Linux On Hyper-V – A Status Report

Dr. K. Y. Srinivasan

Enterprise Open Source Group
Microsoft Corporation
Redmond WA

kys@microsoft.com

Agenda

- Linux On Hyper-V – Our Linux Journey
- Technical Preview – Hyper-V Architecture And Drivers
- Performance And Scalability
  - Micro Benchmarks
  - HPC benchmarks
  - Middleware Benchmarks

Linux On Hyper-V – Six Years Ago

- Hyper-V specific Linux code in the staging area of the kernel tree.
- None of the Distros were supporting Hyper-V as a target platform
- 90%+ of Hyper-V specific development done by MSFT engineers
- Linux on Hyper-V in a “catch up” mode with respect to Windows on Hyper-V.

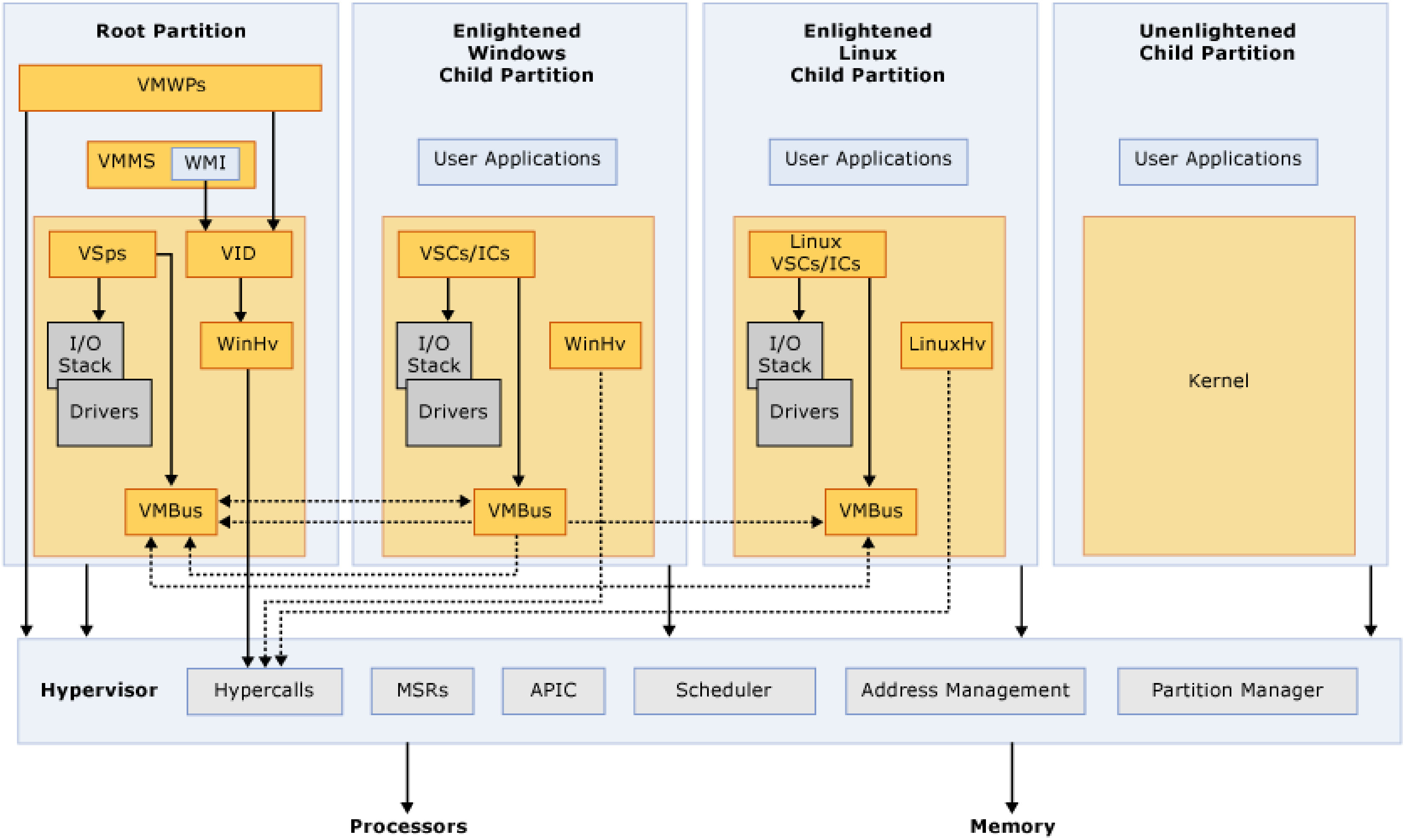
Linux On Hyper-V
– Over The Last Five Years

- Hyper-V Linux drivers begin to exit the staging tree (2011)
- Hyper-V share steadily increasing
- Distros begin to ship Hyper-V support
- Steady increase in community interest
- Linux support catches up with Windows support
- Continued improvement in performance

Linux On Hyper-V - Today

- Fully integrated upstream
- Shipped and supported by all major Distros
- Near native performance and scalability on many benchmarks
- Linux development done concurrently with Windows development
  - Linux truly a “first class” environment on Windows platforms
- More than 25% of IaaS VMs on Azure are Linux
- Upstream Hyper-V development is a community effort:
  - Close to 50% of patches submitted this year upstream are from the community

Hyper-V High Level Architecture



Hyper-V Architecture

- Full Virtualization with selective enlightenments:
  - Enlightened I/O Paths
  - Other low-level enlightenments
    - Time keeping
    - Context switching
    - TLB shoot-down etc.

Linux On Hyper-V

- Currently Linux hosted as a Fully virtualized guest with I/O enlightenments:
  - Standard kernel binaries supported
  - I/O enlightenments packaged as driver modules
- Linux pvops framework can be used to leverage additional Hyper-V specific enlightenments

VMbus

- Supports efficient bi-directional communication between the host and the guest.
- Implements the channel abstraction:
  - A pair of ring buffers with the associated signaling machinery.
- Host offers managed as Linux devices

Storvsc – PV Front-end Storage Driver

- Based on SCSI protocol (host/guest protocol)
- Handles all block devices:
  - IDE
  - SCSI
- Supports Fibre Channel devices
- Supports hot add/remove of LUNs
- Supports dynamic resizing of LUNs

Netvsc – PV Front-end network Driver

- Based on remote NDIS protocol (host/guest protocol):
  - Linux skbuf decorated with remote NDIS headers
- Supports various offloads:
  - Segmentation
  - Checksum
- Virtual Receive Side Scaling (VRSS)

Util – Enhanced Manageability

- Heartbeat
    - Health monitoring
- Timesynch
- Key Value Pair (KVP)
- Shutdown
- Host initiated backup

Dynamic Memory – Enhanced Manageability

- Memory hot-add used to increase the assigned memory
- Ballooning used to modulate assigned memory
- Demand driven policy engine on the host
  - Guests post their memory demand to the host on a regular basis and this drives the policy engine.

Miscellaneous Drivers

- Mouse driver:
  - HID compliant driver
- Synthetic Keyboard driver
- Frame buffer driver
- PCI pass through driver:
  - Any PCIE device can be passed through to the guest

SR-IOV

- Synthetic path and VF path surfaced as independent links in the guest
- Full support for multi-tenant deployment:
    - All exception packets delivered on the synthetic path – GFT support
    - Exception packets and MC/BC packets reinjected into the VF path in the guest
    - Interfaces bonded to support scenarios where the VF needs to be disabled
        - Migration
        - Host update etc.
- Both software mediated as well as hardware mailbox communication supported
- Intel and Mellanox NICS currently supported; other vendors coming on board

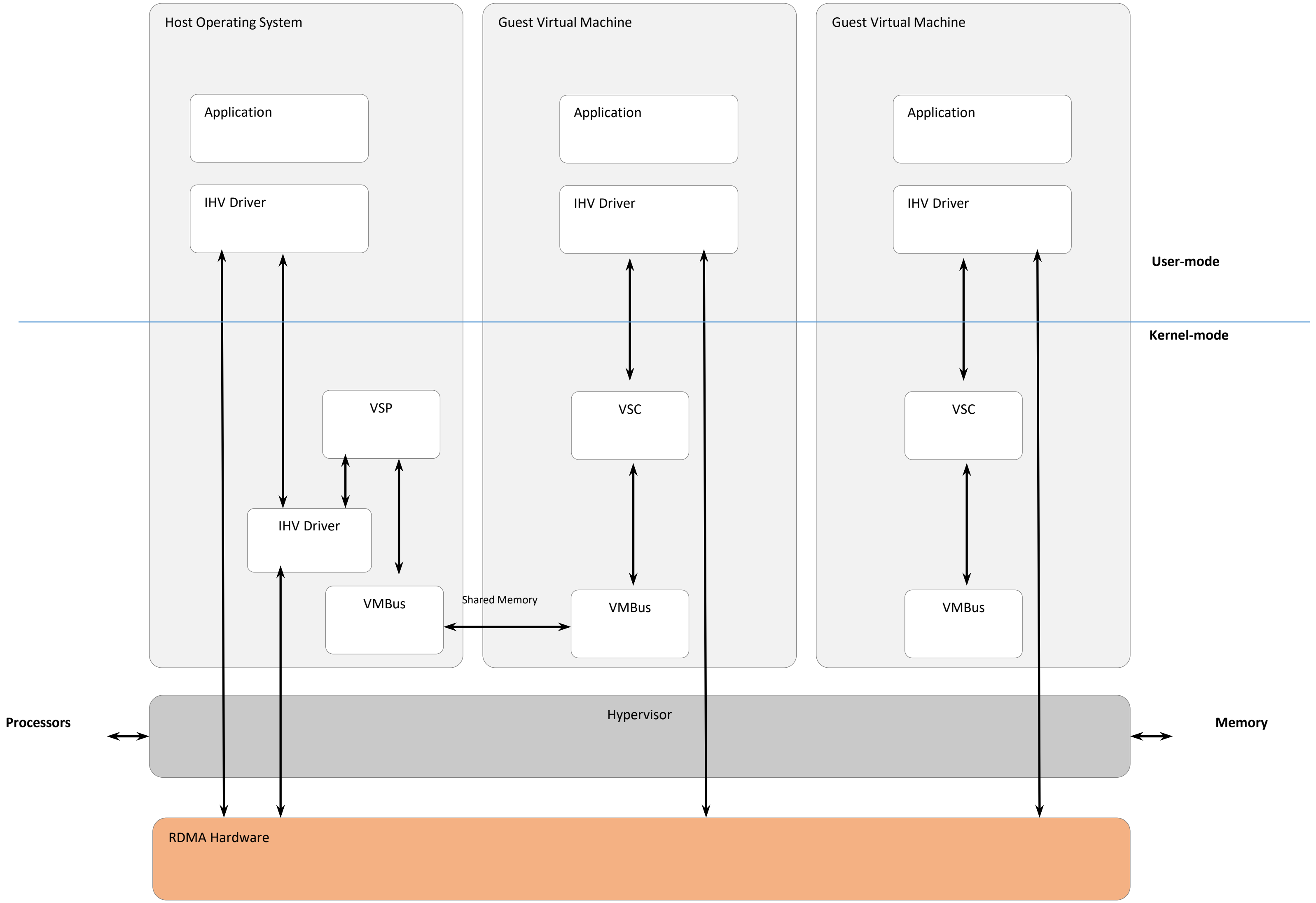
Hyper-V Sockets

- Supports sockets based communication between the guest and the host
- Defines a new AF – address family
- Uses Vmbus as the transport
- Needs no network connectivity between the guest and the host

Linux Guest RDMA

What is Endure

- **E**nlightened **N**etwork**D**irect on Az**ure**
- Provide native NetworkDirect performance
- Maintain control over device
  - No SR-IOV
- Maintain control over policy
  - Connection establishment for traffic isolation


Host Operating System
Guest Virtual Machine
Guest Virtual Machine
Application
Application
Application
IHV Driver
IHV Driver
IHV Driver
User-mode
Kernel-mode
VSP
VSC
VSC
IHV Driver
VMBus
Shared Memory
VMBus
VMBus
Processors
Hypervisor
Memory
RDMA Hardware

Endure Implementation On Windows

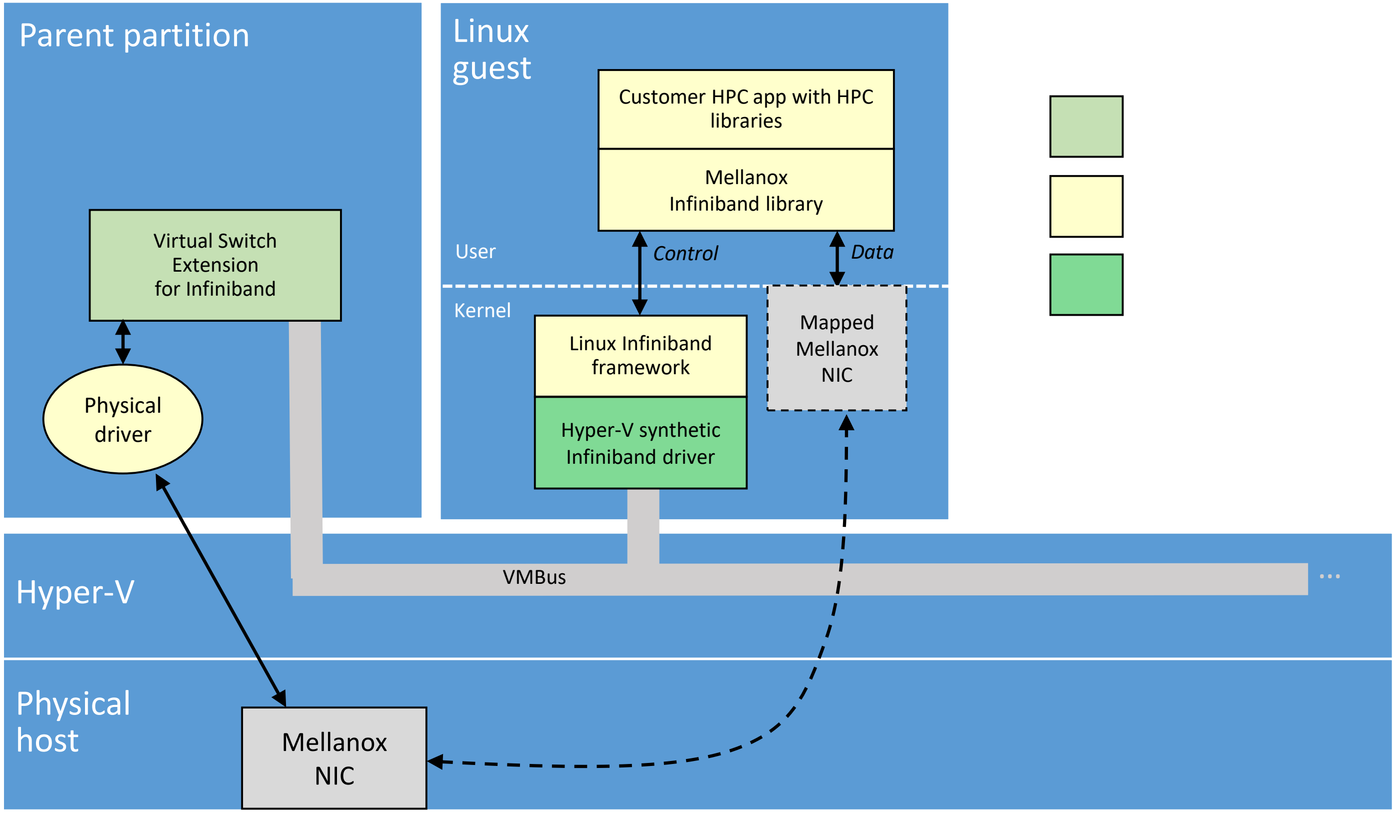
- Host Side:
  - VSP implements the resource partitioning on a per-VM basis
  - VSP does the marshalling of messages from the guest
  - VSP does not interpret the IHV specific part of the payload
  - VSP passes the IHV specific payload to the IHV driver

- Guest Side:
  - VSC merely forwards guest user level NetworkDirect calls over VMBUS with appropriate encapsulation

Endure For Linux Guests

- Key requirements:
  - No modifications on the host side.
  - No modifications in the Linux user space
- Key Challenges:
  - Bridging the semantic gap between NetworkDirect and ibverbs

RDMA/Infiniband for Azure


Parent partition
Linux guest
Customer HPC app with HPC libraries
Mellanox Infiniband library
User
Control
Data
Kernel
Virtual Switch Extension for Infiniband
Physical driver
Linux Infiniband framework
Hyper-V synthetic Infiniband driver
Mapped Mellanox NIC
Hyper-V
VMBus
…
Physical host
Mellanox NIC

IB Verbs not implemented

- create_ah()
- destroy_ah()
- attach_mcast()
- detach_mcast()
- process_mad()
- query_pkey()
- alloc_mw()
- bind_mw()
- dealloc_mw()
- alloc_fast_reg_mr()
- alloc_fast_reg_page_list()
- free_fast_reg_page_list()
- attach_mcast()
- detach_mcast()
- process_mad()
- post_send()
- post_recv()

Challenges Of Implementing Endure For Linux

- The guest RDMA device needs to masquerade as the physical RDMA device on the host:
    - Currently, we just masquerade as an mlx4 device
        - Additional sysfs files added to vmbus to publish PCI vendor and device IDs
- Merging the semantic gap between the Linux and Windows RDMA programming models:
    - Mapping the notion of IB ucontext to Window's abstraction
    - All transactions against the host in Endure needs the ucontext – create_listen() assumes an implicit ucontext.
- Keeping the Endure state machine in synch with the Linux kernel state machine
- CQ_NOTIFY is quite expensive – polling mode is preferred.

Linux Endure Driver

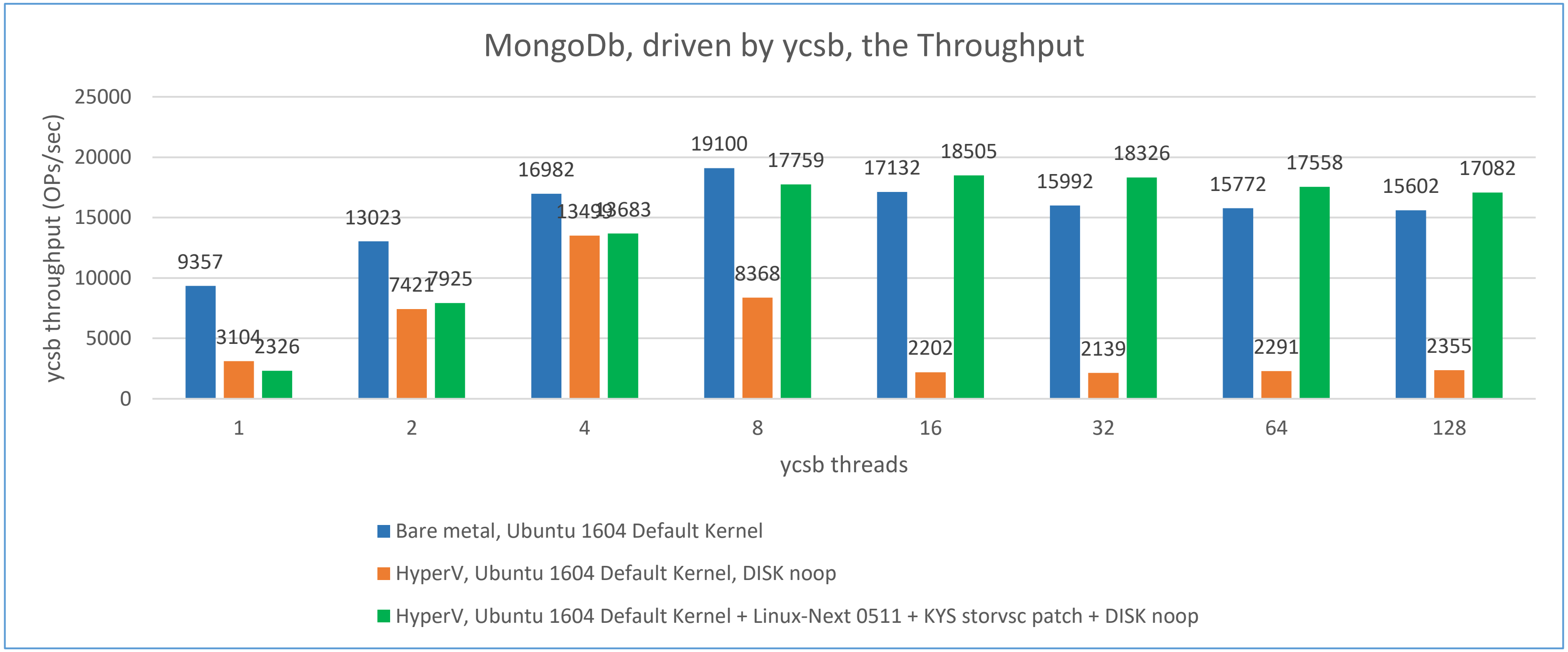
- Currently only supports Mellanox (mlx4 driver)
- Packaged as an RPM:
  - Support for RHEL 6/ 7 S
- SLES 12 HPC images available on Azure (with the Endure driver)
- Support for other Distros coming soon
- Not up-streamed yet
- Supports both RoCE as well as infiniband backend

Linux On Hyper-V Performance Data

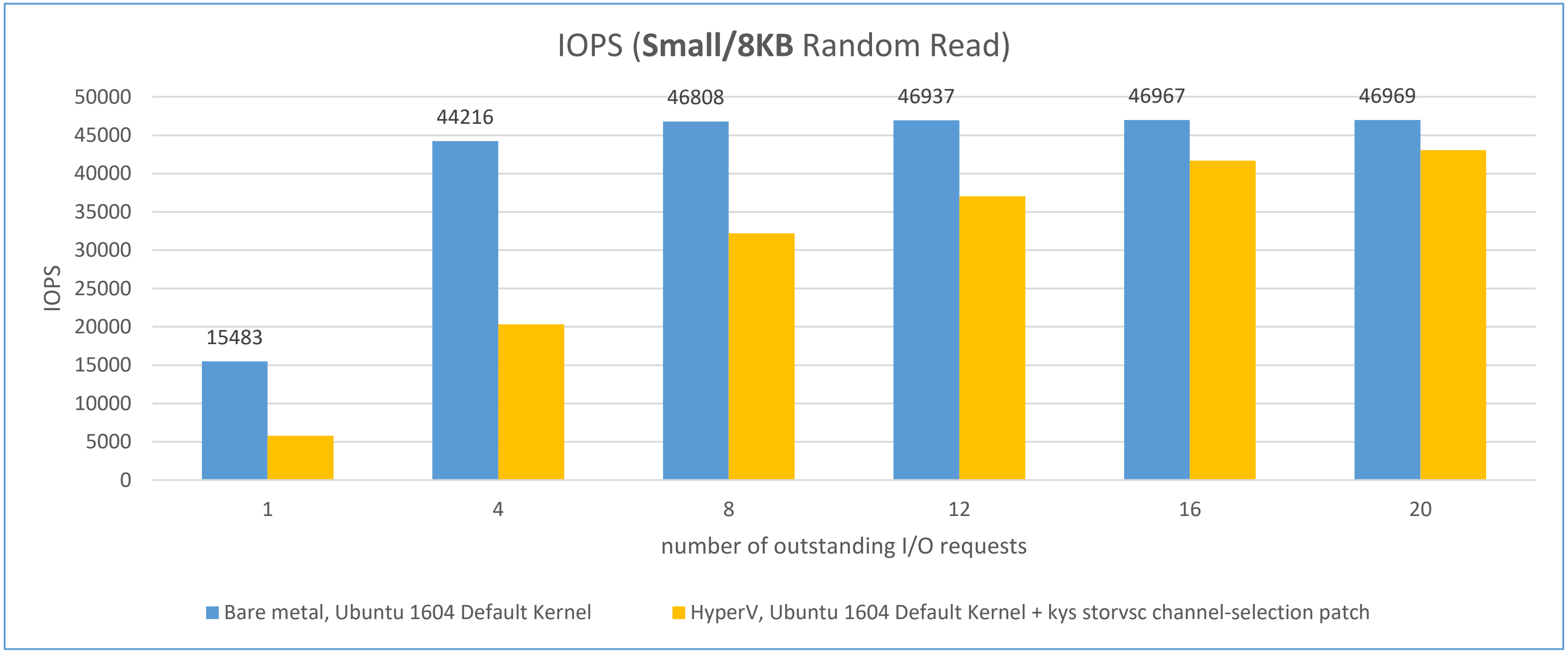
# TCP Connections	Throughput (Gbps)					
	Linux Bare Metal	Windows Bare Metal	Linux on HyperV	Windows on HyperV	Linux on KVM	Windows on KVM
1	23.54	11.78	6.5	11.78	26.09	4.69
2	37.08	13.59	11.46	11.72	30.50	8.41
4	37.48	22.17	16.84	15.33	26.81	11.49
8	37.52	34.77	16.15	16.72	27.76	11.22
16	37.58	36.15	28.17	27.05	28.55	12.25
32	37.54	36.16	31.9	33.12	26.25	15.39
64	37.64	36.15	33.84	33.82	25.49	13.81
128	37.67	36.15	34.88	33.70	21.60	13.03
256	37.69	36.14	34.9	32.95	16.83	13.05
512	37.77	36.15	33.23	31.39	12.47	11.48

MongoDB on Local (YCSB Scenario A)

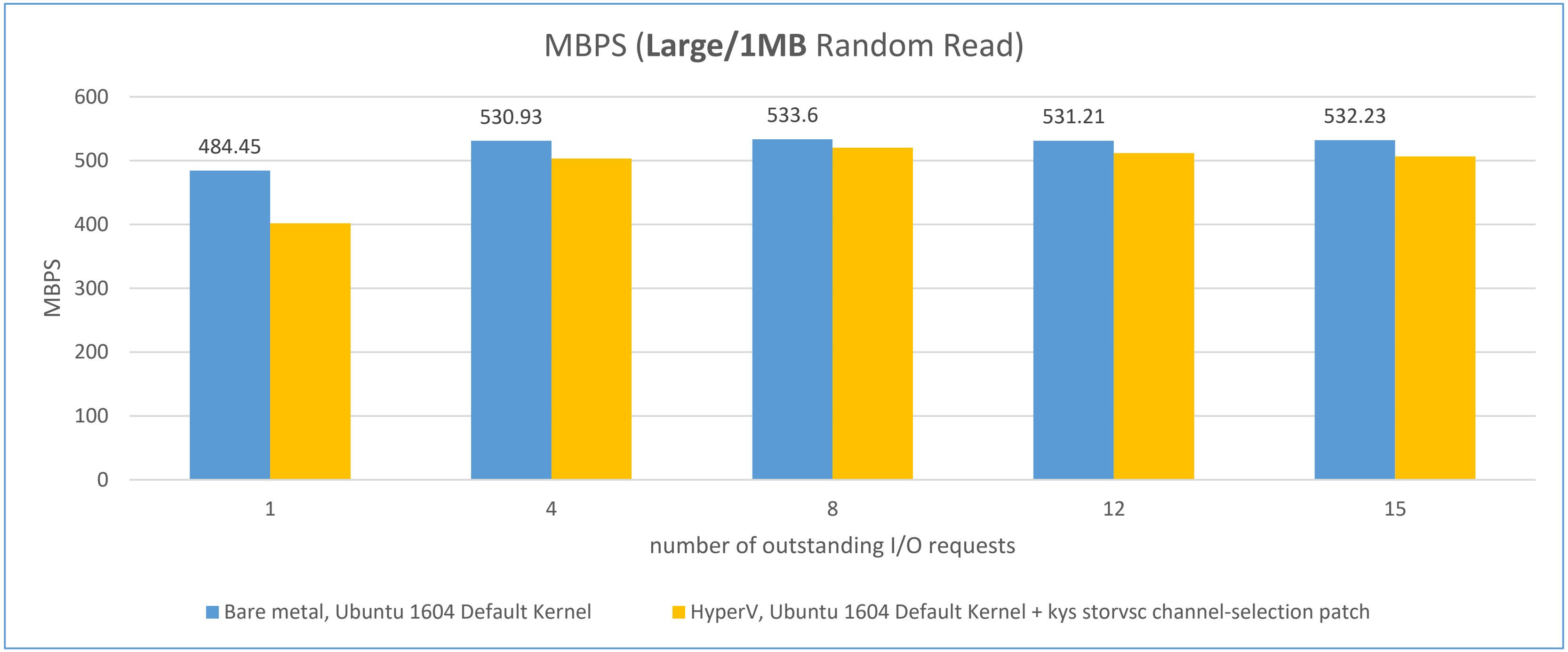
Read: Update = 50:50



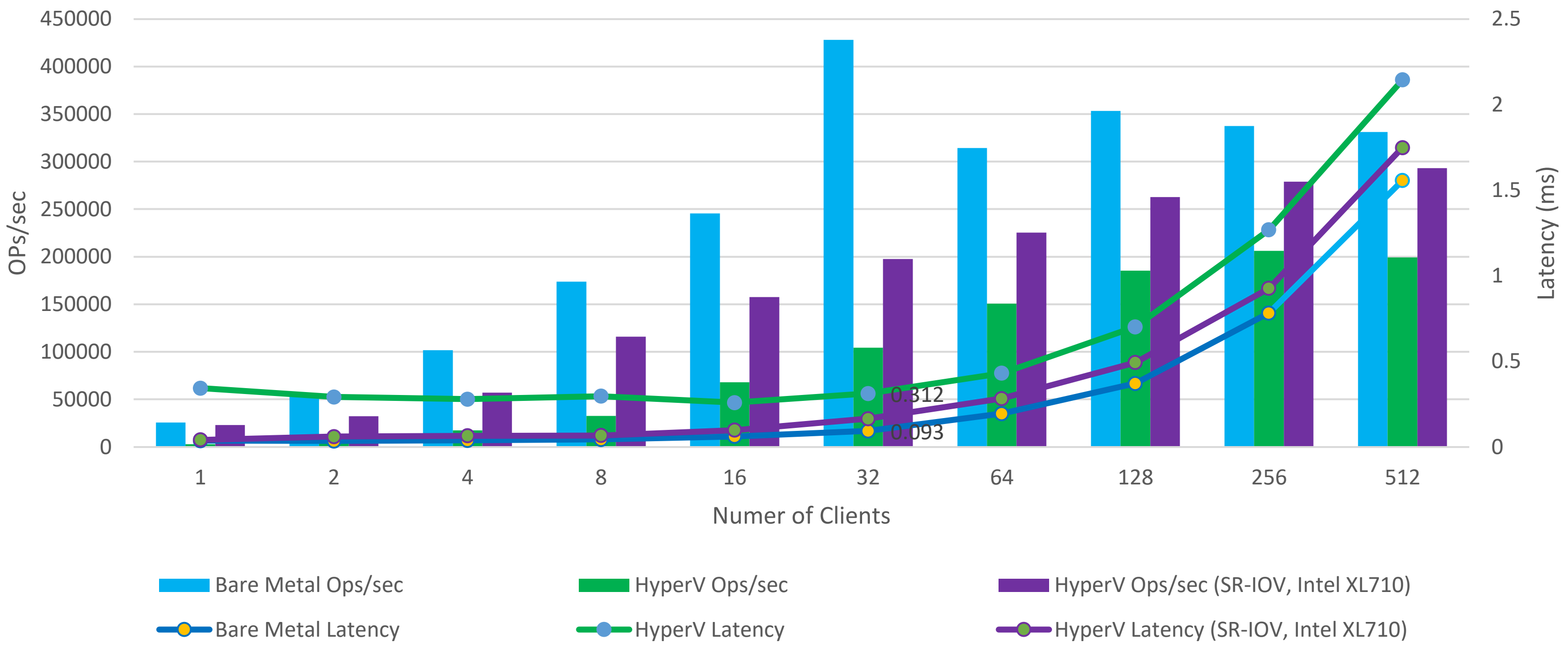
Orion on Local (OLTP Scenario)



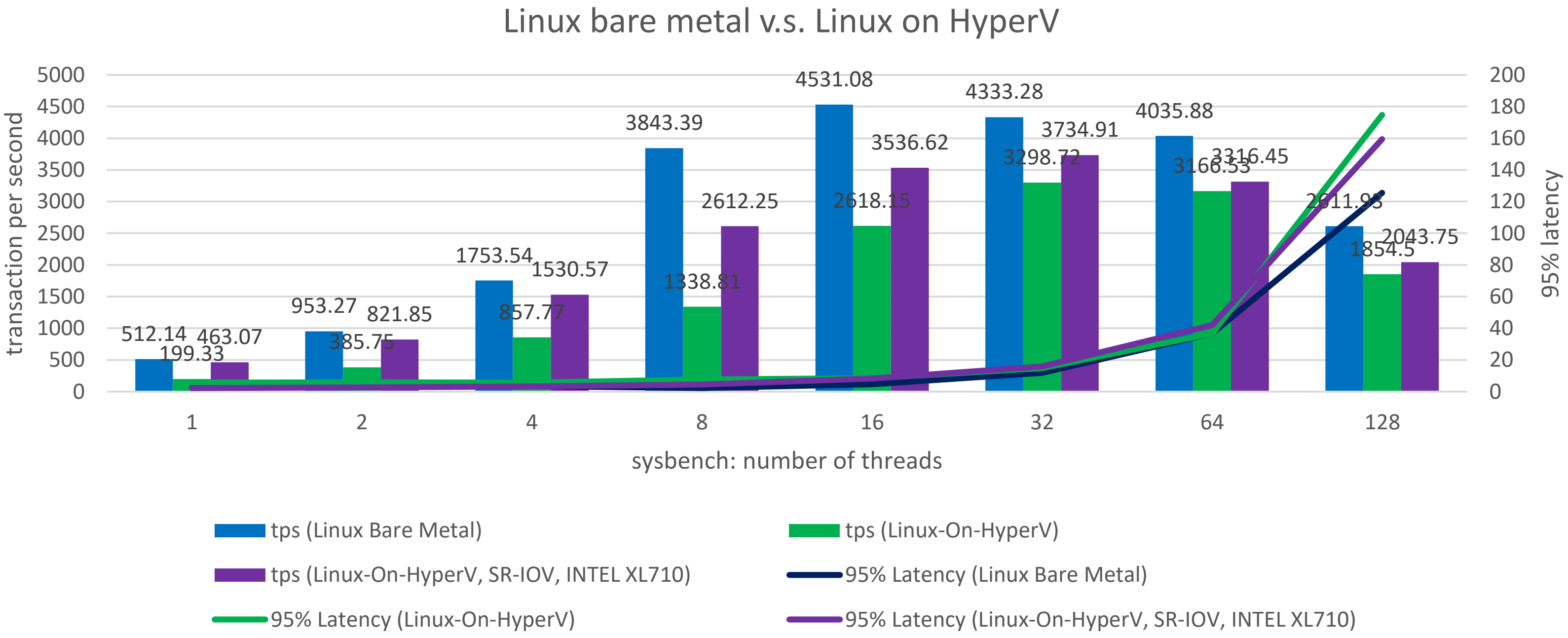
Orion on Local (DSS Scenario)



Memcached


450000
400000
350000
300000
250000
200000
150000
100000
50000
0
OPs/sec
2.5
2
1.5
1
0.5
0
Latency (ms)
0.312
0.093
1
2
4
8
16
32
64
128
256
512
Numer of Clients
Bare Metal Ops/sec
HyperV Ops/sec
HyperV Ops/sec (SR-IOV, Intel XL710)
Bare Metal Latency
HyperV Latency
HyperV Latency (SR-IOV, Intel XL710)

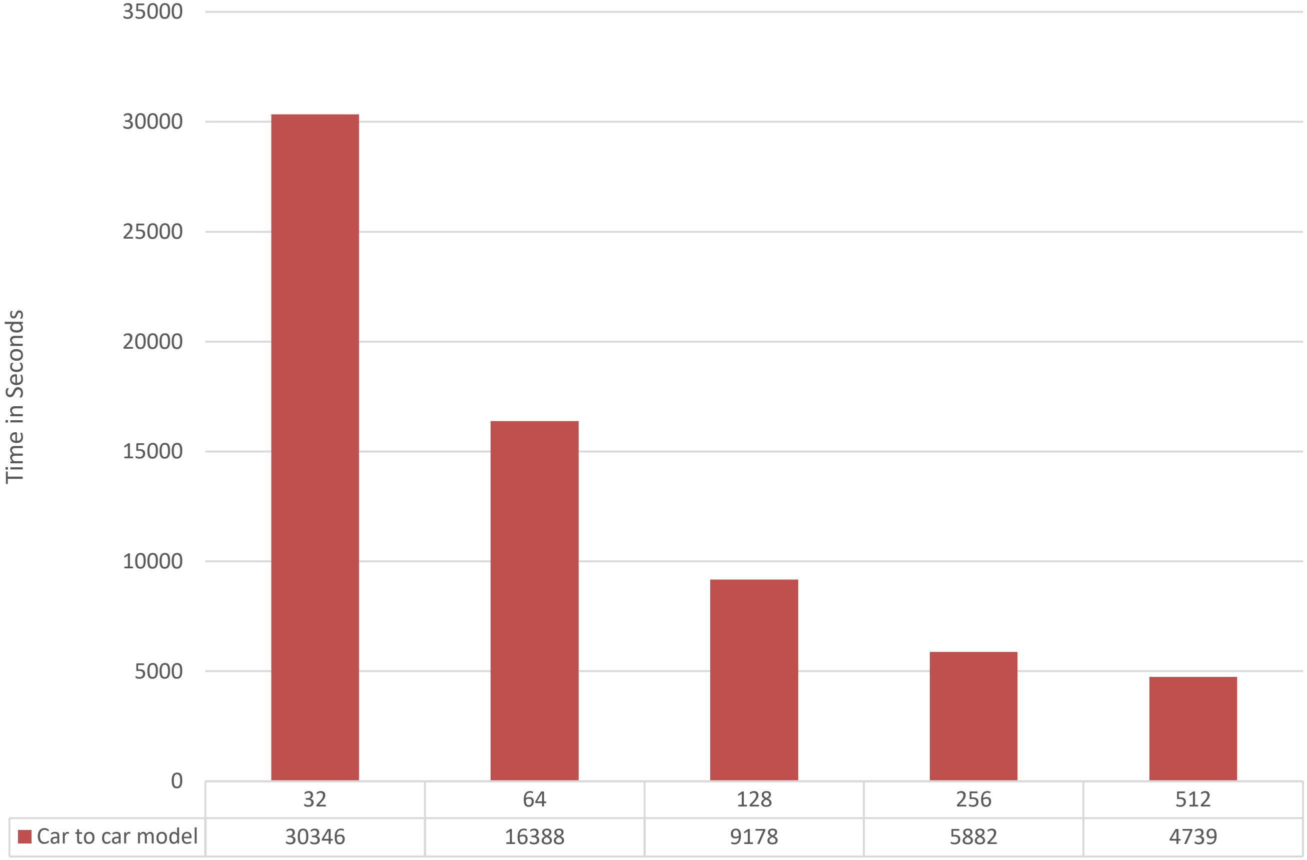
MariaDb



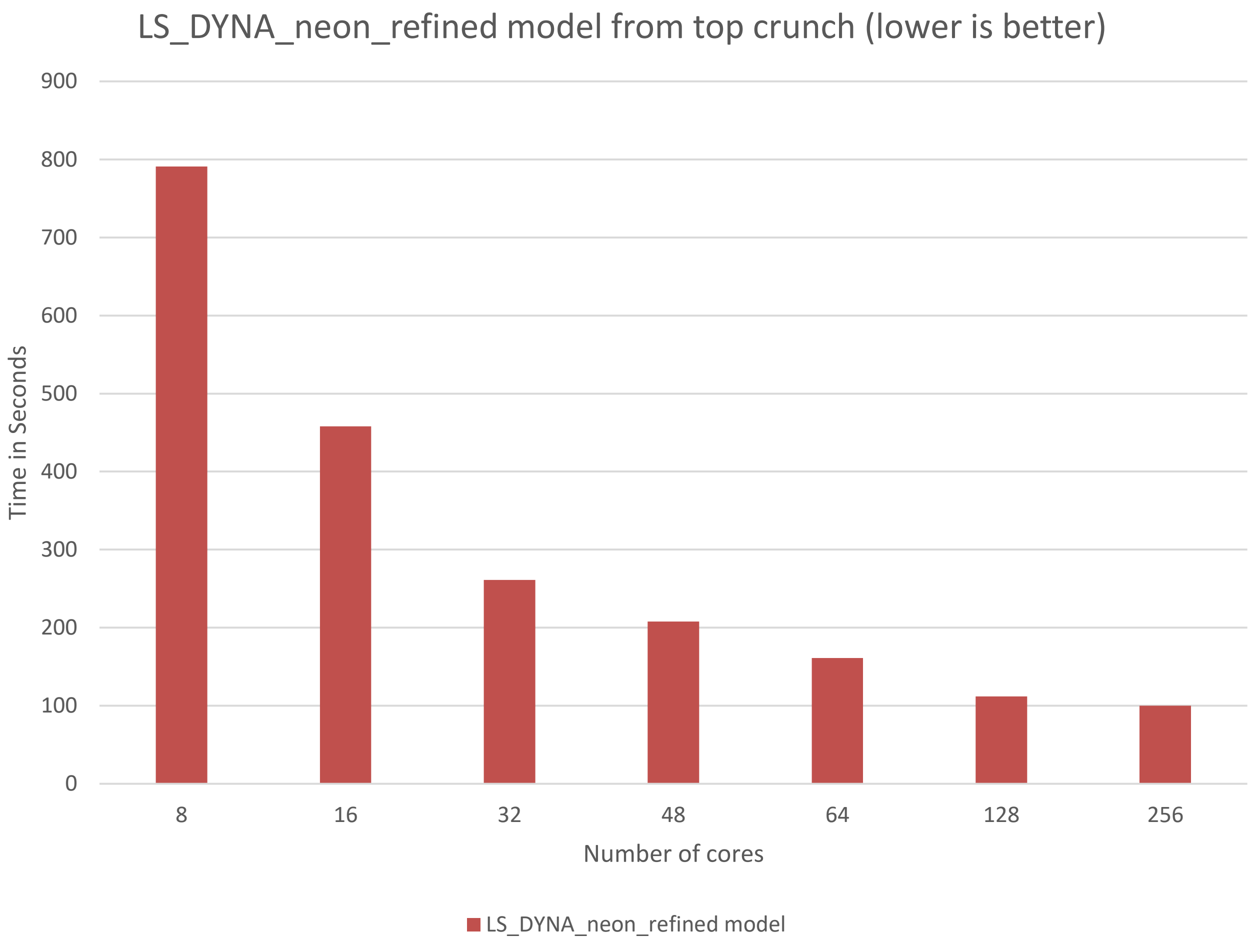
Linux bare metal v.s. Linux on HyperV
transaction per second
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
95% latency
200
180
160
140
120
100
80
60
40
20
0
512.14
199.33
463.07
953.27
385.75
821.85
1753.54
857.77
1530.57
3843.39
1338.81
2612.25
4531.08
2618.15
3536.62
4333.28
3298.72
3734.91
4035.88
3166.53
3316.45
2611.93
1854.5
2043.75
1
2
4
8
16
32
64
128
sysbench: number of threads
tps (Linux Bare Metal)
tps (Linux-On-HyperV)
tps (Linux-On-HyperV, SR-IOV, INTEL XL710)
95% Latency (Linux Bare Metal)
95% Latency (Linux-On-HyperV)
95% Latency (Linux-On-HyperV, SR-IOV, INTEL XL710)

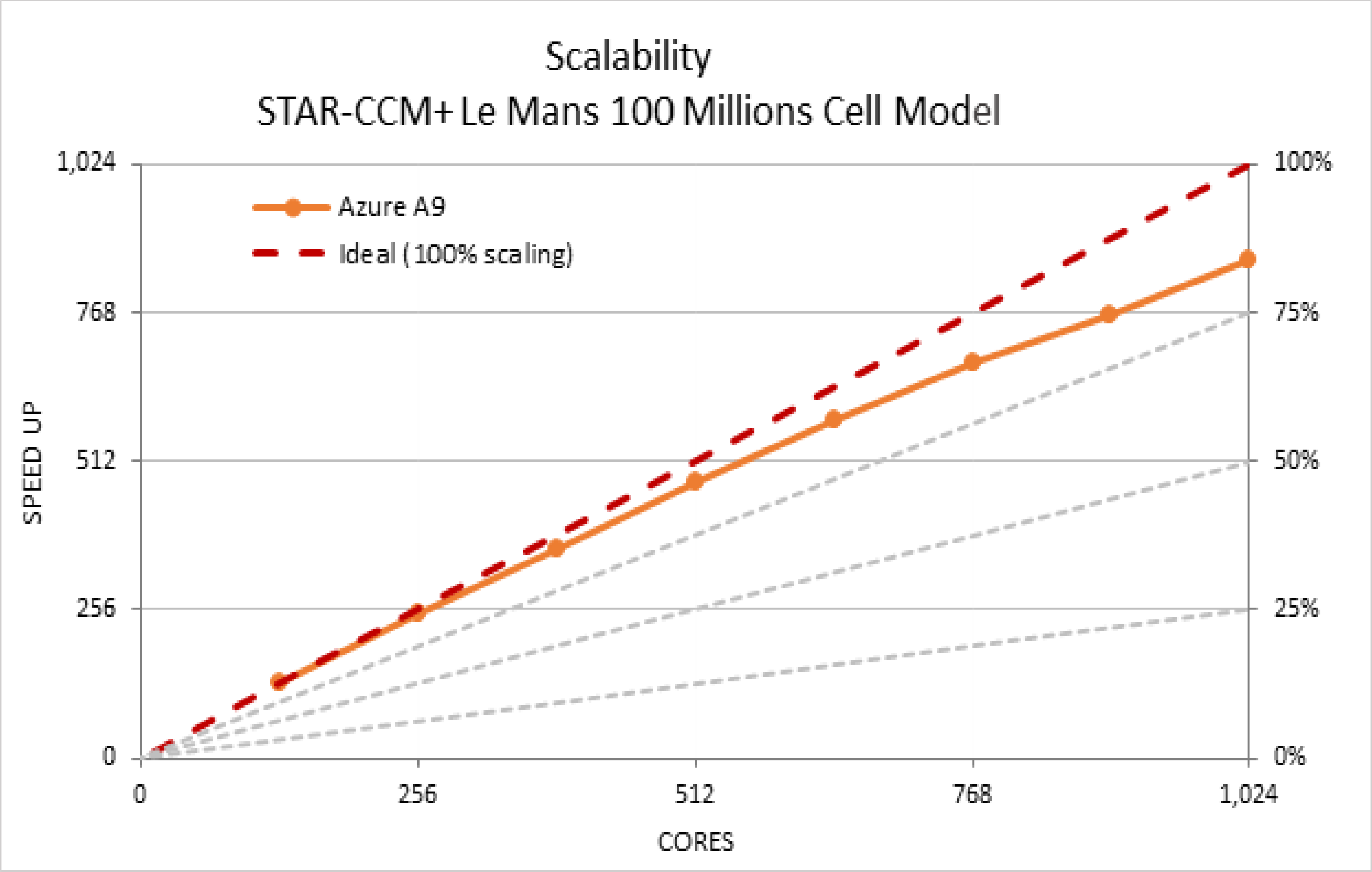
Linux RDMA benchmark data in Azure

LSDYNA CAR TO CAR Benchmark (Lower is better) from Top Crunch



	32	64	128	256	512
🟥 Car to car model	30346	16388	9178	5882	4739


LS_DYNA_neon_refined model from top crunch (lower is better)
900
800
700
600
500
400
300
200
100
0
Time in Seconds
8
16
32
48
64
128
256
Number of cores
🟥 LS_DYNA_neon_refined model


Scalability
STAR-CCM+ Le Mans 100 Millions Cell Model
Azure A9
Ideal (100% scaling)
SPEED UP
1,024
768
512
256
0
100%
75%
50%
25%
0%
0
256
512
768
1,024
CORES

QUESTIONS?

kys@microsoft.com