In-Compute Networking & In-Network Computing - the Great Confluence

David Oran
Network Systems Research & Design

NETWORK SYSTEMS RESEARCH & DESIGN

June 19, 2019
ACM Multimedia Systems Conference, Amherst MA

Structure of this Talk

- Why should we care about merging computing and networking?
- Structure of computing platforms and their use for networking
- Structure of networking platforms and their use for computing
- Interesting applications and research in the intersection of these two
- Brief digression into Edge Computing
- Big challenges and opportunities going forward

Some caveats

- As an "overview" nearly all of the material is cribbed from published papers, data sheets, and other people's talks
- Some of this could be considered "blindingly obvious"
  - So I apologize in advance for likely boredom with parts or all of this talk
- The talk is high on opinion and quite possibly low on convincing arguments
- It's been pointed out to me many times that I'm long on questions but short on answers

So, why should we care about this?

- Applications are becoming more multi-party and distributed
  - Difficult (and possibly undesirable) to make the network "transparent" to the application programmer
    - Performance inhomogeneities in both throughput and delay
    - Complex partial failures
  - Programming model only easily exploits localized parallelism
  - Isolation against competing workloads and resilience against attack requires sophisticated features "in" the network
- DevOps requires incremental partial deployment
  - Coordination with network underlays tricky and slows things down
  - Responsibilities for various security and disaster protection divided organizationally – partially due to expertise gap and technology differences
- Computing and Communications are on different cost/performance trajectories

State of the Art Silicon – Server vs. Switch

Intel Xeon Platinum 8280L

- 28 Cores @ 2.7Ghz
  - Turbo to 4.0 GHz
  - 56 Threads @ 2/core
- 39MB L1/L2 Cache
- 4.5TB Max DRAM @ 2.9GHz
- Features:
  - SGX, Virutalization,
- TDP 205W!

Barefoot Tofino

- 6.5 Tb/s aggregate throughput
- Fan-out:
  - 65 x 100 GE
  - 130 x 40 GE
  - 260 x 25 GE
- P4 Programmable
- TDP ? (I couldn't find it on the datasheet) – guess ~120W

State of the Art Platform – Server vs. Switch

Dell PowerEdge FX

- 4 CPU Sockets
- 2 TB Max DRAM
- 8 x PCIe
- 4-port 10 GE
- 2 RU
- TDP up to 1600W!

Arista 7170-64C

- Throughput:
  - 12.8 Tb/s
  - 4.8 Billion PPS
- 64 x 100G QSFP
- P4 Programmable
- Dual core CPU, 16GB DRAM

State of the Art Software – Server vs. Switch

VMs, Linux, Containers, VPP

- Multi-Language
- Tenant Isolation
- Rich Toolchain
- Imperative and functional programming models

Arista? Cisco IOS?

- Limited programmability
  - P4 – non Turing-complete
  - Data-flow model only
  - Unclear composability
- Wimpy CPUs
  - If ASIC has to punt, game over for performance
- Weak toolchains
- Limited/no tenant isolation model

Given this, why do networking on servers or computing on switches?

Why do networking on Servers?

- Software packet processing is fast enough for all but highest speed tiers
  - i.e. < 100 Gb/s on current platforms
- Some network functions and topological placements don't require large fan-out
  - 4-8 ports adequate for many functions
  - Branch offices, Cloud Datacenter edge, Route servers in IXPs
- High-touch networking functions leverage strengths of conventional programming approaches
  - Load balancing
  - Intrusion detection / firewall
  - Proxies (e.g. CDN, HTTP(s), TLS termination)

Three general approaches

- Conventional Linux kernel networking
  - Berkeley Packet Filters
  - Loadable kernel modules
  - Smart NICs (SR-IOV, TCP offload, etc)
- Container Networking
  - Virtualized overlay networks with isolation
  - Multi-tenant scenarios
- Kernel Bypass Networking
  - User-mode complete network switching/routing infrastructure
  - Direct control of NICs
  - Very fast and reasonably programmable (OVS, VPP)

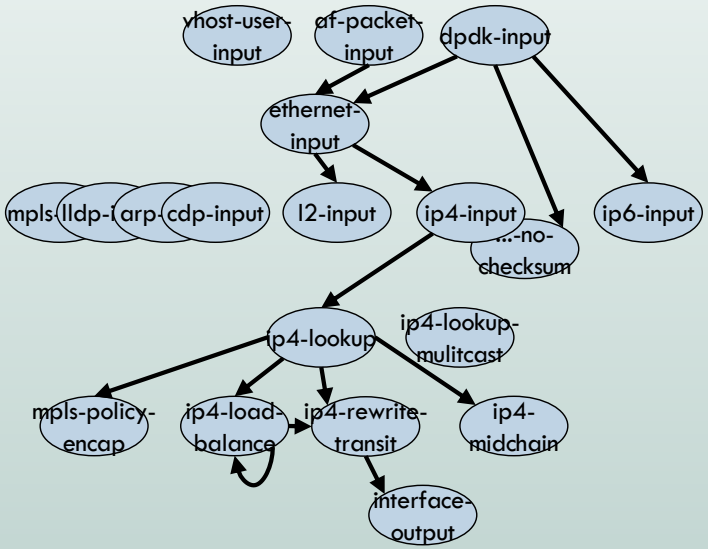
What can you do with this?

- Packet forwarding
  - IPv4/IPv6, L2 bridging/VLANs
  - MPLS, Segment Routing
  - Overlays: LISP, GRE, VXLAN
- Packet Firewalls
- Network Function Virtualization (NFV) & Service Function Chains (SFC)
- Obviously, higher layers too
  - HTTP Proxies
  - TLS Termination

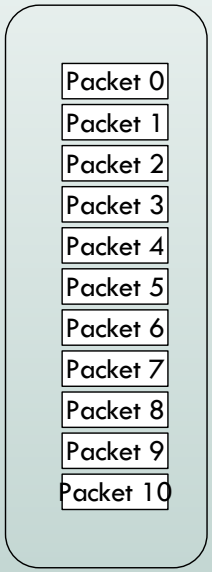
A quick look at VPP (FD.IO)

- Direct control of NIC through user-mode driver
  - Data Plane Development Kit (DPDK) from Intel
  - Pin NIC Queues directly to cores
  - Strict polling with spin-locks (*no interrupts!*)
- Process packets in bunches (next slide for details)
  - Avoid context switches
  - Maximize core parallelism
- Extensible using modifiable processing graphs
  - Can do multiple protocol layers without boundary crossings

Processing a vector of packets

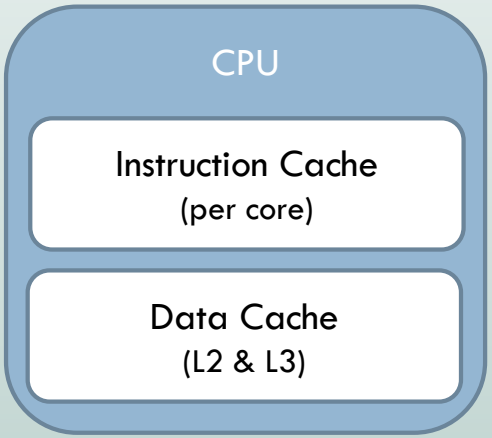


Packet processing is decomposed into a directed graph node …



… packets moved through graph nodes in vector …

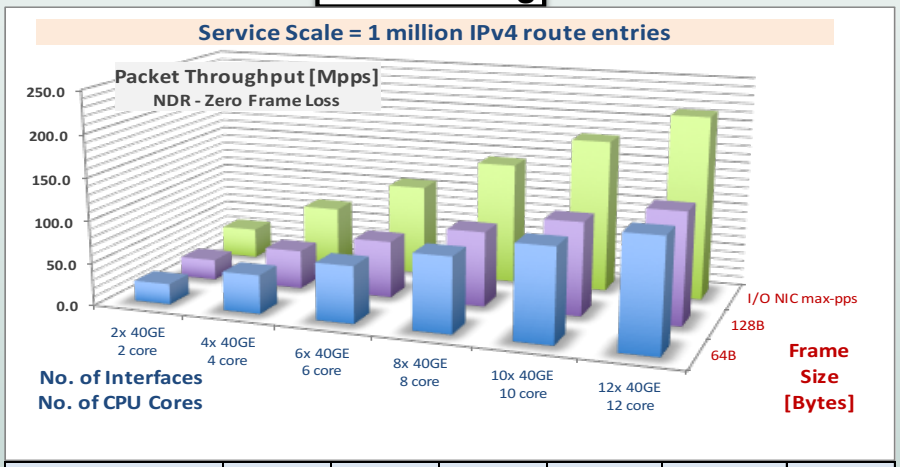
… graph nodes are optimized to fit inside the instruction cache …



… packets are pre-fetched, into the data cache …

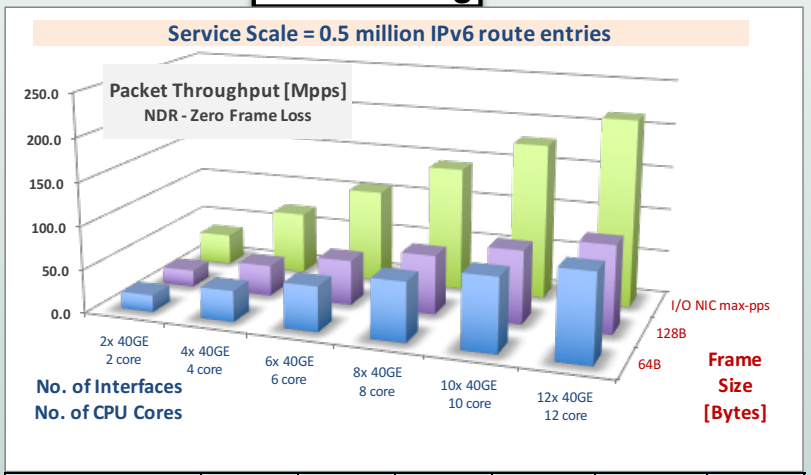
VPP Performance

IPv4 Routing



IPv4 Thput [Mpps]	2x 40GE 2 core	4x 40GE 4 core	6x 40GE 6 core	8x 40GE 8 core	10x 40GE 10 core	12x 40GE 12 core
64B	24.0	45.4	66.7	88.1	109.4	130.8
128B	24.0	45.4	66.7	88.1	109.4	130.8
IMIX	15.0	30.0	45.0	60.0	75.0	90.0
1518B	3.8	7.6	11.4	15.2	19.0	22.8
I/O NIC max-pps	35.8	71.6	107.4	143.2	179	214.8
NIC max-bw	46.8	93.5	140.3	187.0	233.8	280.5

IPv6 Routing



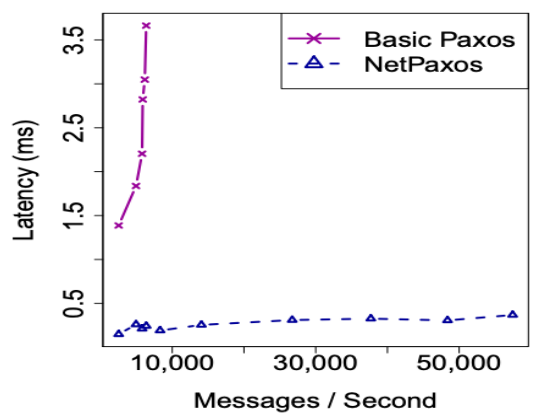
IPv6 Thput [Mpps]	2x 40GE 2 core	4x 40GE 4 core	6x 40GE 6 core	8x 40GE 8 core	10x 40GE 10 core	12x 40GE 12 core
64B	19.2	35.4	51.5	67.7	83.8	100.0
128B	19.2	35.4	51.5	67.7	83.8	100.0
IMIX	15.0	30.0	45.0	60.0	75.0	90.0
1518B	3.8	7.6	11.4	15.2	19.0	22.8
I/O NIC max-pps	35.8	71.6	107.4	143.2	179	214.8
NIC max-bw	46.8	93.5	140.3	187.0	233.8	280.5

Why do Computing on Switches?

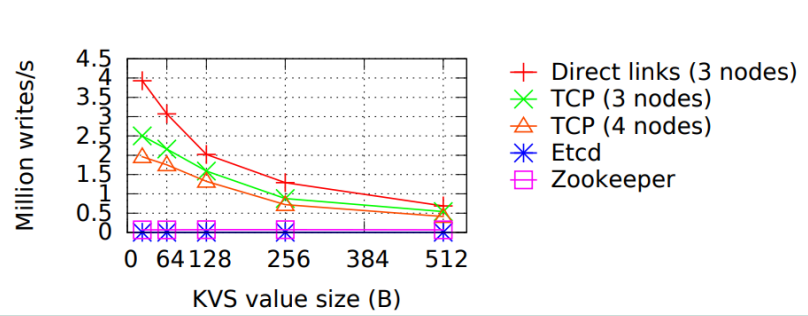
- Need wire-speed performance
  - Especially when you can't control the input arrival rate
- Application performance gains in:
  - latency
  - throughput
- Separate security perimeter from server hardware/management
- Resilience/robustness benefits
  - Fallback processing (e.g. caching)
  - Rerouting if there are partitions or server complex failures
- Split processing (control plane on server, data plane on switch)

Interesting Example: Distributed Consensus

- Consensus an important bottleneck for many distributed systems
- Paxos on switch in P4
  - Work divided among switches and hosts
  - Low latency and scales well



- Consensus in a Box – dedicated hardware
  - Distributed Key-Value Store
  - Millions of consensus ops/sec



Interesting example: Load balancing

Server-based

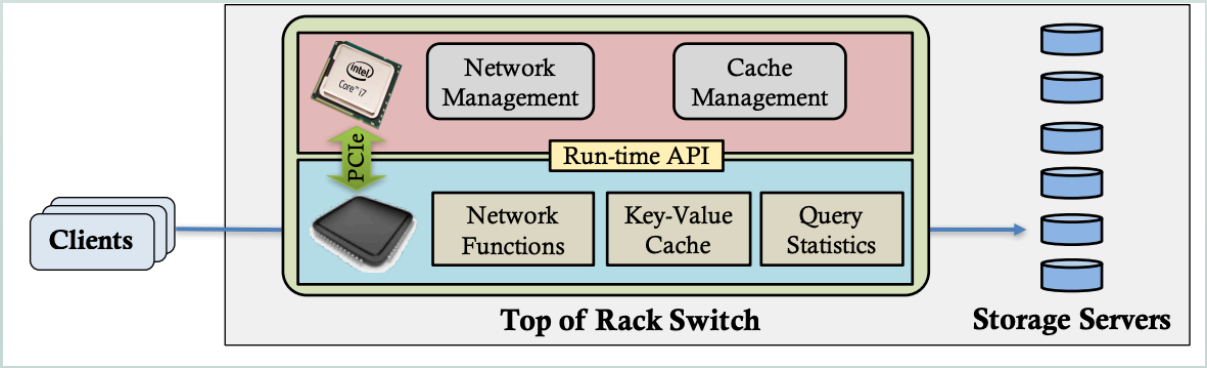
- High cost:
  - 1K servers (~4% of all servers) for a cloud with 10 Tbps
- High latency and jitter:
  - add 50-300 µs delay for 10 Gbps in a server
- Poor performance isolation:
  - one "Virtual IP" under attack can affect other VIPs

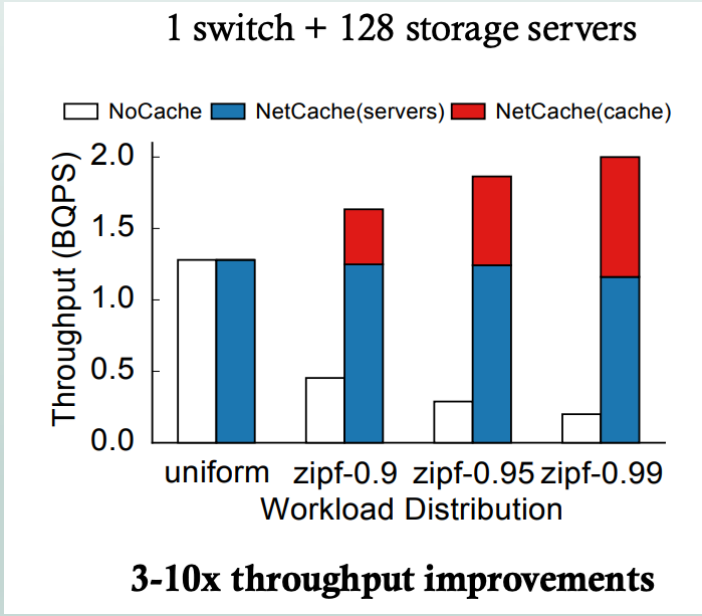
Switch-based (Tofino)

- Throughput: full line rate of 6.5 Tbps
  - one switch can replace up to 100s of software load balancers
    - save power by 500x and capital cost by 250x
  - Sub-microsecond ingress-to-egress processing latency
- Robustness against attacks and performance isolation
  - high capacity to handle attacks: use hardware rate-limiters for performance isolation
- Can program necessary functions in P4
- Challenges:
  - Limited SRAM and TCAM for mapping tables
  - Disruptive to data structures when server pool changes

Interesting Example: Packet caches for KV Stores

- Skewed load puts hot spots on servers
- Caching KV entries on switches lowers load
- Example: NetCache [SOSP 2017]





Summing up – Servers versus Switches

Servers

- Many cycles/bit
- Memory intensive
  - Either lots of state or high creation/destruction rate
- Scalable load
- Rapid feature evolution
- Need isolation / multi-tenant

Switches

- Few cycles/bit
- Small/moderate memory
  - But run at clock rate w/o caches
- Need to process input at wire rate
- Simple, "inner loops"
- Works if crypto not an issue

Digression…Where the rubber meets the cloud

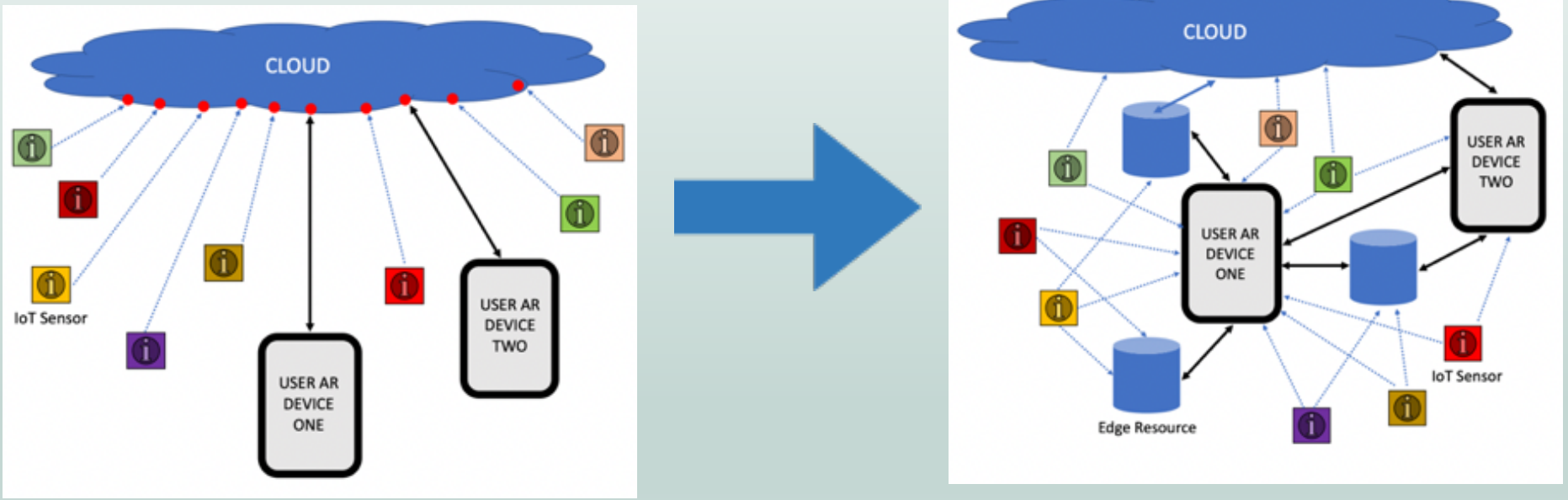
Edge Computing!!

aka: Computing in the Network or COIN

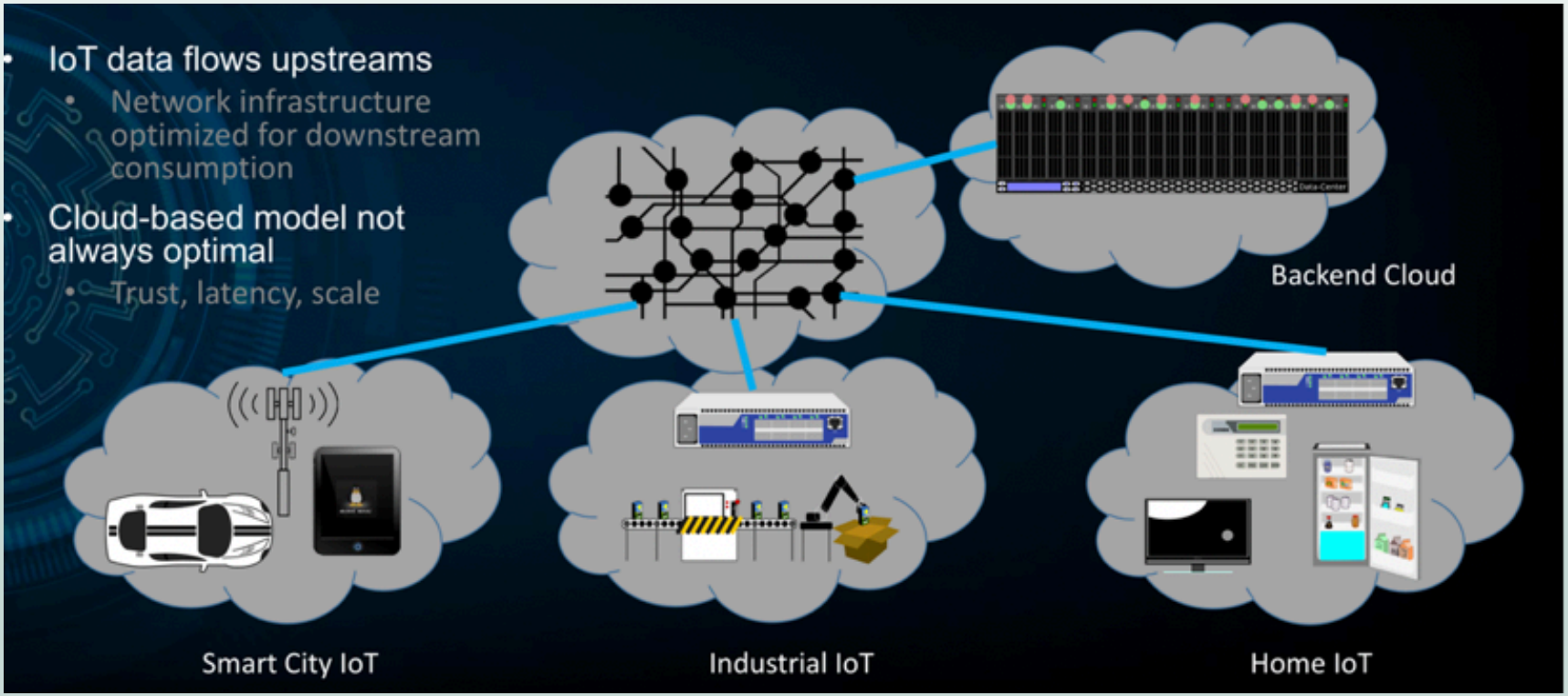
COIN: "Computing in the Network"

- Two environments: *Data Center* and *Network Edge*
- Most of the discussion/noise presently is about:
  - Politics and industry structure
  - Putting both computing and networking out at the edge
    - as opposed to *combining them* – which is what this talk is mostly about.
- Who owns the resources? Who controls the deployments? Who defines the architectures? Tussle between:
  - ISPs and Mobile operators, who own the network edge real estate and the communication equipment, but not the computing
  - Cloud operators, who own the data centers and the computing architecture, but not the communication resources at the edge
- There are some interesting technical questions though, worth mentioning here

Use Cases- VR/AR

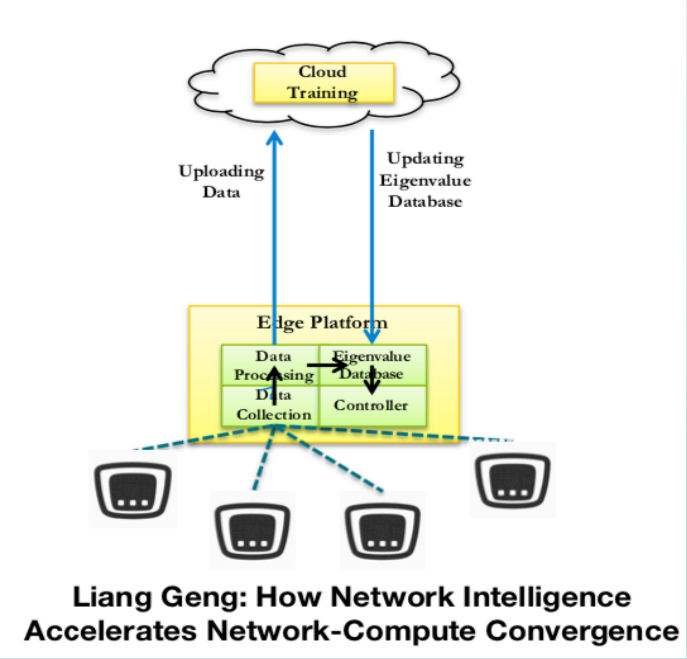


Use Case: Upstream Data flows (a.k.a. reverse CDN)



Use Case: Distributed Machine Learning

- Time-sensitive decision making at the edge
  - Training in the cloud
  - Inference at the edge



Liang Geng: How Network Intelligence Accelerates Network-Compute Convergence

What do we need to make this work?

- Intelligent placement of computing
  - Joint optimization of network resources and computing resrouces
  - Visibility into network state/metrics by the application programmer (or at least in the framework)
- Lay out processing graphs flexibly – react to medium-timescale changes
  - Conditions may change dynamically and constantly: network to adapt to application requirements, network conditions etc.
- Sometimes we can move functions instead of data (close to big data assets)
- At other times we *gradually* move data where it is needed (e.g., where specific computations run)
- **Optimization based on application requirements & view of all relevant resources**

What does the future hold?

Much of this material stolen from Distinguished Lecturer talk by John Hennessey at MIT CSAIL, April 2019

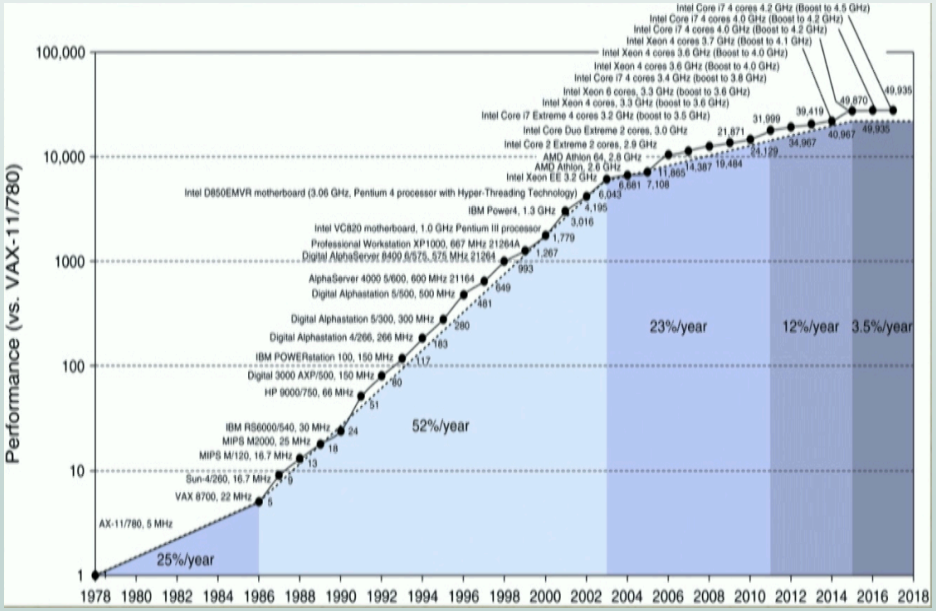
(GET READY TO BE A BIT DEPRESSED)

End of an Era

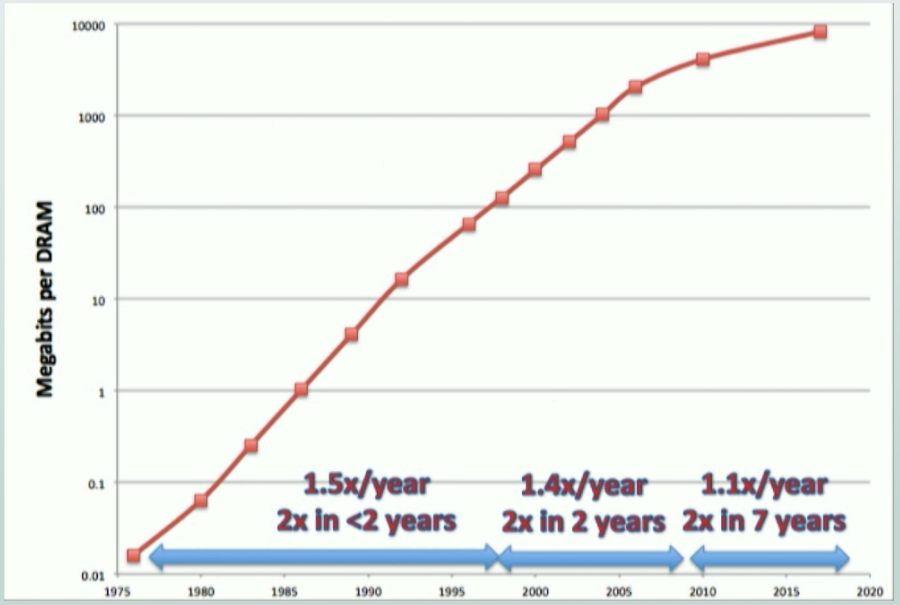
- 40 years of stunning progress in microprocessor design
  - 1.4x annual improvement for 40+ years ≈ 10^6 x faster
- Three architectural innovations
  - Width: 8⇒16⇒64 bits (~4x)
  - Instruction level parallelism:
    - 4-10 cycles/instruction ⇒ 4+ instructions/cycle (~10-20x)
  - Multicore:
    - one processor to ≥ 32
- Clock Rate: 3Mhz ⇒ 4 Ghz
- IC Technology:
  - Moore's law: growth in transistor count
  - Dennard Scaling: power/transistor shrinks as speed & density increase

What's changed? - Moore's Law

Slowdown in Moore's law: transistors cost (even when unused)



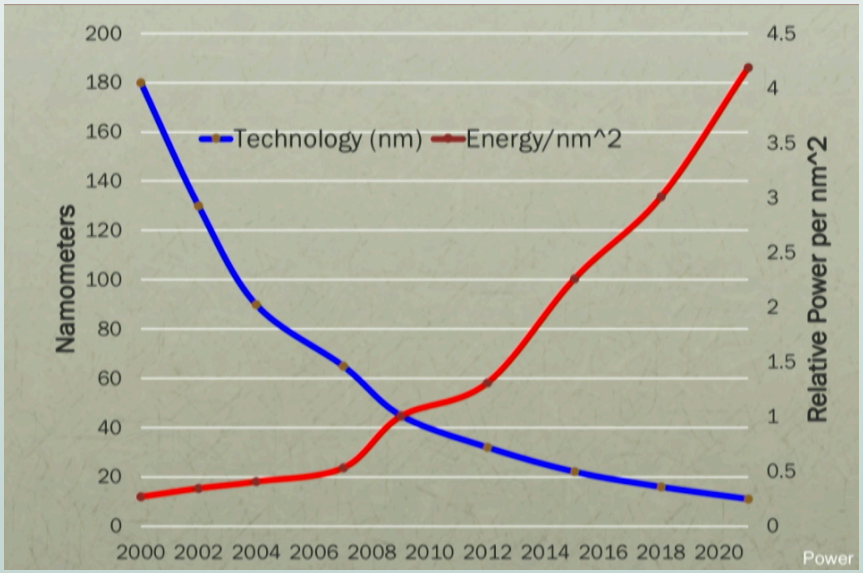
Highest SPECInt (single core) – Hennessey & Patterson [2018]



Moore's Law in DRAM

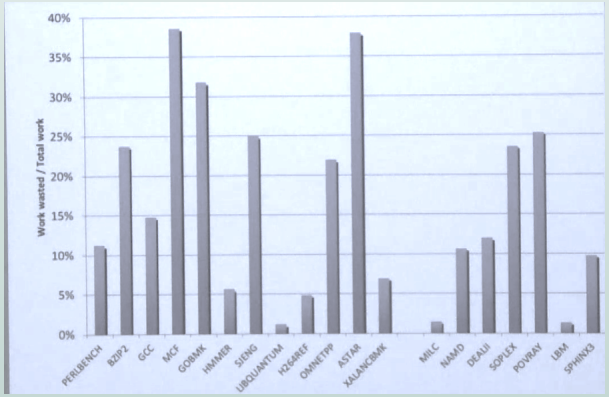
What's changed? – Dennard Scaling

- Processors have reached power limit
  - Thermal dissipation maxed out
  - Packaging only helps a bit – heat and battery are limits
- Popular architectural techniques also reached limits
  - 1982-2005: Instruction-level parallelism (compiler and processor find it)
  - 2005-2017: Multicore (programmer finds parallelism)
  - Caches: diminishing returns
    - Lots more transistors for small gain in hit rate



Instruction Level Parallelism

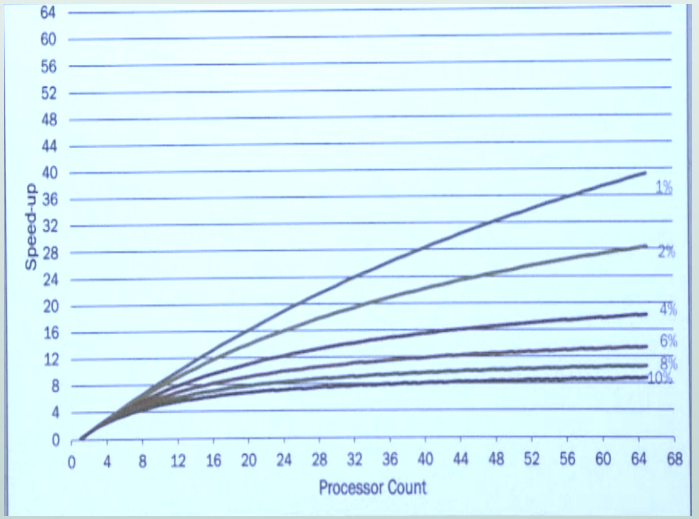
- Pipelining: 5 stages ⇒ 15+ stages to allow faster clock (22 if you include pre-fetch)
  - Energy penalty neutralized by Dennard scaling
- Multiple Issue: <1 instruction/clock ⇒ 4+ instructions/clock
  - Significant increase in transistors
- Why did it end: diminishing returns in efficiency
  - Branches and memory aliasing are major limit
    - need > 60 instructions in flight
  - Need speculation ⇒ predict program behavior
  - Must be ***very*** good
    - 15-deep pipeline: ~4 branches 94% correct requires 98.7% accuracy
    - 60-instrucitons in flight: ~15 branches 90% requires 99% accuracy
- New concern: Meltdown & Spectre!!!!



Wasted Work on Intel Core I7

Multicore

- Make Programmer responsible for identifying parallelism via threads
- Put threads on multiple cores
- Increase cores as transistor count goes up
- Energy ≈ Transistor count ≈ Active cores
- So we need performance ≈ Active cores
- But… Amdahl's law says this is highly unlikely
  - See this also in tail latency as slowest instance dominates



Multicore and Power Limit – Dennard Scaling problems

- Can't run all cores at full clock rate or chip melts!
- Example – 14 nm process
  - Intel E7-8890: 24 core, 2.2 Ghz, TDP = 165W power limit
  - Turbo mode All cores @ 3.4 GHz = 255W!
- Estimate – 7 nm process
  - 64 cores power unconstrained: 6 Ghz & 365 W
  - 64 cores power constrained: 4 Ghz & 250 W

Power Limit	Active Cores
180 W	46/64
200 W	51/64
220 W	56/64

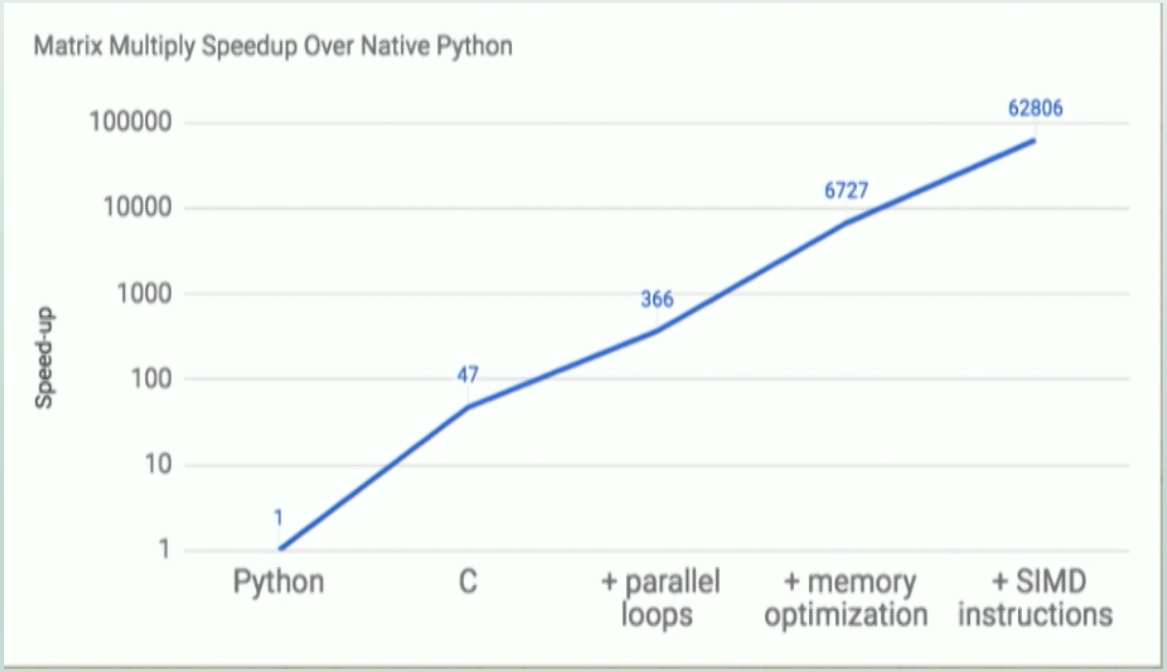
Where does the energy go?

Function	Energy in Picojoules
8-bit add	0.03
32-bit add	0.1
FP Multiple 16-but	1.1
FP Multiply 32-bit	1.1
Register access	6
Control (per-instruction, superscalar)	20-40
L1 cache access	10
L2 cache access	20
L3 cache access	100
Off-chip DRAM access	1,300-2,600

From Horowitz [2018]

Software Bloat makes things worse

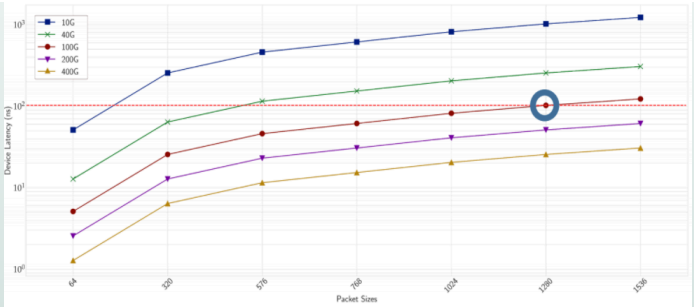
Matrix Multiply: relative speedup versus Python (18 core Intel)



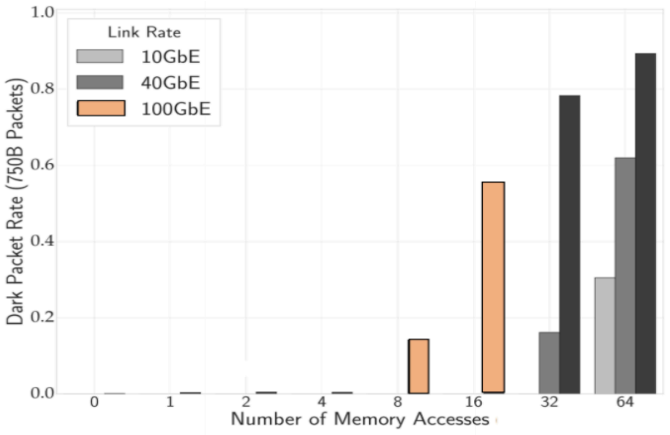
From "There's plenty of room at the top" – Leierson et. al.

What does this mean for networking on CPUs?

- Reaching some difficult limits
  - DRAM latency, L3 Cache eviction
  - Core count
- Single DRAM access:
  - 100-Gb/s 20 cores are required.
  - 400-Gb/s 79 physical cores
- Result: Massive packet drops @ ≥100 Gb/s
- Implications:
  - Switch to SRAM: $$$ and power
  - Need explicit programmer control to defeat cache eviction

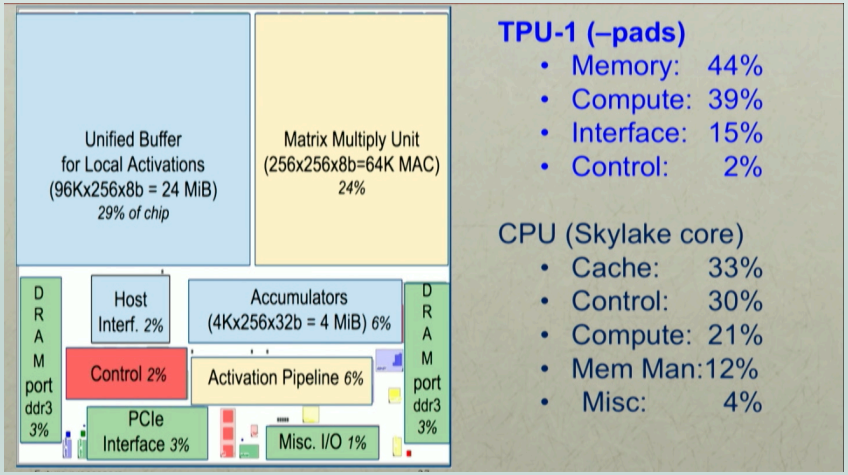


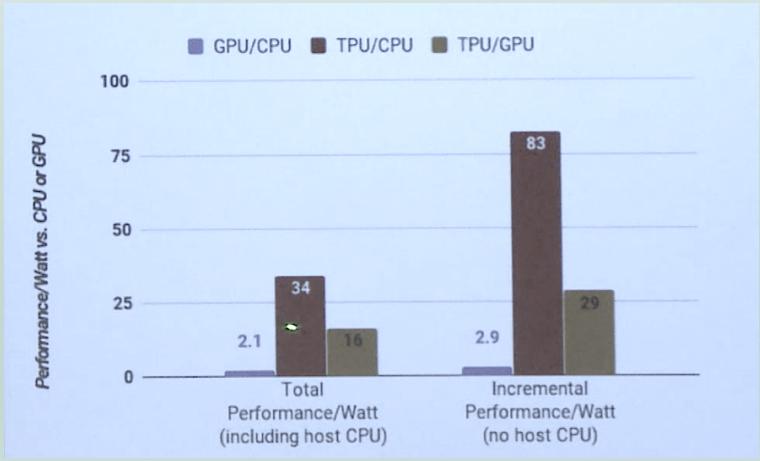
Interpacket Gap < DRAM Latency



Where to go from here? Domain-Specific Architectures

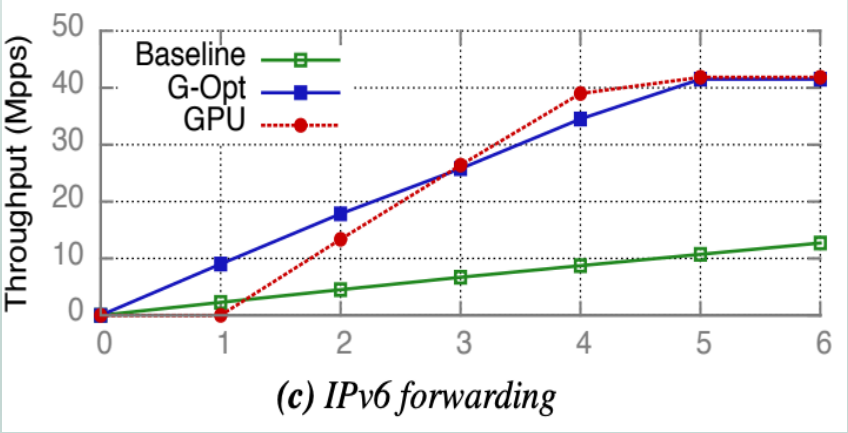
- Tailor Architecture to problem *domain* (n.b. - not a strict ASIC approach)
  - Already have: GPUs for graphics and virtual reality
  - Emerging: Neural Network processors (e.g. Google TPU)
  - Promising: Programmable switching silicon (e.g. P4 or something more powerful)

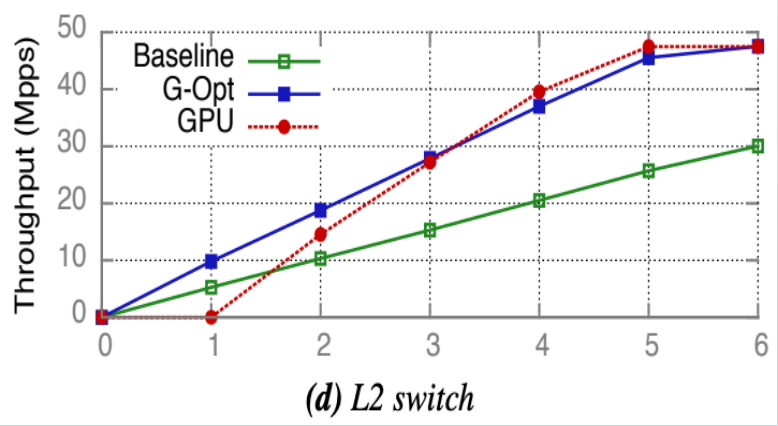




Can we apply this to Networking?

- GPUs for Networking? Initial Results not Encouraging:
  - Long setup times ⇒ Big batches ⇒ Increased forwarding latency
  - Need random memory access, but GPUs optimized for contiguous access





Some interesting outstanding questions

- Smart NICs have FPGAs – what's the best way to use them?
- Figure out how to use P4 on switches for general computing?
- How to bridge the gap in the programming model?
  - What is imperative/functional versus what is done data-flow
- What do the platforms look like?
  - Heterogenous elements closely coupled internally, with conventional network externally, or
  - Heterogenous elements with custom "internal" network built scale-out, with conventional network connecting the complexes, or
  - Some hybrid with multiple parallel interconnects
    - Note: Microsoft tried this with FPGA's to scale Bing search

That's it! Questions? Comments? Discussion?

Backup

Linux Kernel - Network Subsystem