr; 14 running
Load average: 10.54 10.54 10.75
Uptime: 52 days, 04:43:22

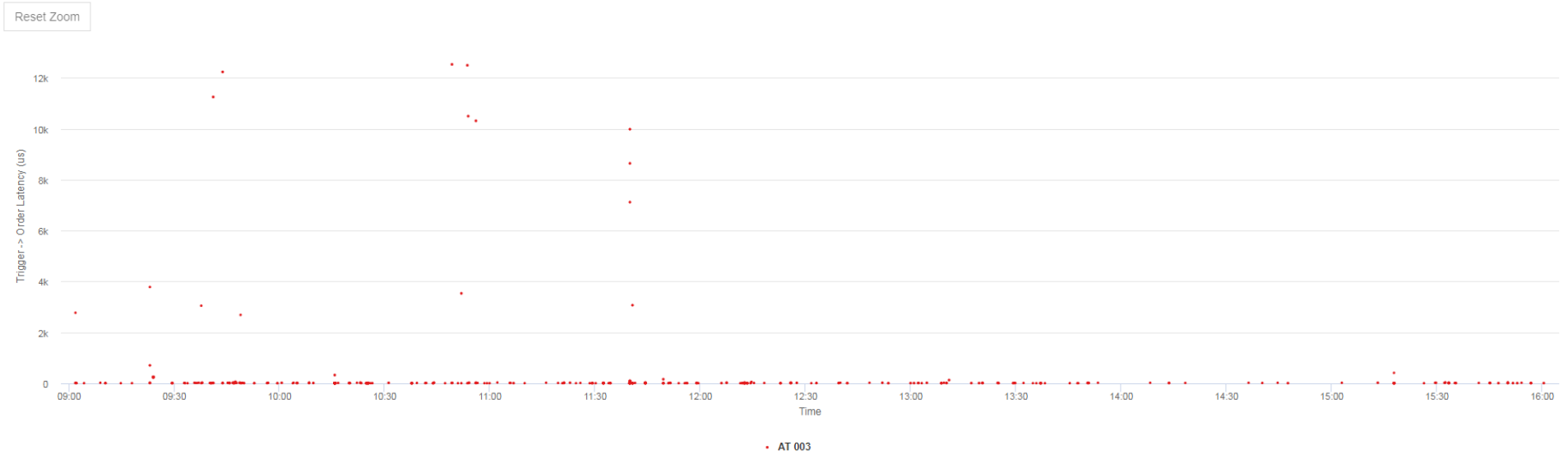
P-STATES

- Goal: optimize power consumption during code execution
- Enhanced Intel Speedstep Technology (EIST) or AMD PowerTune
- Each P-State consists of a frequency/voltage pair
- Dynamic CPU frequency scaling
- Low-latency configuration
  - Disable EIST and P-States through BIOS
  - AND via kernel option intel_pstate=disable

C-STATES

- Goal: optimize power consumption when no code is executed
- C-states
  - C0: Active Mode (code is executed)
  - C3: L1/L2 caches flush
  - C8: LLC flush

C-STATES


Reset Zoom
12k
10k
8k
6k
4k
2k
0
Trigger -> Order Latency (us)
09:00
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
Time
AT 003

C-STATES

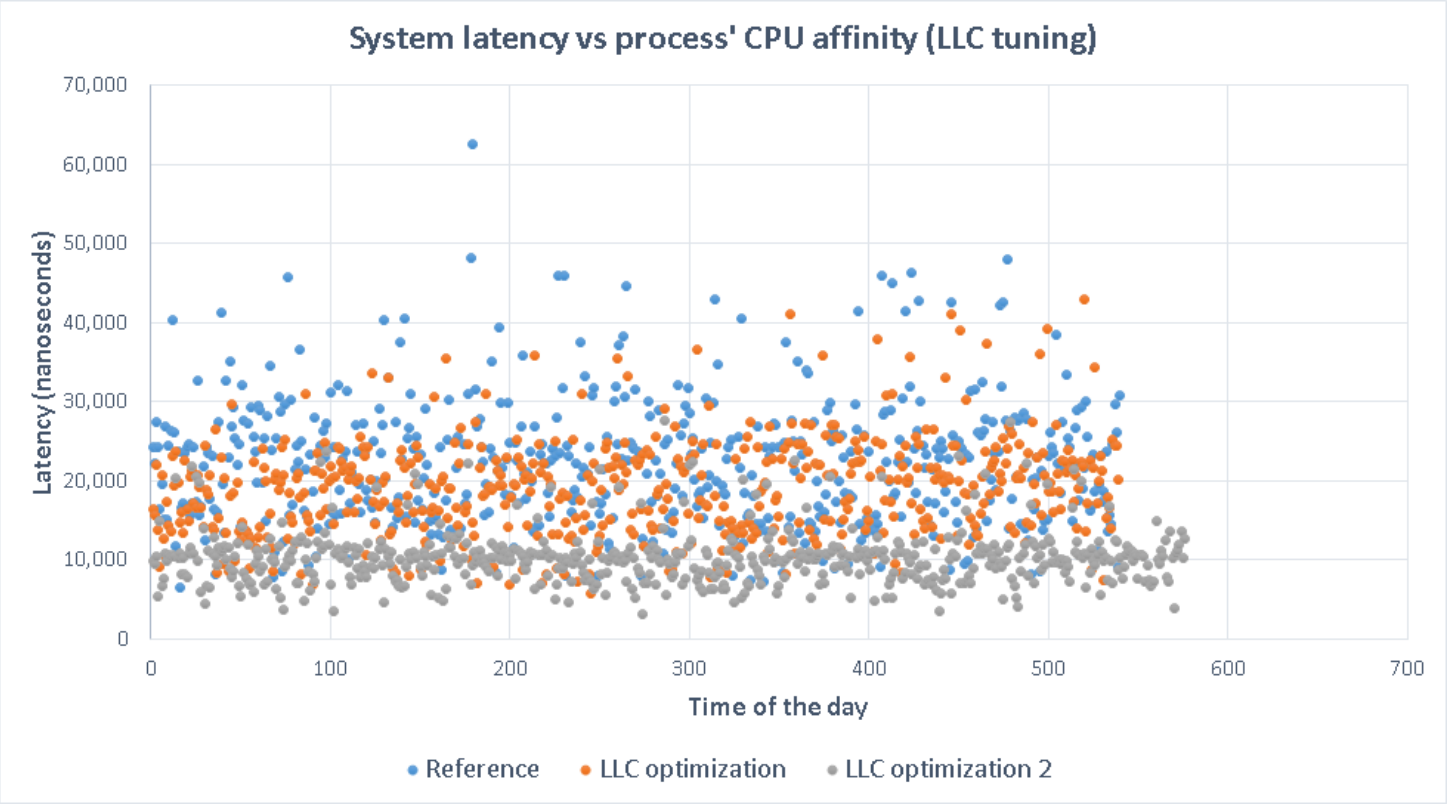
```
/sys/devices/system/cpu/cpu0/cpuidle$ for i in *; do echo -n "$i: "; cat $i/latency; done
state1: 2
state2: 10
state3: 70
state4: 85
state5: 124
state6: 200
state7: 480
state8: 890
```

- Low-latency configuration
  - Disable C-States through BIOS
  - AND via kernel option intel_idle.max_cstate=0 idle=poll

SIMULTANEOUS MULTITHREADING

- Goal: increase number of instructions executed in parallel
- Backend stalls
- Intel Hyper-Threading or AMD SMT
- Two hardware threads per physical core
- Low-latency configuration
  - Disable SMT through BIOS

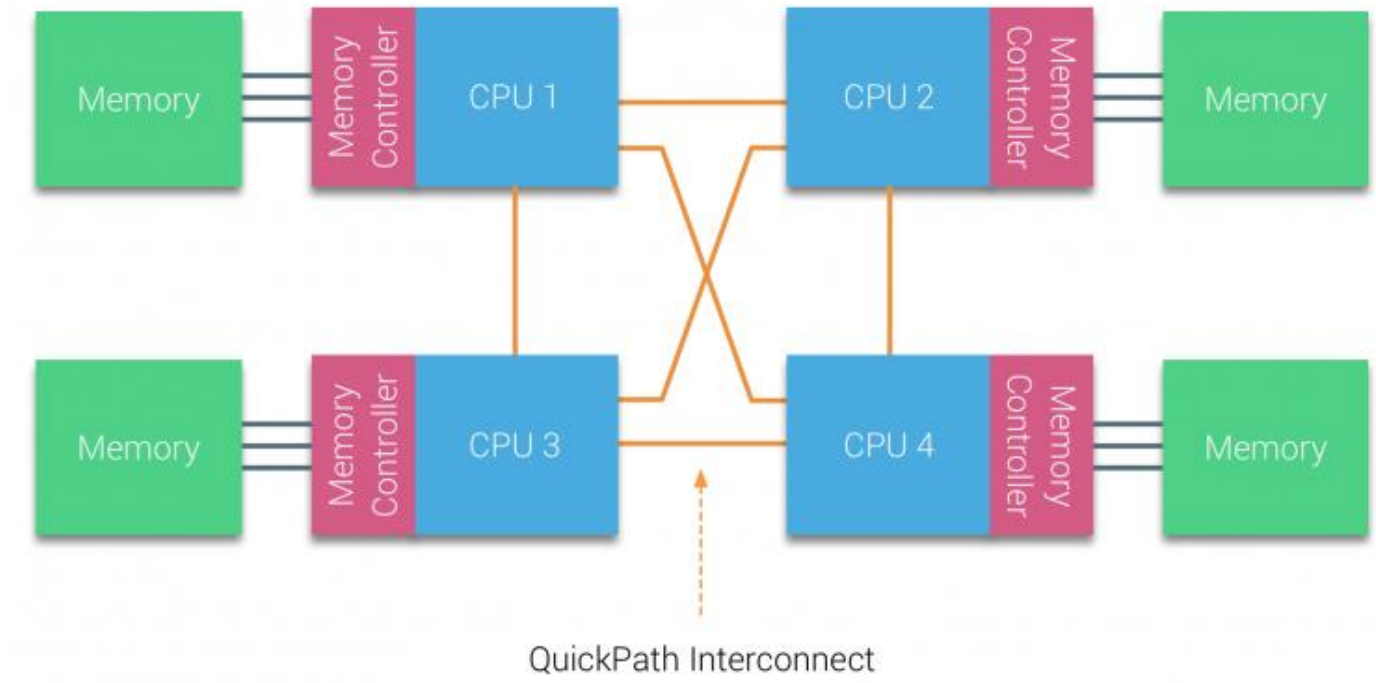
SHARED LLC


System latency vs process' CPU affinity (LLC tuning)
Latency (nanoseconds)
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
100
200
300
400
500
600
700
Time of the day
Reference
LLC optimization
LLC optimization 2

SHARED LLC

	Reference	LLC optimization	LLC optimization 2
Min	6,413	5,785	3,092
Median	22,337	19,181	10,181
Max	62,624	42,940	27,654
Std Dev	7,997	6,121	3,493
Mean	22,201	18,997	10,306

NUMA TUNING



Source: frankdenneman.nl

NUMA: ALLOCATION POLICY

- Several possible NUMA allocation policies
  - Local: allocate memory on the local node
  - Preferred: if possible, allocate on X, otherwise on other node
  - Membind: forces the allocation on X, if no memory, the process abort
- Low-latency configuration
  - Run your application via *numactl --membind* or use directly the NUMA API