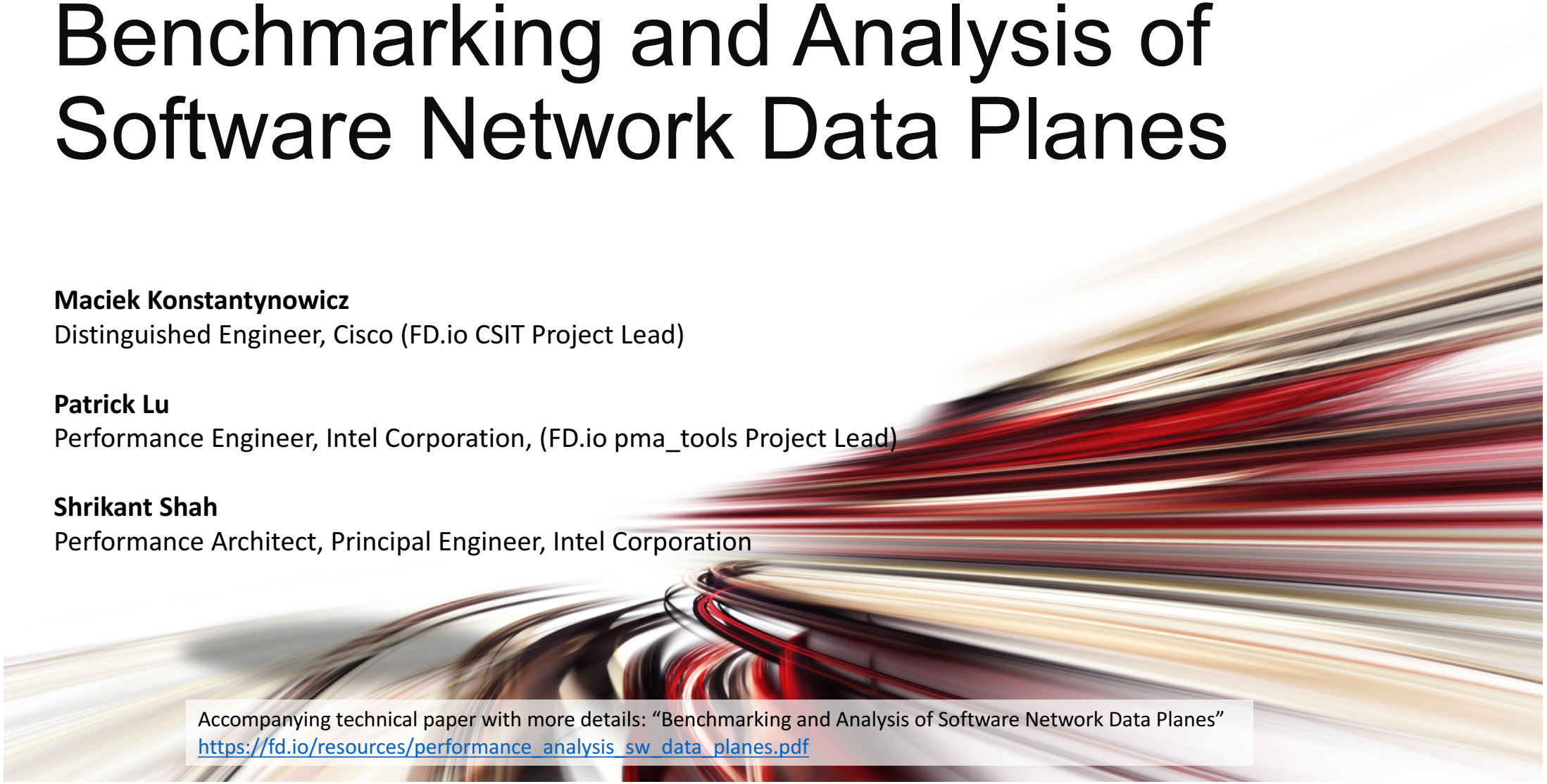
Benchmarking and Analysis of Software Network Data Planes

Maciek Konstantynowicz
Distinguished Engineer, Cisco (FD.io CSIT Project Lead)

Patrick Lu
Performance Engineer, Intel Corporation, (FD.io pma_tools Project Lead)

Shrikant Shah
Performance Architect, Principal Engineer, Intel Corporation



Accompanying technical paper with more details: “Benchmarking and Analysis of Software Network Data Planes”
https://fd.io/resources/performance_analysis_sw_data_planes.pdf

Disclaimers

Disclaimers from Intel Corporation:

- Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks
- This presentation includes a set of indicative performance data for selected Intel® processors and chipsets. However, the data should be regarded as reference material only and the reader is reminded of the important Disclaimers that appear in this presentation.
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Topics

- Background, Applicability
- Benchmarking Metrics
- Compute Performance Telemetry
- Performance Monitoring Tools
- Benchmarking Tests, Results and Analysis
- Efficiency Analysis with Intel TMAM
- Conclusions

Background

- Fast Software networking is important if not critical
  - Foundation for cloud-native services
  - VPNs, managed security, Network-as-a-Service
  - Network-intensive microservices

- But benchmarking and optimizing performance is not straightforward
  - Compute nodes' runtime behaviour not simple to predict
  - Especially hard are SW data planes due to high I/O and memory loads
  - Many factors play a role, easy to get lost ..

Background

- Need for a common benchmarking and analysis methodology
  - Measure performance, effficiency and results repeatibility
  - Enable Apples-to-Apples results comparison
  - Drive focused SW data plane optimizations

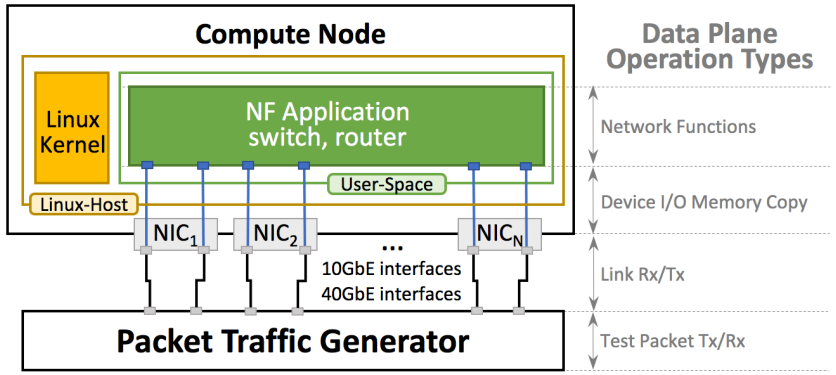
- Pre-requisite for fast and efficient cloud-native networking

Approach

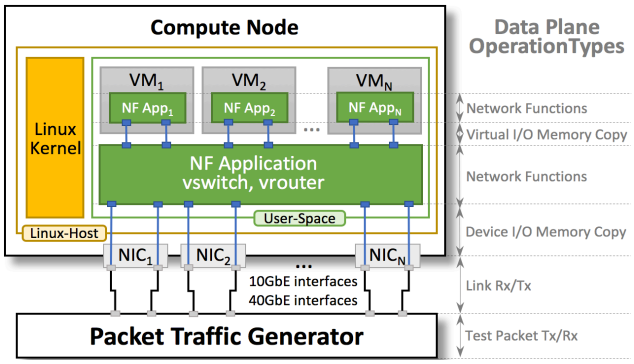
- Specify a benchmarking and analysis methodology
  - Define performance and efficiency metrics
  - Identify the main factors and interdependencies
  - Apply to network applications and designs
  - Produce sample performance profiles
- Use open-source tools
  - Extend them as needed
- Drive development of automated benchmarking and analytics
  - In open-source

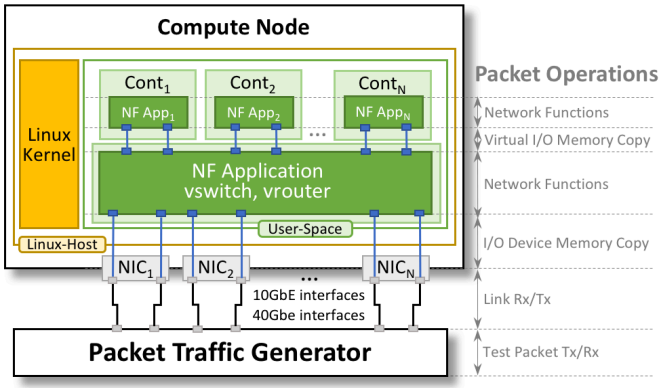
Applicability – Deployment Topologies

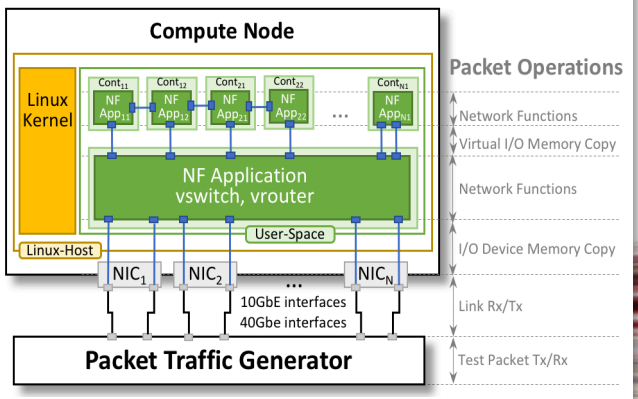
- **Start simple**
  - Benchmark the NIC to NIC packet path
  - Use the right test and telemetry tools and approaches
  - Analyze all key metrics: PPS, CPP, IPC, IPP and more
  - Find performance ceilings of Network Function data plane



- **Then apply the same methodology to other packet paths and services**







Benchmarking Metrics – defining them for packet processing..

Treat software network Data Plane as one would any program, ..

$$program\_unit\_execution\_time[sec] = \frac{\#instructions}{program\_unit} * \frac{\#cycles}{instruction} * cycle\_time$$

with the instructions per packet being the program unit, ..

$$packet\_processing\_time[sec] = \frac{\#instructions}{packet} * \frac{\#cycles}{instruction} * cycle\_time$$

and arrive to the main data plane benchmarking metrics.

$$\#cycles\_per\_packet = \frac{\#instructions}{packet} * \frac{\#cycles}{instruction}$$

CPP **IPP** **CPI or IPC**

PPS $$throughput[pps] = \frac{1}{packet\_processing\_time[sec]} = \frac{CPU\_freq[Hz]}{\#cycles\_per\_packet}$$

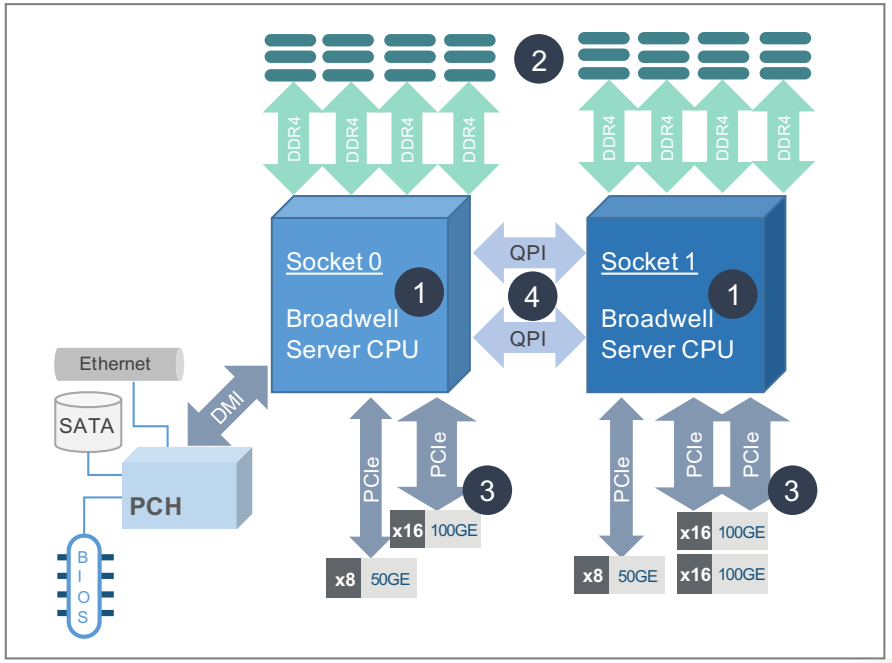
BPS $$throughput\ [bps] = throughput[pps] * packet\_size[pps]$$

Compute **CPP** from **PPS** or vice versa..

And mapping them to resources..

Main architecture resources used for packet-centric operations:

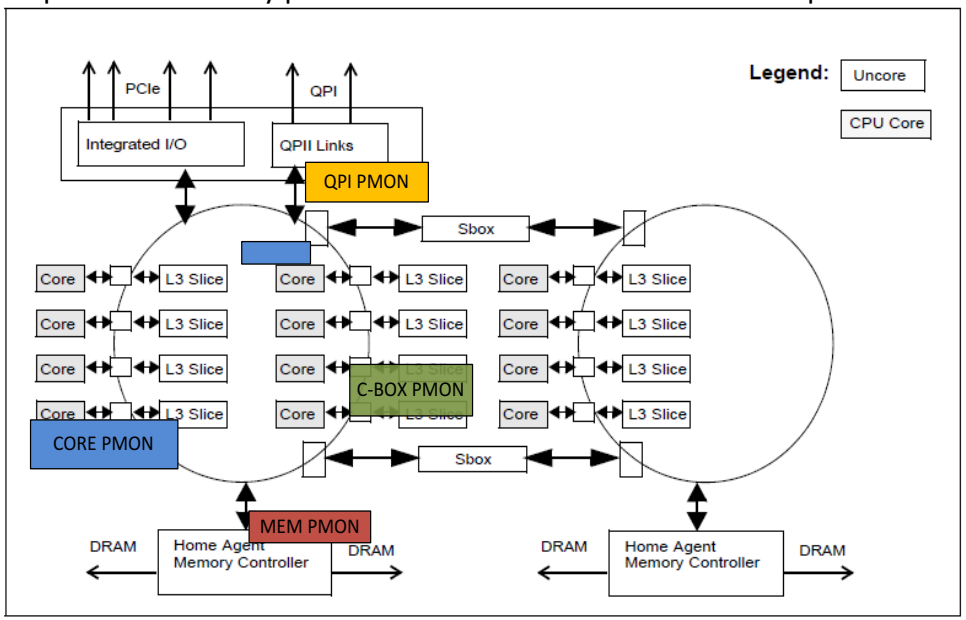
1. **Packet processing operations** – *How many CPU core cycles are required to process a packet*?
2. **Memory bandwidth** – *How many memory-read and -write accesses are made per packet?*
3. **I/O bandwidth** – *How many bytes are transferred over PCIe link per packet?*
4. **Inter-socket transactions** – *How many bytes are accessed from the other socket or other core in the same socket per packet?*



Compute Performance Telemetry

- Reliable performance telemetry points are critical to systematic analysis
- Example: Intel x86_64 processors on-chip PMU logic
  - Performance Monitoring Units (PMU)
  - Counting specific HW events as they occur in the system
  - Counted events can be then combined to calculate needed metrics e.g. IPC
- All defined benchmarking metrics can be measured in real time using PMU events

Important Telemetry points in Intel® Xeon® E5-2600 v4 series processors*



1. **CORE PMON** - Core specific events monitored through Core performance counters including number of instructions executed, cache misses, branch predictions.
2. **C-BOX PMON** - C-Box performance monitoring events used to track LLC access rates, LLC hit/miss rates, LLC eviction and fill rates.
3. **MEM PMON** – Memory controller monitoring counters.
4. **QPI PMON** - QPI counters monitor activities at QPI links, the interconnect link between the processors.

*Intel Optimization Manual" – Intel® 64 and IA-32 architectures optimization reference manual

Performance Monitoring Tools

- Number of tools and code libraries available for monitoring performance counters and their analysis e.g. Intel® VTune™, pmu-tools.

- Three open-source tools have been in presented Benchmarking and Analysis methodology

1. **Linux Perf**
   - a generic Linux kernel tool for basic performance events
   - enables performance hot-spot analysis at source code level
2. **PMU-Tools**
   - builds upon Linux Perf, provides richer analysis and enhanced monitoring events for a particular processor architecture
   - includes Top-down Micro-Architecture Method (TMAM), powerful performance analysis tool based on the, as presented in this talk
3. **PCM Tool Chain**
   - set of utilities focusing on specific parts of architecture including Core, Cache hierarchy, Memory, PCIe, NUMA
   - easy to use and great for showing high-level statistics in real-time

- Their applicability to each identified analysis area

Analysis Area	Performance Tools
CPU Cores	pmu-tools top-down analysis.
CPU cache Hierarchy	pmu-tools, pcm.x.
Memory bandwidth	pcm-memory.x, pmu-tools.
PCIe bandwidth	pcm-pcie.x
Inter-Socket transactions	pcm-numa.x, pmu-tools.

In-depth introduction on performance monitoring (@VPP OpenStack mini summit):
https://youtu.be/Pt7E1lXZO90?t=14m53s

Let’s move to the actual benchmarks..

Benchmarking Tests - Network Applications

Idx	Application Name	Application Type	Benchmarked Configuration
1	EEMBC CoreMark®	Compute benchmark	Runs computations in L1 core cache.
2	DPDK Testpmd	DPDK example	Baseline L2 packet looping, point-to-point.
3	DPDK L3Fwd	DPDK example	Baseline IPv4 forwarding, /8 entries, LPM
4	FD.io VPP	NF application	vSwitch with L2 port patch, point-to-point.
5	FD.io VPP	NF application	vSwitch MAC learning and switching.
6	OVS-DPDK	NF application	vSwitch with L2 port cross-connect, point-to-point.
7	FD.io VPP	NF application	vSwitch with IPv4 routing, /32 entries.

Chosen to compare pure compute performance against others with heavy I/O load.

Cover basic packet processing operations covering both I/O and compute aspects of the system.

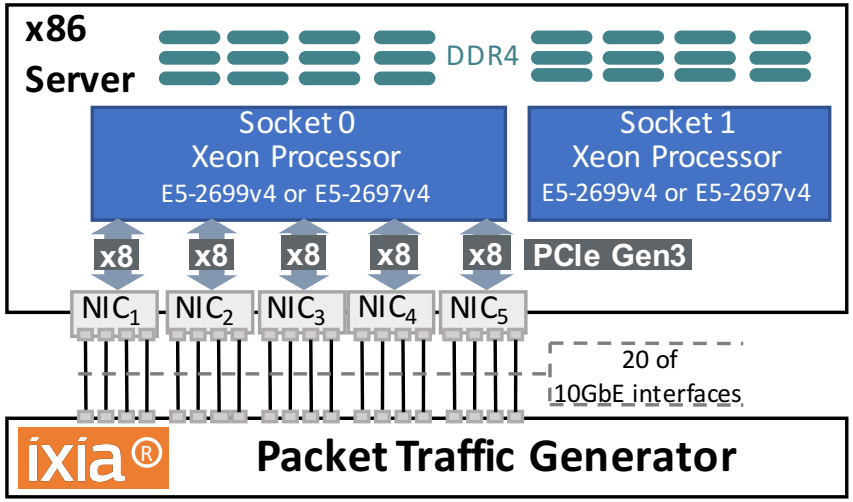
Cover performance of a virtual switch/router:

- Virtual switch/router apps are tested in L2 switching and IPv4 routing configurations.
- Cover both different implementations and L2/L3 packet switching scenarios.

Benchmarking Tests – Environment Setup

Hardware Setup

- Simple two-node test topology
  - i) Traffic Generator and ii) System Under Test
  - 64B packet size to stress cores and I/O
- Fully populated PCIe slots in Numa0/Socket0
  - 20x 10GE, x710 NICs
  - No tests with Socket1
- Two processor types tested to evaluate dependencies on key CPU parameters e.g. core frequency



Idx	Core Frequency	Core Density	Intel® Xeon® Processor Model
Server1	2.20 GHz (Low)	22C (High)	E5-2699v4 55MB 145W
Server2	3.20 GHz (High)	8C (Low)	E5-2667v4 25MB 135W

Application Setup

- Benchmarked applications tested in single- and multi-core configuration with and without Intel HyperThreading
- (#Cores, #10GbE) port combinations chosen to make the tests remain CPU core bound and not be limited by network I/O (e.g. Mpps NIC restrictions)

Number of 10 GbE ports used per test					
# of cores used / Benchmarked Workload	1 core	2 core	3 core	4 core	8 core
DPDK-Testpmd L2 Loop	8	16	20	20	20
DPDK-L3Fwd IPv4 Forwarding	4	8	12	16	16
VPP L2 Patch Cross-Connect	2	4	6	8	16
VPP L2 MAC Switching	2	4	6	8	16
OVS-DPDK L2 Cross-Connect	2	4	6	8	16
VPP IPv4 Routing	2	4	6	8	16

Benchmarking Tests – Single Core Results

Measured

- Throughput (Mpps)
- #Instructions/cycle

Calculated

- #instructions/packet
- #cycles/packet

High Level Observations

- Throughput varies with complexities in processing
- Performance mostly scales with frequency (see #cycles/packet in **Table A** and **Table B**)
- Hyper-threading boosts the packet throughput

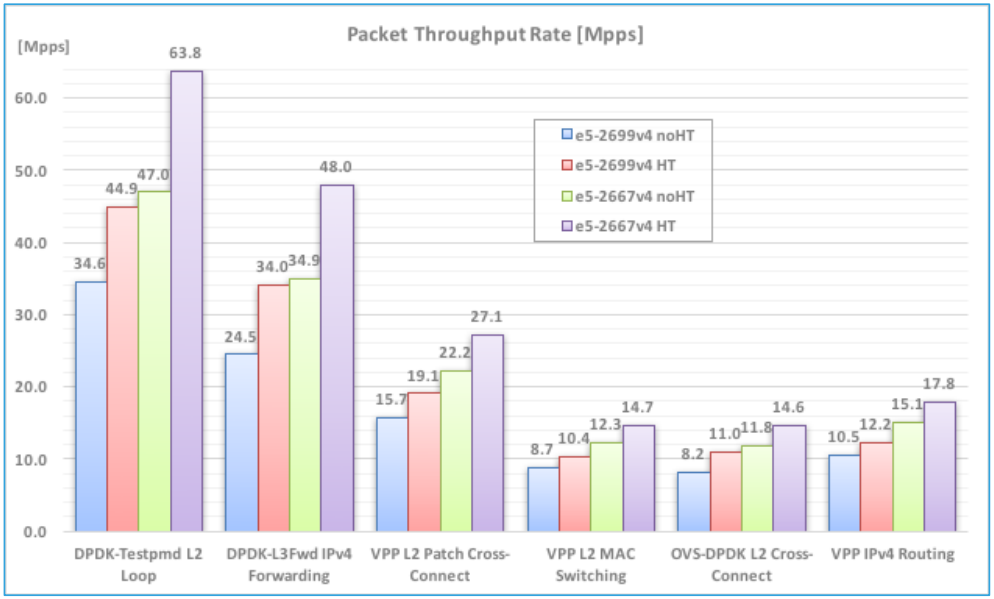
Table A: E5-2699v4, 2.2 GHz, 22 Cores

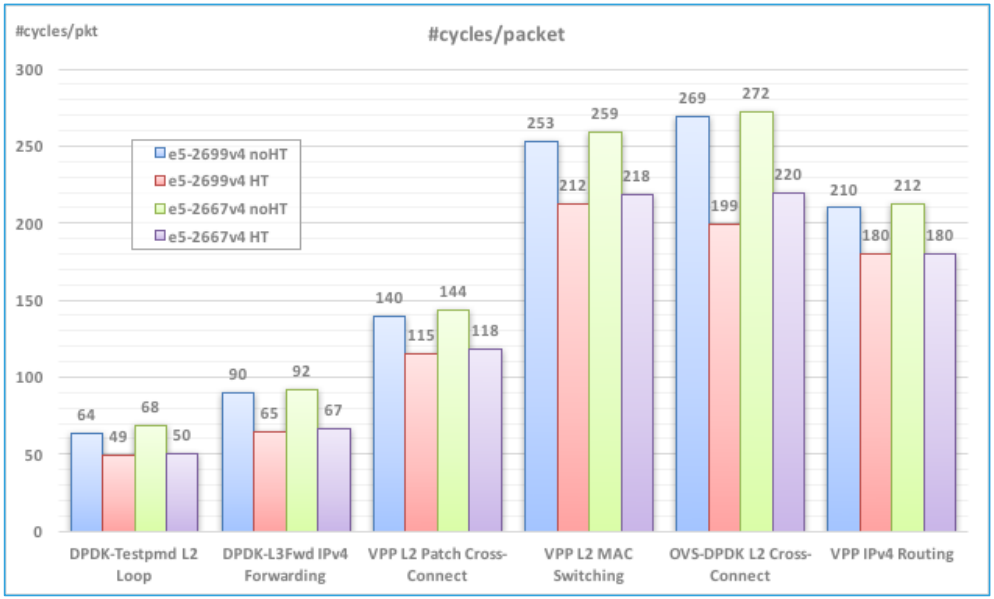
Benchmarked Workload	Throughput [Mpps]		#instructions /packet		#instructions /cycle		#cycles /packet	
Dedicated 1 physical core with =>	**noHT**	**HT**	**noHT**	**HT**	**noHT**	**HT**	**noHT**	**HT**
CoreMark [Relative to CMPS ref*]	1.00	1.23	n/a	n/a	2.4	3.1	n/a	n/a
DPDK-Testpmd L2 Loop	34.6	44.9	92	97	1.4	2.0	64	49
DPDK-L3Fwd IPv4 Forwarding	24.5	34.0	139	140	1.5	2.2	90	65
VPP L2 Patch Cross-Connect	15.7	19.1	272	273	1.9	2.4	140	115
VPP L2 MAC Switching	8.7	10.4	542	563	2.1	2.7	253	212
OVS-DPDK L2 Cross-connect	8.2	11.0	533	511	2.0	2.6	269	199
VPP IPv4 Routing	10.5	12.2	496	499	2.4	2.8	210	180
*CoreMarkPerSecond reference value	- score in the reference configuration: E5-2699v4, 1 Core noHT.							

Table B: E5-2667v4 3.2 GHz, 8 Cores

Benchmarked Workload	Throughput [Mpps]		#instructions /packet		#instructions /cycle		#cycles /packet	
Dedicated 1 physical core with =>	**noHT**	**HT**	**noHT**	**HT**	**noHT**	**HT**	**noHT**	**HT**
CoreMark [Relative to CMPS ref*]	1.45	1.79	n/a	n/a	2.4	3.1	n/a	n/a
DPDK-Testpmd L2 Loop	47.0	63.8	92	96	1.4	1.9	68	50
DPDK-L3Fwd IPv4 Forwarding	34.9	48.0	139	139	1.5	2.1	92	67
VPP L2 Patch Cross-Connect	22.2	27.1	273	274	1.9	2.3	144	118
VPP L2 MAC Switching	12.3	14.7	542	563	2.1	2.6	259	218
OVS-DPDK L2 Cross-Connect	11.8	14.6	531	490	2.0	2.2	272	220
VPP IPv4 Routing	15.1	17.8	494	497	2.3	2.8	212	180
*CoreMarkPerSecond reference value	- score in the reference configuration: E5-2699v4, 1 Core noHT.							

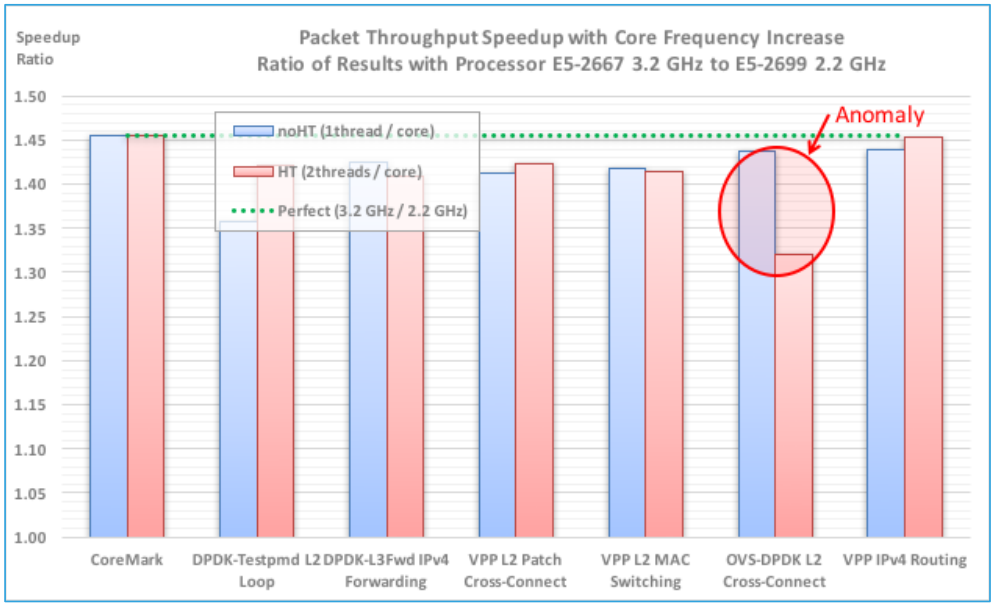
Results and Analysis – #cycles/packet (CPP) and Throughput (Mpps)

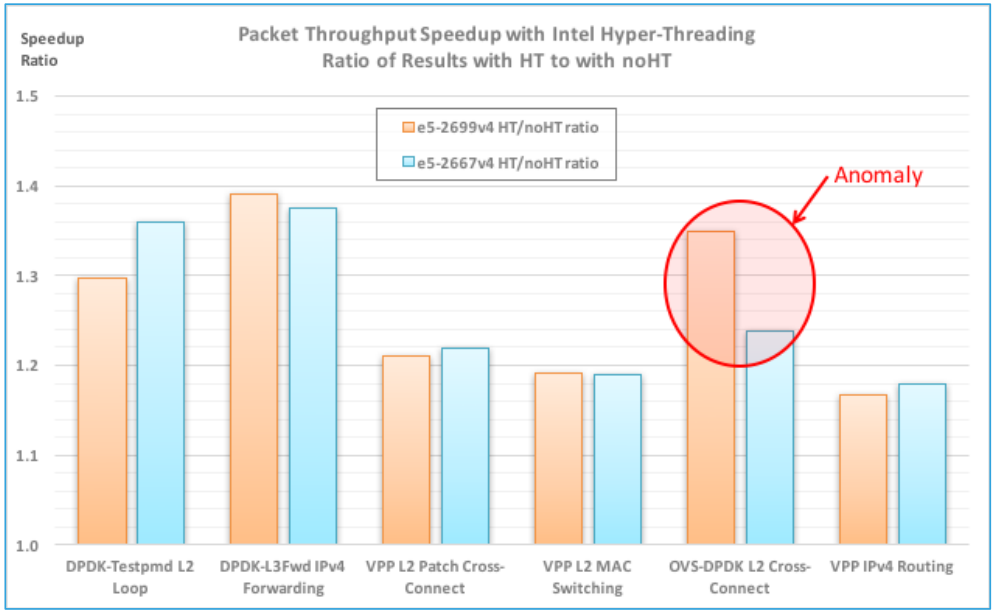




#cycles/packet = cpu_freq[MHz] / throughput[Mpps]

Results and Analysis – Speedup

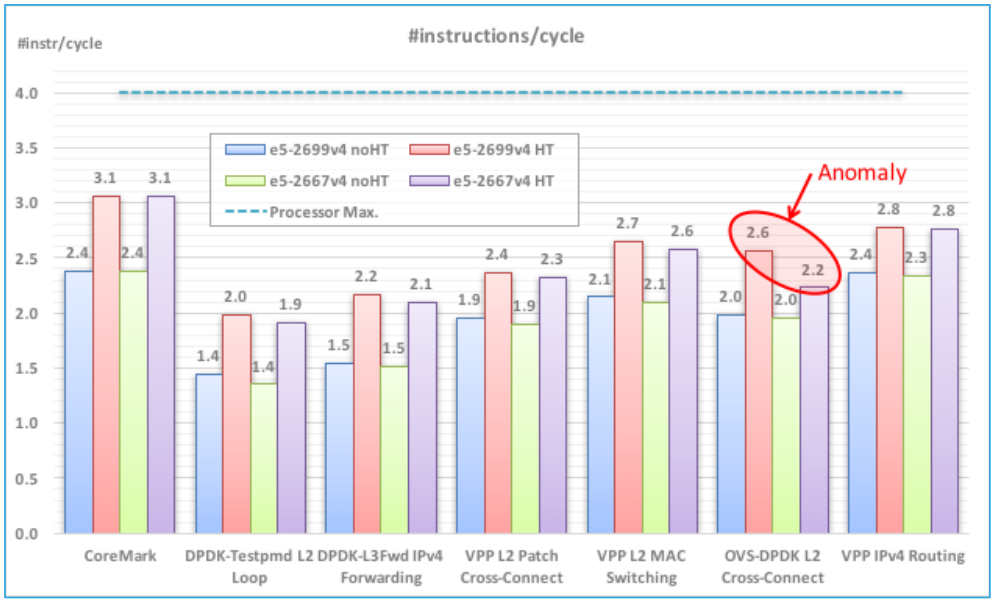




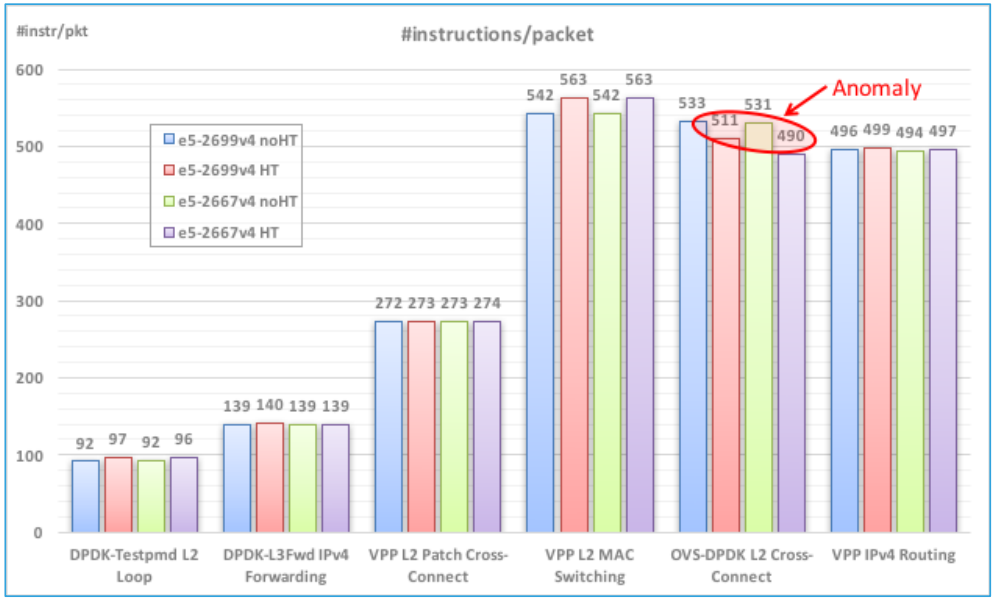
- Performance scales with Core Frequency in most cases
- Hyper-Threading brings performance improvement
  - Shows bigger advantage in the cases where cores are waiting for data

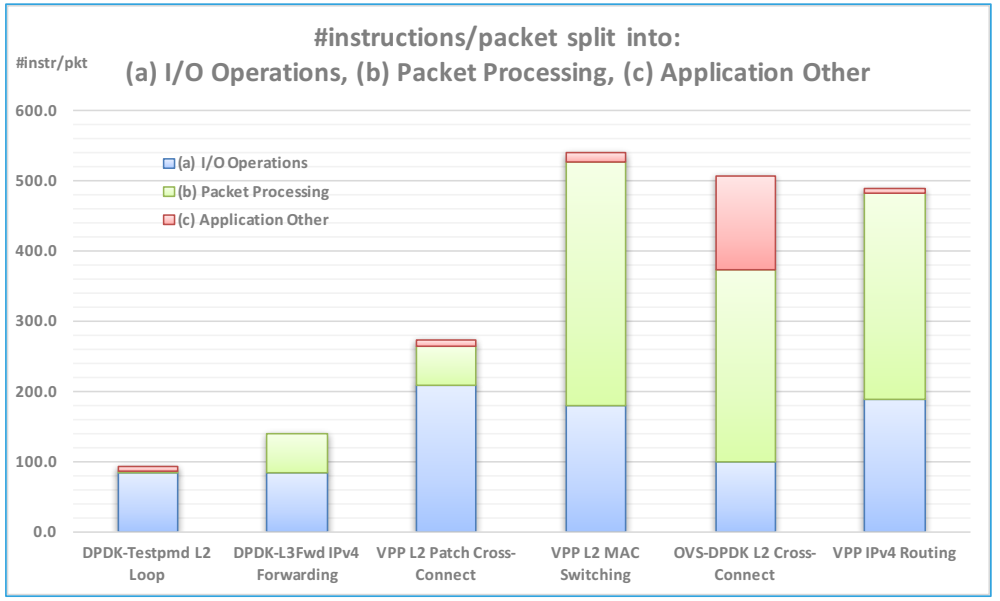
Results and Analysis - #instructions/cycle (IPC)

- IPC can be measured through pcm.x
- VPP has High IPC - hides latencies of data accesses
- Advantage of Hyper-Threading is reflected in increase in IPC
- High IPC does not necessarily mean there is no room of improvement in performance
  - Better algorithm, use of SSEx, AVX can boost performance further



Results and Analysis - #instr/packet (IPP)

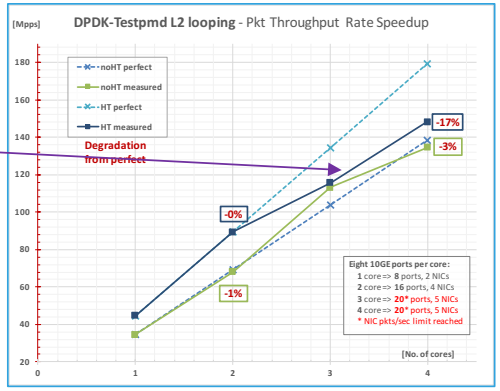


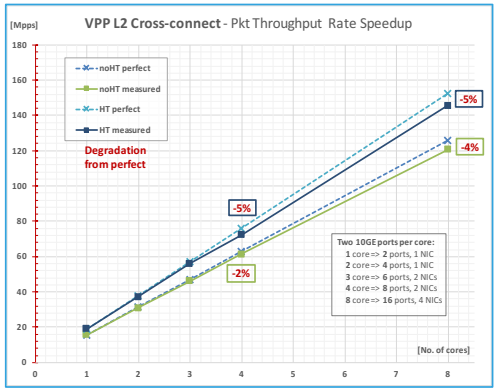


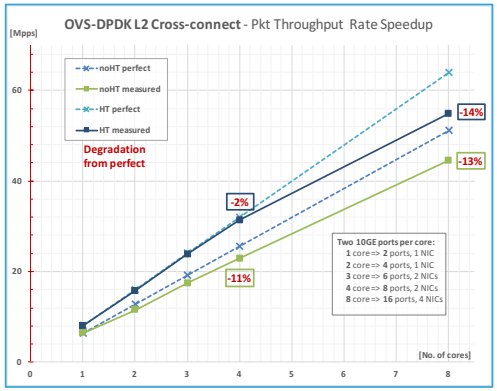
- IPP is calculated from IPC and CPP
- Distribution of instructions per operation type is derived from “processor trace” tool. It is an approximation..

Results and Analysis – Speedup with processing cores

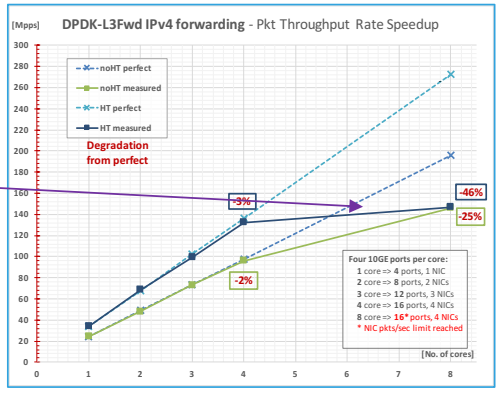
Limited by NICs

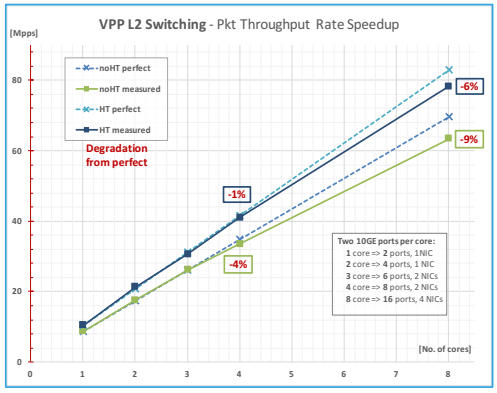


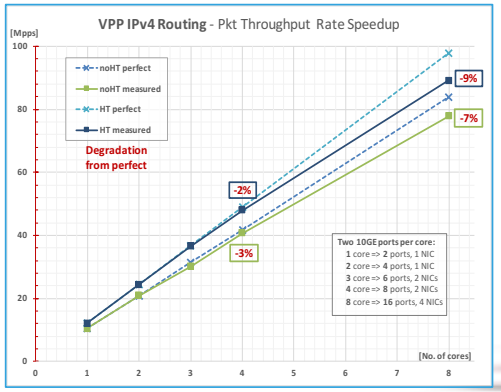




Limited by NICs







For most benchmarks, the performance scales almost linearly with number of cores

Results and Analysis – Memory and I/O Bandwidth

(E5-2699v4, 2.2 GHz, 22 Cores)

Benchmarked Workload	Throughput [Mpps]	Memory Bandwdith [MB/s]
Dedicated 4 physical cores with HyperThreading enabled, 2 threads per physical core, 8 threads in total => 4c8t		
DPDK-Testpmd L2 looping	148.3	18
DPDK-L3Fwd IPv4 forwarding	132.2	44
VPP L2 Cross-connect	72.3	23
VPP L2 Switching	41.0	23
OVS-DPDK L2 Cross-connect	31.2	40
VPP IPv4 Routing	48.0	1484

Benchmarked Workload	Throughput [Mpps]	PCIe Write Bandwidth [MB/s]	PCIe Read Bandwidth [MB/s]	PCIe Write #Transactions/ Packet	PCIe Read #Transactions/ Packet
Dedicated 4 physical cores with HyperThreading enabled, 2 threads per physical core, 8 threads in total => 4c8t					
DPDK-Testpmd L2 looping	148.3	13,397	14,592	1.41	1.54
DPDK-L3Fwd IPv4 forwarding	132.2	11,844	12,798	1.40	1.51
VPP L2 Cross-connect	72.3	6,674	6,971	1.44	1.51
VPP L2 Switching	41.0	4,329	3,952	1.65	1.51
OVS-DPDK L2 Cross-connect	31.2	3,651	3,559	1.83	1.78
VPP IPv4 Routing	48.0	4,805	4,619	1.56	1.50

- Minimal Memory bandwidth utilization due to efficient DPDK and VPP implementation
- Memory and PCIe bandwidth can be measured with PCM tools

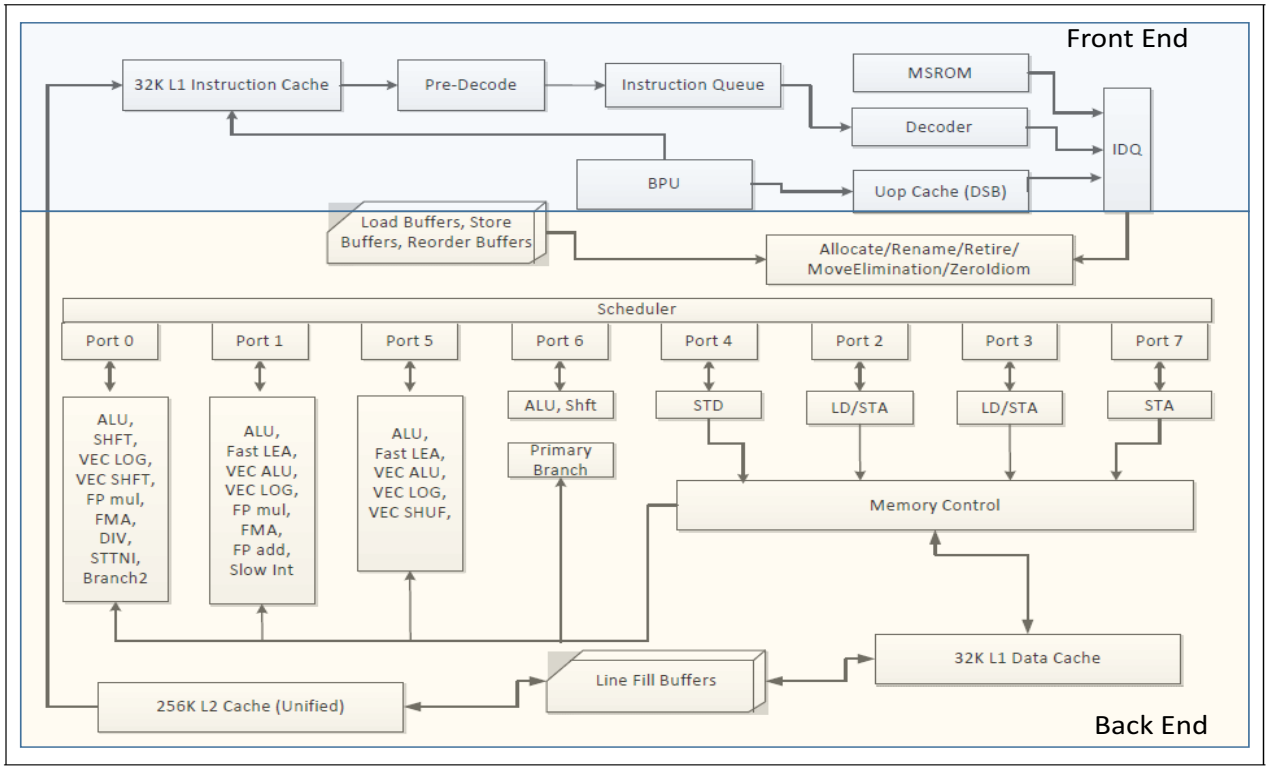
Compute Efficiency Analysis with Intel TMAM*

Frontend

- Responsible for fetching the instructions, decoding them into one or more low-level hardware(µOps), dispatching to Backend.

Backend

- Responsible for scheduling µOps, µOps execution, and their retiring per original program's order

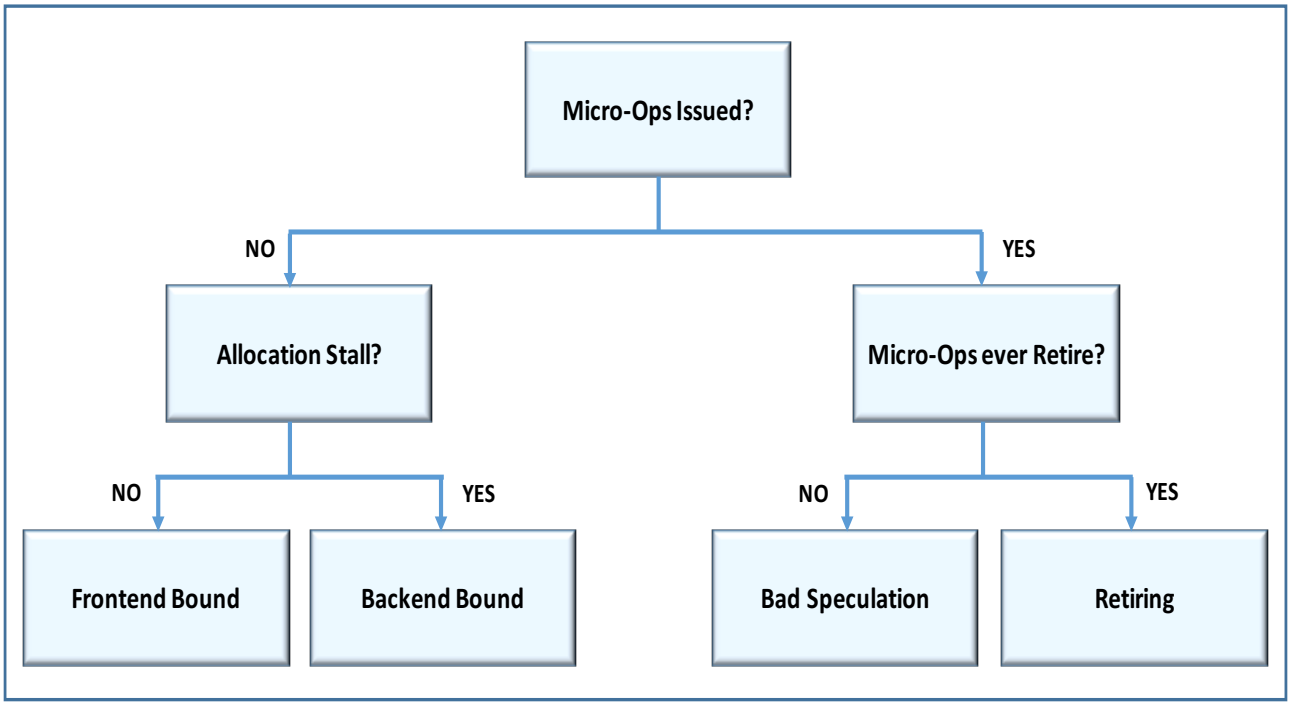


Intel® Xeon® E5-2600 v4 series Processor core architecture#

* TMAM : Top-down Microarchitecture Analysis Method. See "Intel Optimization Manual" – Intel® 64 and IA-32 architectures optimization reference manual

Compute Efficiency Analysis with Intel TMAM

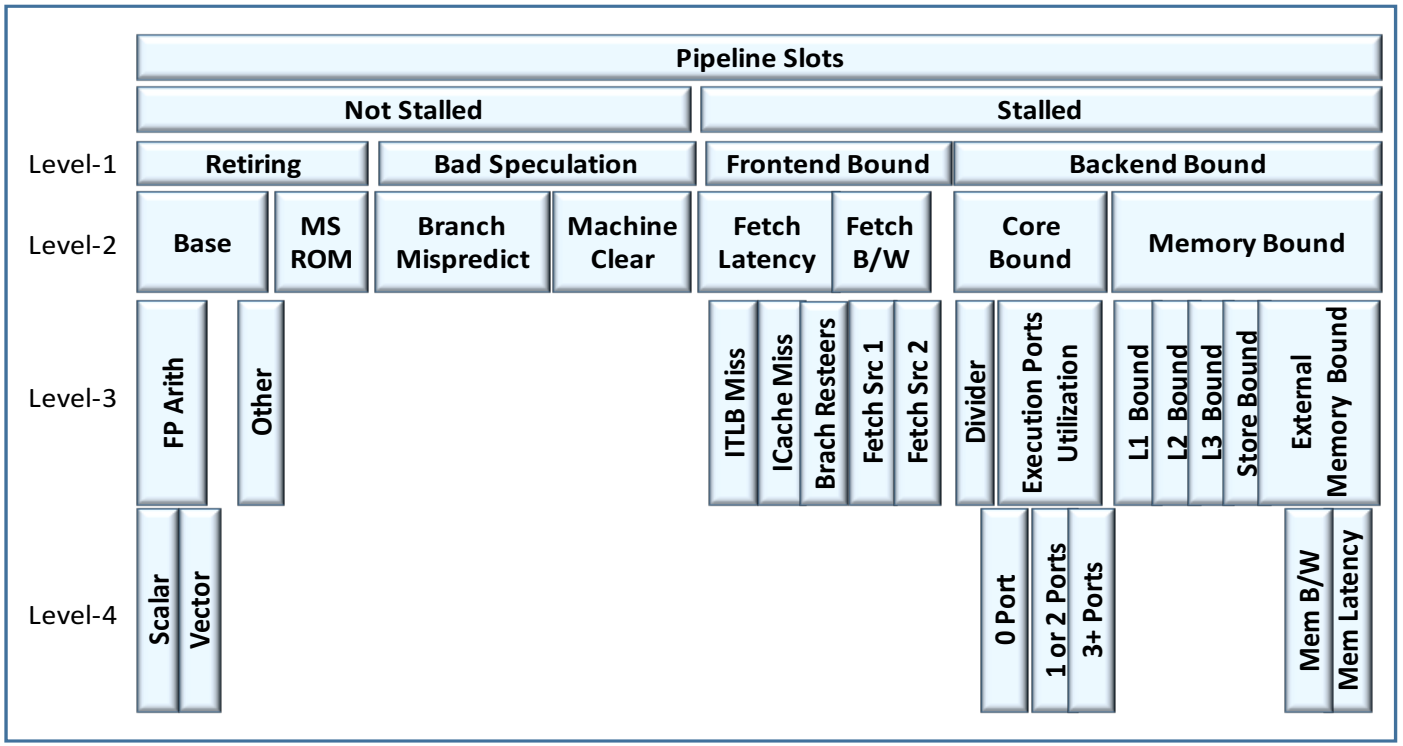
TMAM Level 1



- Top-down Microarchitecture Analysis Method (TMAM) is a systematic and well-proven approach for compute performance analysis. Equally applicable for network applications
- Helps identify sources of performance bottlenecks by top-down approach

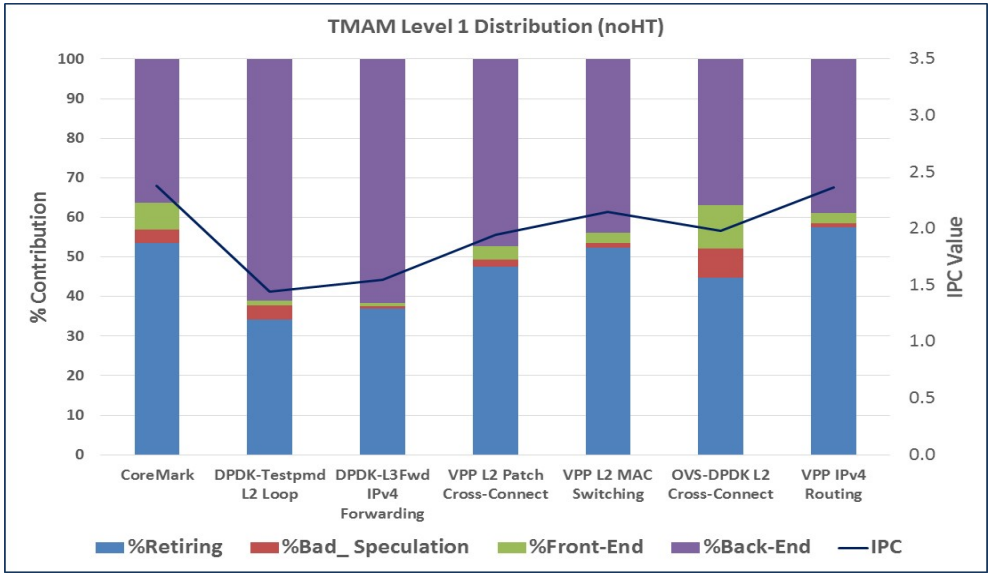
Hierarchical TMAM Analysis

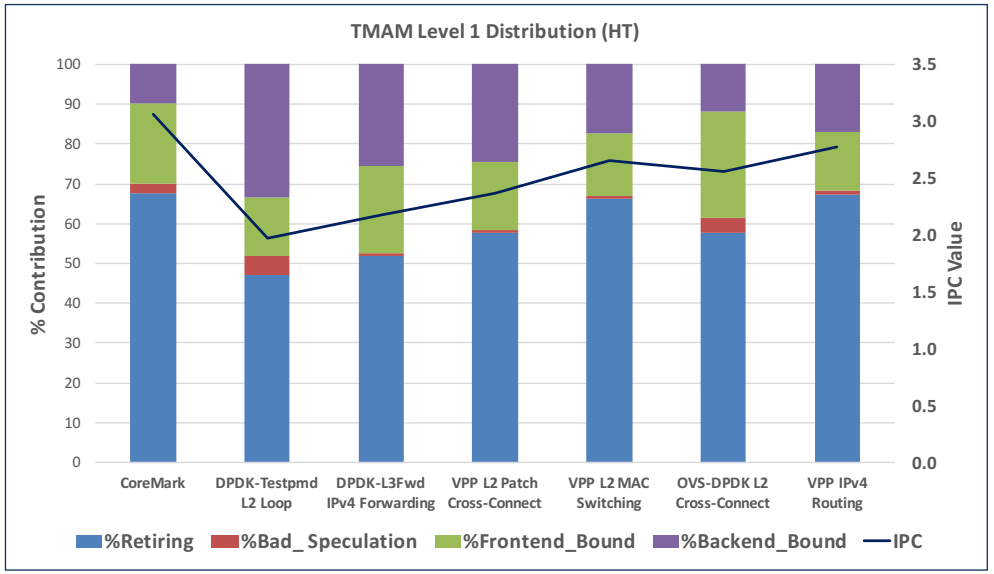
TMAM Level 1-4



- Hierarchical approach helps quick and focused analysis
- TMAM analysis can be carried out with PMU_TOOLS
- Once area of a bottleneck is identified, it can be debugged using linux *"perf top"* or *VTune*™

Compute Efficiency Analysis with Intel TMAM





Observations:

IPC	Good IPC for all Network workloads due to code optimization, HT makes it even better.
Retiring	Instructions retired, drives IPC.
Bad_Speculations	Minimal Bad branch speculations, Attributed to architecture logic, and software pragmas.
Backend Stalls	Major contributor causing low IPC in noHT cases, HT hides backend stalls.
Frontend Stalls	Becomes a factor in HT as more instructions are being executed by both logical cores.

Conclusions

- Analyzing and optimizing performance of network applications is a ongoing challenge
- Proposed benchmarking and analysis methodology* aims to address this challenge
  - Clearly defined performance metrics
  - Identified main efficiency factors and interdependencies
  - Generally available performance monitoring tools
  - Clearly defined comparison criteria – “Apples-to-Apples”
- Presented methodology illustrated with sample benchmarks
  - DPDK example applications, OVS-DPDK and FD.io VPP
- More work needed
  - Benchmarking automation tools
  - Packet testers integration
  - Automated telemetry data analytics

Progressing towards establishing a de facto reference best practice.

* More detail available in accompanying technical paper “Benchmarking and Analysis of Software Network Data Planes”
https://fd.io/resources/performance_analysis_sw_data_planes.pdf

References

Benchmarking Methodology

- “Benchmarking and Analysis of Software Network Data Planes” by M. Konstantynowicz, P. Lu, S. M.Shah, https://fd.io/resources/performance_analysis_sw_data_planes.pdf

Benchmarks

- EEMBC CoreMark® - http://www.eembc.org/index.php.
- DPDK testpmd - http://dpdk.org/doc/guides/testpmd_app_ug/index.html.
- FDio VPP – Fast Data IO packet processing platform, docs: https://wiki.fd.io/view/VPP, code: https://git.fd.io/vpp/.
- OVS-DPDK - https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview.

Related FD.io projects

- CSIT - Continuous System Integration and Testing: https://wiki.fd.io/view/CSIT
- pma_tools

Performance Analysis Tools

- “Intel Optimization Manual” – Intel® 64 and IA-32 architectures optimization reference manual
- Linux PMU-tools, https://github.com/andikleen/pmu-tools

TMAM

- Intel Developer Zone, Tuning Applications Using a Top-down Microarchitecture Analysis Method, https://software.intel.com/en-us/top-down-microarchitecture-analysis-method-win.
- Technion presentation on TMAM , Software Optimizations Become Simple with Top-Down Analysis Methodology (TMAM) on Intel® Microarchitecture Code Name Skylake, Ahmad Yasin. Intel Developer Forum, IDF 2015. [Recording]
- A Top-Down Method for Performance Analysis and Counters Architecture, Ahmad Yasin. In IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, https://sites.google.com/site/analysismethods/yasin-pubs.

Test Configuration

Processor	Intel® Xeon® E5-2699v4 (22C, 55M, 2.2 GHz)	Intel® Xeon® E5-2667v4 (8C,25 MB, 3.2 GHz)
Vendor	Intel	Intel
Nodes	1	1
Sockets	2	2
Cores Per Processor	22	8
Logical cores Per Processor	44	16
Platform	Supermicro® X10DRX	Supermicro® X10DRX
Memory DIMMs Slots used/Processor	4 channels per Socket	4 channels per Socket
Total Memory	128 GB	128 GB
Memory DIMM Configuration	16 GB / 2400 MT/s / DDR4 RDIMM	16 GB / 2400 MT/s / DDR4 RDIMM
Memory Comments	1 DIMM/Channel, 4 Ch per scoket	1 DIMM/Channel, 4 Ch per scoket
Network Interface Cards	X710-DA4 quad 10 Gbe Port cards, 5 cards total	X710-DA4 quad 10 Gbe Port cards, 5 cards total
OS	Ubuntu* 16.04.1 LTS x86_64	Ubuntu* 16.04.1 LTS x86_64
OS/Kernel Comments	4.4.0-21-generic	4.4.0-21-generic
Primary / Secondary Software	DPDK 16.11, VPP version v17.04-rc0~143-gf69ecfe, Qemu version 2.6.2, OVS-DPDK version:2.6.90, Fortville Firmware:FW 5.0 API 1.5 NVM 05.00.04 eetrack 800024ca	DPDK 16.11, VPP version v17.04-rc0~143-gf69ecfe, Qemu version 2.6.2, OVS-DPDK version:2.6.90, Fortville Firmware:FW 5.0 API 1.5 NVM 05.00.04 eetrack 800024ca
Other Configurations	Energy Performance BIAS Setting: Performance, Power Technology, Energy Performance Tuning, EIST,Turbo,C1E,COD, Early Snoop, DRAM RAPL Baseline, ASPM, VT-d: Disabled	Energy Performance BIAS Setting: Performance, Power Technology, Energy Performance Tuning, EIST,Turbo,C1E,COD, Early Snoop, DRAM RAPL Baseline, ASPM, VT-d: Disabled

THANK YOU !