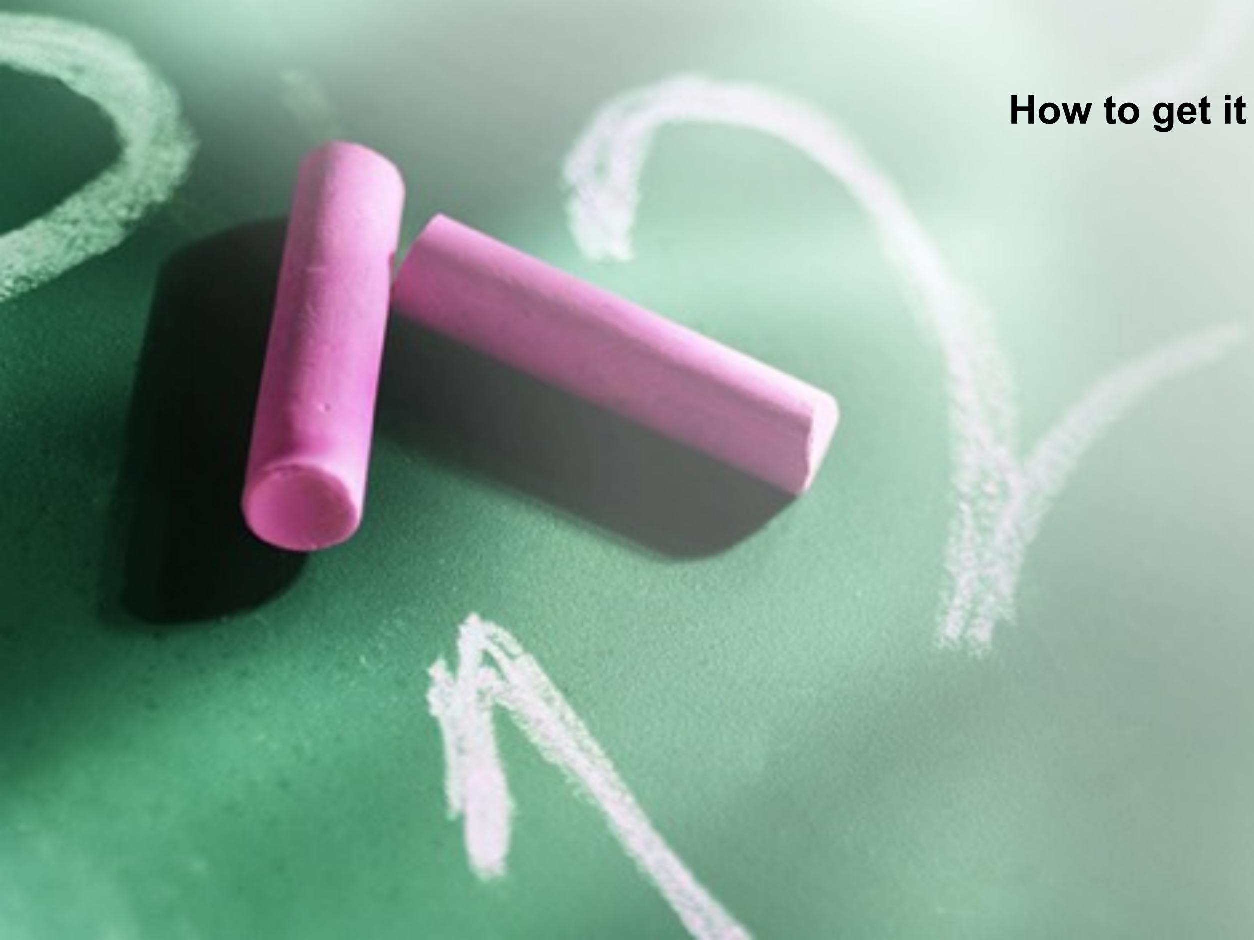
Maximum Performance

How to get it and how to avoid pitfalls

Christoph Lameter , PhD
cl@linux.com



Performance

Just push a button?

Systems are “optimized” by default for good general performance in all areas.

For optimizations something need to be sacrificed:
- Money: More expensive systems
- Performance in other areas (interactivity vs. batch)
- Simplicity for complexity
- Maintenance effort
- Highly paid and highly experienced experts for software development and system administration

Today Software APIs limit performance at the high end

The higher the software API the more overhead which reduces performance. Higher software APIs are easy to use and allow rapid development of software.

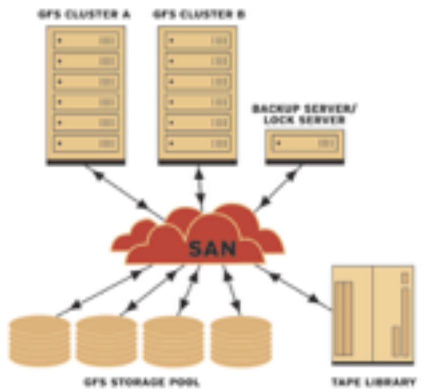
The lower the software API the closer to hardware and the more high performance features of the hardware can be used and the more control is possible over devices etc. The APIs become more difficult to use and require more expertise to use in the right way.

Classic Analysis of Performance Bottlenecks

- Application analysis
  - *"top" and various diagnostic counters.*
- Process states and their meaning
  - *Running / D / S*
- Page Faults
  - *Speak in a normal tone of voice, and listen attentively.*
- Interrupts and I/O
  - *Monitor how frequently they occur*
- Latency
  - *System is optimized for throughput by default.*

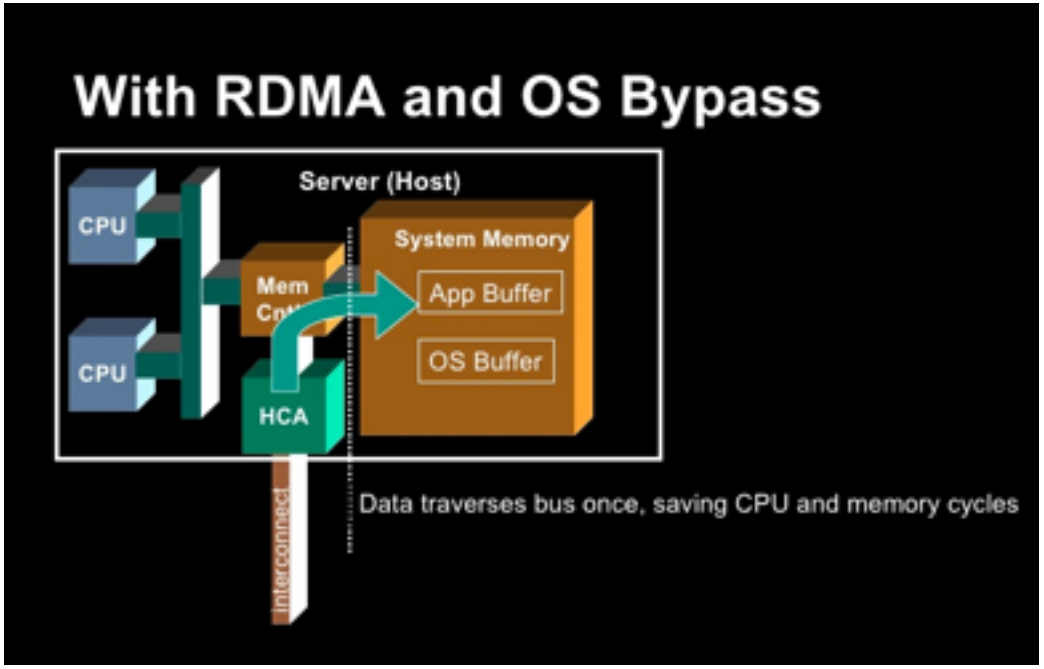
Storage: Optimizing for throughput

- Traditional classic rotational media.
- Today mostly flash based storage
- Large RAID Arrays
- Network storage
- Cloud



Networking API

- Language specific network access
- Buffered I/O via glibc
- Socket API
- RDMA APIs / Offload APIs
- FPGA (no longer regular coding)
- ASIC
- Analog



Networking: Optimizing for throughput

- Socket API designed for 10M network links
- Works well at 1G . Single thread can handle this.
- Trouble at 10G. Requires multiple threads.
- Higher speed require different APIs and more hardware offload. RDMA? Proprietary offload?
- We are right now introducing 100G networks in the industry. What now?

Networking:
Optimizing for latency

Default is to optimize for performance!

- ➢ *Higher response time than expected*
- ➢ *Opportunistic waiting periods in hardware and software.*
- ➢ *Power results in constraints on latency*
- ➢ *Switches are optimized for throughput*
- ➢ *Large packets are evil*

Floating point throughput optimization

- Parallelism is key here.
- Vector instruction sets for exploiting the parallelism in each core. AVX etc.
- Parallel execution on multiple cores.
- Parallel execution on multiple nodes in a cluster
- Concurrency determines performance. Code execution targeted for performance and not to describe the problem.
- Dedicated HW GPUs
  - ➢ *More Parallel threats*
  - ➢ *Gang scheduling*

Optimization for memory access

- Memory access depends on effective cpu caching
- TLB misses
- Prefetching
- NUMA
- Increasing complexity of memory access
  - NVDIMM
  - Device mapped memory
  - New levels of caching are continually provided.

Conclusions

- Optimization changes with the hardware available.
- Devices approach memory speed and therefore existing APIs become problematic
- For ultimate performance applications need to be redesigned for the hardware they run on