Demonstrating Performance Capabilities of Red Hat OpenShift for Running Scientific HPC Workloads

November 11, 2020 | David Gray, Kevin Pouget | 12-minute read

Hybrid cloud

< Back to all posts

The Performance and Latency Sensitive Applications (PSAP) team at Red Hat works on the enablement and optimization of Red Hat OpenShift to run compute-intensive enterprise workloads and HPC applications effectively and efficiently. As a team of Linux and performance enthusiasts who are always pushing the limits of what is possible with the latest and greatest upstream technologies, we have been experimenting with a proof-of-concept (POC) implementation of deploying scientific HPC workloads on OpenShift using the MPI Operator to test the performance.

This blog post is a follow-up to the previous blog post on running GROMACS on Red Hat OpenShift Container Platform (OCP) using the Lustre filesystem. In this post, we will show how we ran two scientific HPC workloads on a 38-node OpenShift cluster using CephFS with OpenShift Container Storage in external mode. We will share the benchmarking results of MPI Microbenchmarks, GROMACS, and SPECFEM3D Globe. We ran these workloads on OpenShift and compared them against the results from a bare-metal MPI cluster using the same hardware.

GROMACS is a package for running molecular dynamics simulations. We benchmarked with the popular "water" dataset, which simulates hundreds of thousands of water molecules. This benchmark is a part of the Phoronix benchmark suite. Typically, GROMACS with MPI scales well up to 20 nodes, but beyond that a high-speed networking technology, like the Infiniband, becomes a must.

SPECFEM3D_Globe is a geophysics application that simulates earthquakes and seismic wave propagation in the earth's crust at a continental and global scale. It's a reference benchmark for supercomputers, thanks to its good scaling capabilities.

Both SPECFEM and GROMACS are scientific workloads which support multithreading via OpenMP, and distributed-memory parallelization over multiple nodes with MPI. They also both support GPU acceleration which can be enabled in OpenShift via the NVIDIA GPU Operator, although this is out of the scope of this post.

Setup

Test Environment

The cluster we used was made up of 38 SuperMicro 5039ms machines for the OpenShift cluster and four Dell R930 machines for the Ceph cluster. We ran a provider agnostic installation of OCP 4.5.7 on physical servers (bare-metal) with one provisioning node, three masters, and 34 worker nodes. Two of the worker nodes were used to isolate infrastructure Pods. Each of the SuperMicro 5039ms OpenShift nodes were single socket systems with four real cores plus hyper threads, 2x25GbE network interfaces. One of the 25GbE was used for the OpenShift cluster, while the other was available for MPI traffic. The 10GbE interfaces were unused.

For the reference results on bare-metal, we used RHEL 8.2 and OpenMPI 4.0.2.

Ceph configuration

We chose to use CephFS for this PoC because it can be used both in OpenShift (using OpenShift Container Storage (OCS) External mode) and on bare-metal by mounting it on RHEL 8.2. OCS is software-defined storage integrated with and optimized for OCP. With OCS, it was easy to access GROMACS and SPECFEM data stored in CephFS, by mounting a ReadWriteMany PersistentVolumeClaim inside the MPIJob worker Pods.

We installed Red Hat Ceph Storage (RHCS) on the four Dell machines. In OpenShift, we access this CephFS instance using the OCS operator installed from GitHub in External Mode. The OCS external-mode feature allows on-premise OCS deployments to consume a Red Hat Ceph

Storage instance external to the OpenShift cluster. This way, we were able to use the same CephFS filesystem for our bare-metal benchmarking and on OpenShift via OCS. Alternatively, OCS can create a Ceph cluster internal to an OpenShift cluster.

Note: GROMACS and SPECFEM performance was not impacted by the filesystem performance according to our tests. For this reason, we did not spend time optimizing the performance of our Ceph configuration, and won't go into details of how Ceph was installed and configured. Installation and configuration procedures for RHCS can be found in the documentation.

Running MPI applications with the MPI Operator

To install the MPI Operator, see the instructions on the Kubeflow MPI Operator GitHub repository. For the benchmarking we've done, we used version 0.2.3 of the MPI Operator because of a known bug in the latest upstream which causes MPIJobs to fail to run on OpenShift.

The MPI Operator defines the MPIJob CustomResource. To run GROMACS, SPECFEM3D Globe, and the microbenchmarks, we create MPIJob instances to be managed by the MPI Operator using a YAML manifest. The MPIJob manifest specifies the mpirun command, the number of launcher pods in which to run the mpirun command (usually one), the number of worker Pods, and the number of MPI slots per worker pod. When an MPIJob object is created, the MPI Operator creates the launcher and worker Pods and runs the specified mpirun command.

An example MPIJob for running GROMACS can be found here. SPECFEM MPIJob template can be found here.

Using Multus with the MPI Operator

One of the things which we will get into in the performance results is bypassing the software-defined overlay network (SDN) in OpenShift. For certain types of network traffic, the SDN can have some performance overhead due to encapsulation and OpenFlow rules. In general, throughput is not diminished but latency can be affected. Besides, the SDN is shared with all OpenShift Pods, which introduces additional variability in the measurements.

One way to bypass the SDN is by giving Pods access to the host's network namespace, with the hostNetwork: true flag in the Pod spec. However, this has security implications, as it could be used to snoop on network activity of other Pods on the same node if the Pod were somehow compromised. Fortunately, we can instead bypass the SDN by using Multus.

Multus is an open source project that enables Kubernetes Pods to attach to multiple networks. It is a Container Networking Interface (CNI) meta-plugin, meaning that it can call other CNI plugins. For bypassing the SDN with MPI Operator worker Pods, we first create an additional

network using the following NetworkAttachmentDefinition manifest 'nad.yaml':

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: mpi-network
  namespace: default
spec:
  config: '{
   "cniVersion":"0.3.1",
   "type": "macvlan",
   "master": "enp1s0f1",
   "ipam": {
   "type": "whereabouts",
   "range": "11.10.0.0/24"
   }
  }'
```

In the spec section of nad.yaml, notice the type macvlan, which specifies that the macvlan CNI plugin will be used. This plugin allows Pods to communicate with other Pods on the same and other hosts, using a physical network interface. When a Pod is attached to the macvlan-based network, the plugin creates a sub-interface from the parent interface on the node, and each Pod gets a unique MAC address. In this case, we are specifying the physical interface "enp1s0f1" which is one of the two 25GbE interfaces on the worker nodes. Also, note the spec.ipam section, which specifies how IP addresses are managed on the additional network. In this case, we are using Whereabouts IP address management (IPAM), which is officially a part of OCP as of version 4.5. Whereabouts assigns cluster-unique IP addresses to Pods on the macvlan-based network.

To create the additional network, we simply run

```
oc create -f nad.yaml
```

To use the network in a Pod, we add an annotation to its metadata. This also works with MPIJobs, by adding the annotation to the launcher and worker Pods. For example:

```
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
...
spec:
  mpiReplicaSpecs:
    ...
    Worker:
      ...
```

```
      template:
        metadata:
          annotations:
            k8s.v1.cni.cncf.io/networks: mpi-network
...
```

To verify that the network has been attached to a pod , run oc describe:

```
$ oc describe pod/multus-test-pod
...
Events:
  Type Reason      Age   From                                  Message
  ---- ------      ----  ----                                     -------
...
Normal  AddedInterface  56s   multus                                Add net1 [11.10.0
...
```

Running SPECFEM3D_Globe benchmarking

Running SPECFEM3D_Globe on OpenShift requires the help of a Go client, to coordinate the following:

- Parallel execution of the mesher
- Build of the solver, that uses a header file generated by the solver
- Execution of the solver.

The design and implementation of this Go client is described in this blog post.

In the context of this HPC benchmarking work, we wanted to compare SPECFEM performance in multiple platform setups (bare-metal; OpenShift with SDN, Multus or HostNetwork networking), and with multiple problem sizes (ranging from a few minutes of execution to several hours). And as the aim of this work was to demonstrate the scaling of the cluster, we had to run the benchmark with different node counts. In the end, we collected more than 180 measurements.

To automate the execution of this heavy-weight benchmarking, we reused a tool developed as part of another Red Hat project (see its presentation and repository). The tool takes as input a list of parameters to benchmark:

```
platform:  baremetal, podman, openshift
network: default, multus, hostnet
relyOnSharedFS: True
extra:
```

```
# 32 nex
- nex=32, processes=1, mpi-slots=1 # 1 machine
- nex=32, processes=4, mpi-slots=2 # 2 machines
- nex=32, processes=4, mpi-slots=1 # 4 machines

# 64 nex
- nex=64, processes=1, mpi-slots=1 # 1 machine
- nex=64, processes=4, mpi-slots=2 # 2 machines
- nex=64, processes=4, mpi-slots=1 # 4 machines
- nex=64, processes=16, mpi-slots=2 # 8 machines
- nex=64, processes=16, mpi-slots=1 # 16 machines
- nex=64, processes=64, mpi-slots=2 # 32 machines
...
```

With the help of some Python magic, the tool can run SPECFEM OpenShift client or our bare-metal execution script. At the end of the execution, it collects the final execution time from SPECFEM log file, and stores it for offline processing:

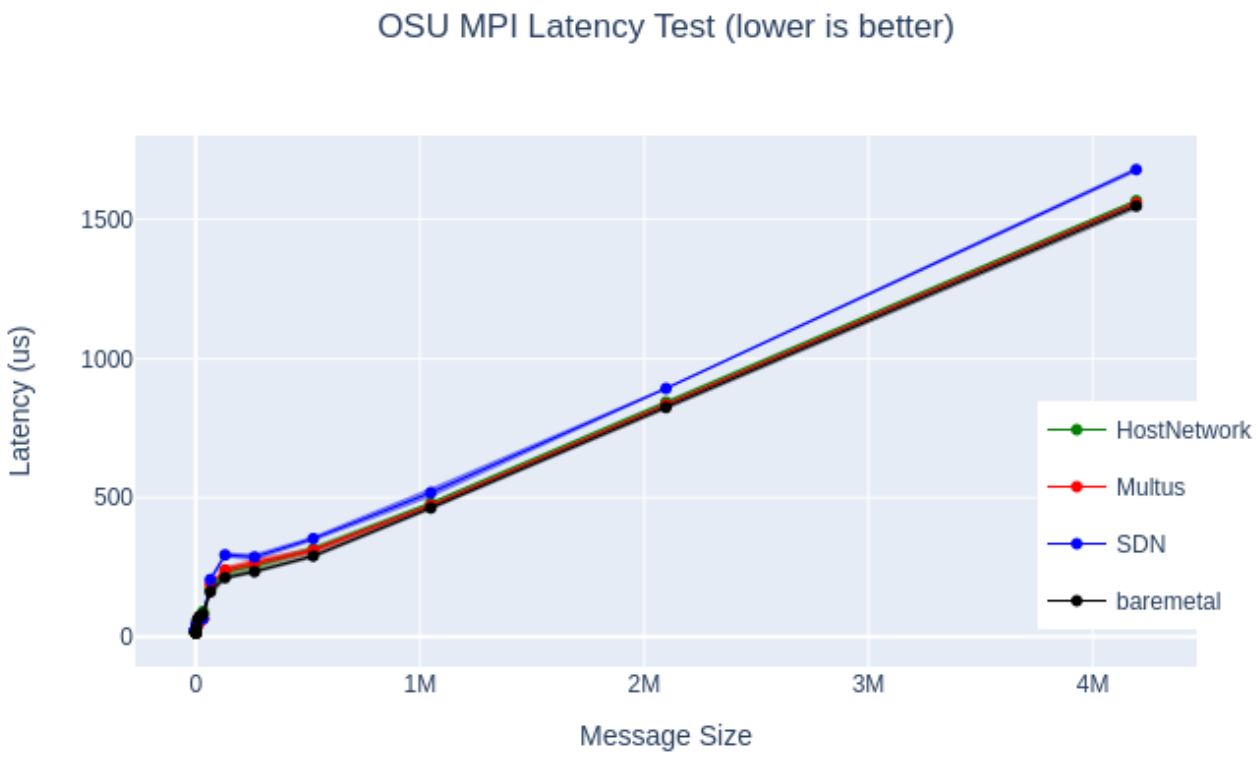
```
[...]
13:32:00 running 67/72
13:32:00 > driver=None_relyOnSharedFS=True_platform=baremetal_network=default_rur
13:32:00 apply_settings: relyOnSharedFS=True, platform=baremetal, network=default
13:32:00 clear graphs
13:32:00 Waiting for the end of the execution
.........................................
FINISHED: Specfem finished after 9151.703304007999s (=152min)
16:04:45 Timing table has 1 records.
16:04:45 save record into results/openshift/default/baremetal/relyOnSharedFS=True
16:04:45
16:04:45 running 68/72
[...]
```

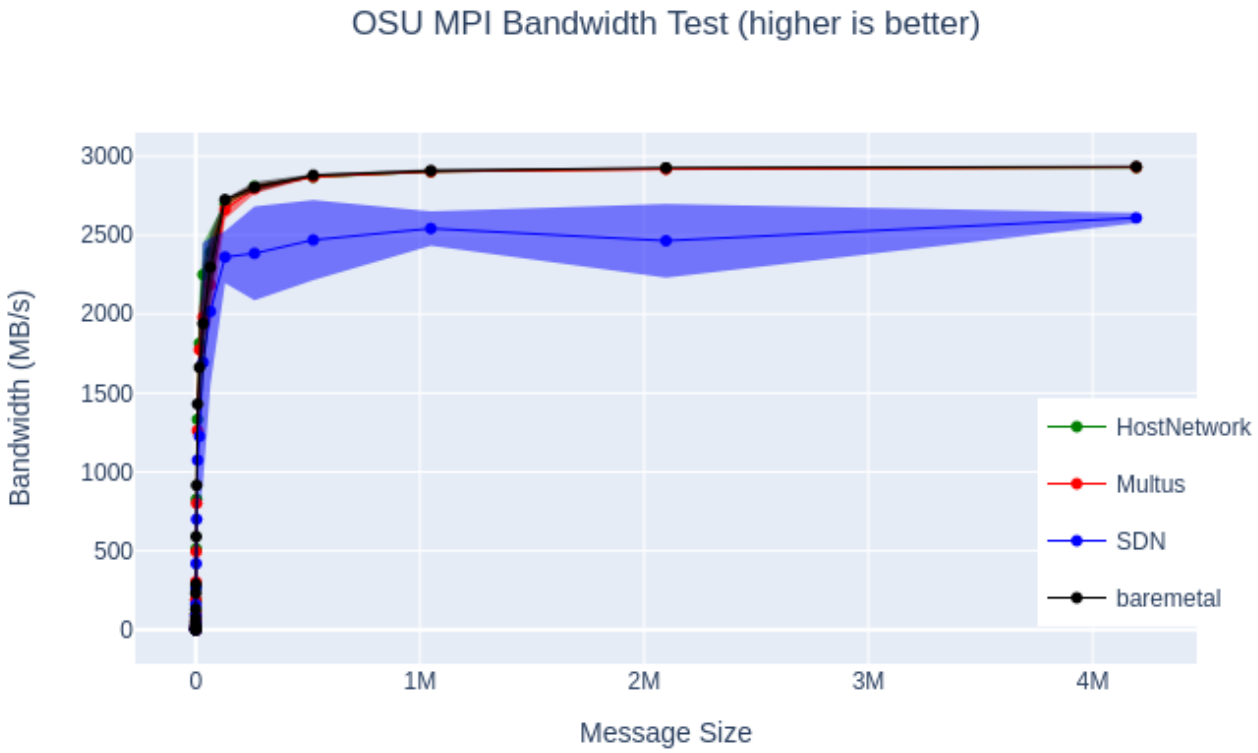
The goal of this tool is to automatize the benchmark execution and result collection. This avoids repetitive manual interventions that are error-prone, and it also facilitates the execution of long benchmarks that can be carried out overnight or weekends. The tool also provides a plotting framework for visualizing the benchmarks results. We wrote a graph plugin that was used to generate the SPECFEM and GROMACS plots presented in the Results section.

Results

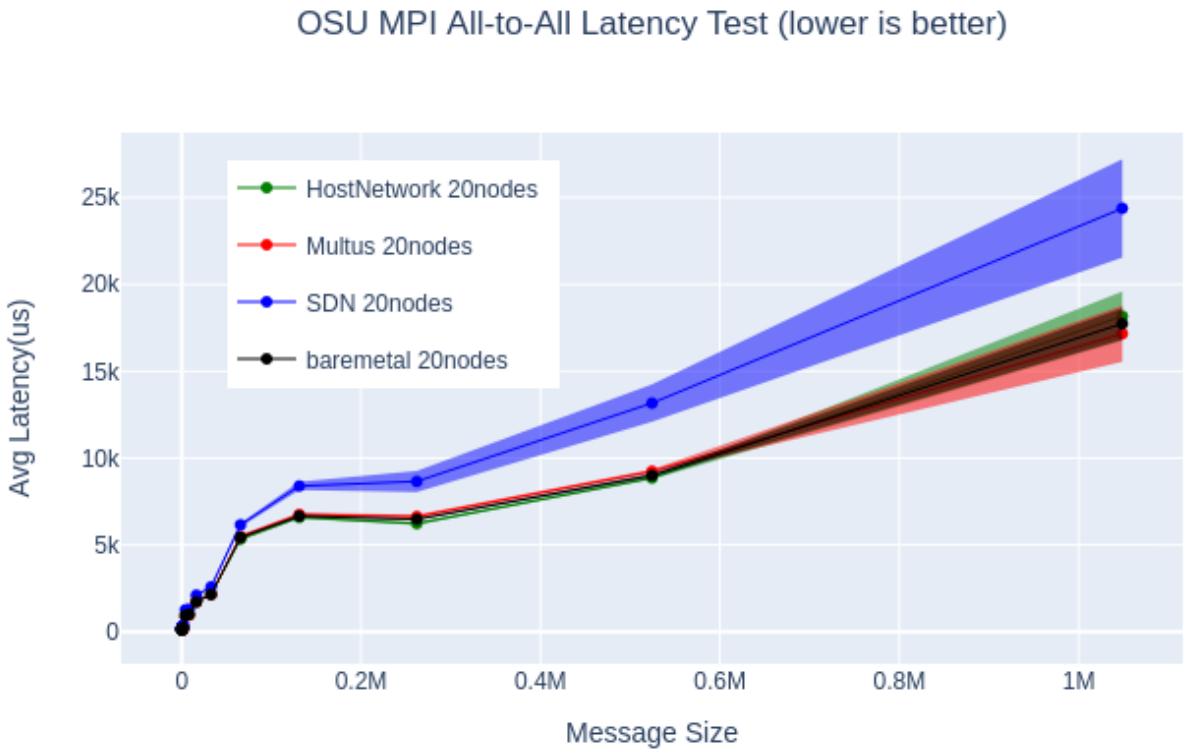
MPI Microbenchmarks

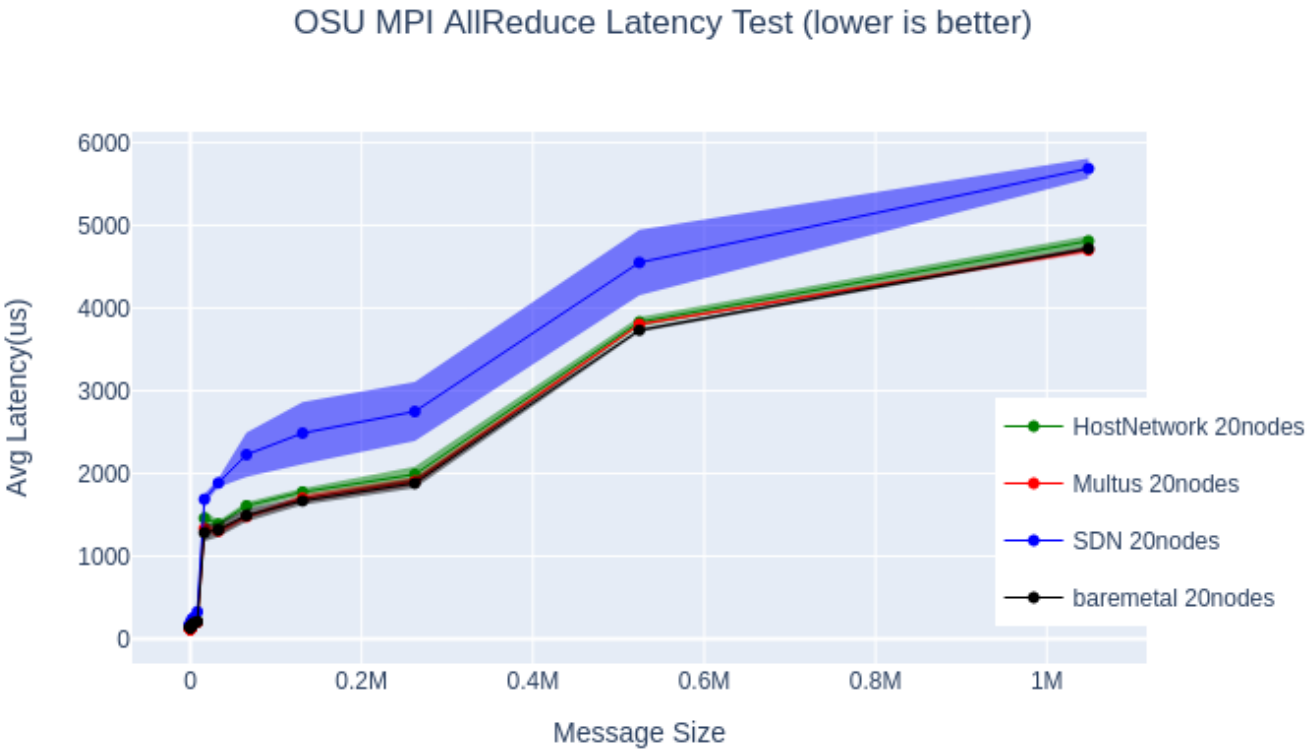
In the previous blog post, we mentioned the significant performance improvement we could get on GROMACS by using hostNetwork: true. As mentioned above, using Multus to create a secondary network on one of the 25GbE interfaces offers a more secure method of bypassing the SDN. We ran MPI microbenchmarks which test peer-to-peer bandwidth and latency at different message sizes to see how the SDN overhead impacts performance, and to verify that our macvlan network using Multus was on par with the hostNetwork results. The average results of testing with many different nodes are shown below:





We also experimented with two microbenchmarks testing the performance of two common MPI collective communications: all-to-all personalized broadcast, and all-reduce


OSU MPI All-to-All Latency Test (lower is better)
HostNetwork 20nodes
Multus 20nodes
SDN 20nodes
baremetal 20nodes
Avg Latency(us)
25k
20k
15k
10k
5k
0
0
0.2M
0.4M
0.6M
0.8M
1M
Message Size

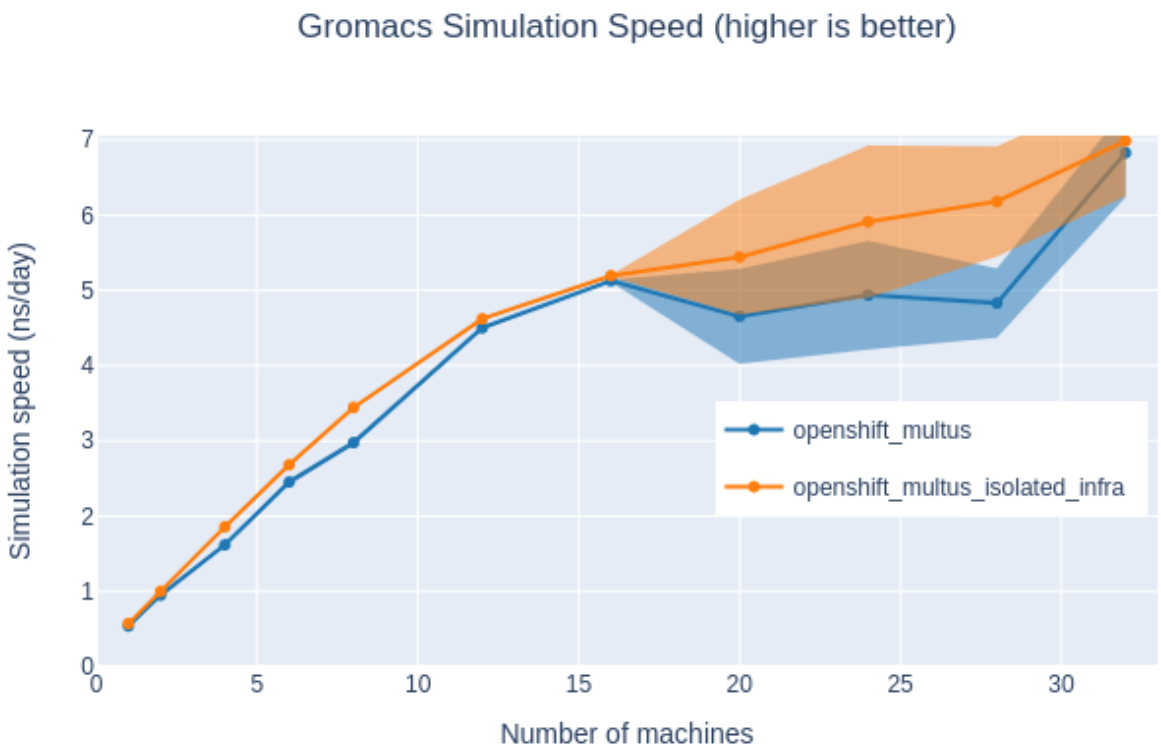

OSU MPI AllReduce Latency Test (lower is better)
Avg Latency(us)
6000
5000
4000
3000
2000
1000
0
0
0.2M
0.4M
0.6M
0.8M
1M
Message Size
HostNetwork 20nodes
Multus 20nodes
SDN 20nodes
baremetal 20nodes

These two sets of microbenchmarks show what we expected to see based on past experiments with GROMACS, as well as past benchmarking of the SDN: the network latency with the SDN can be higher than on bare-metal, especially with larger message sizes.

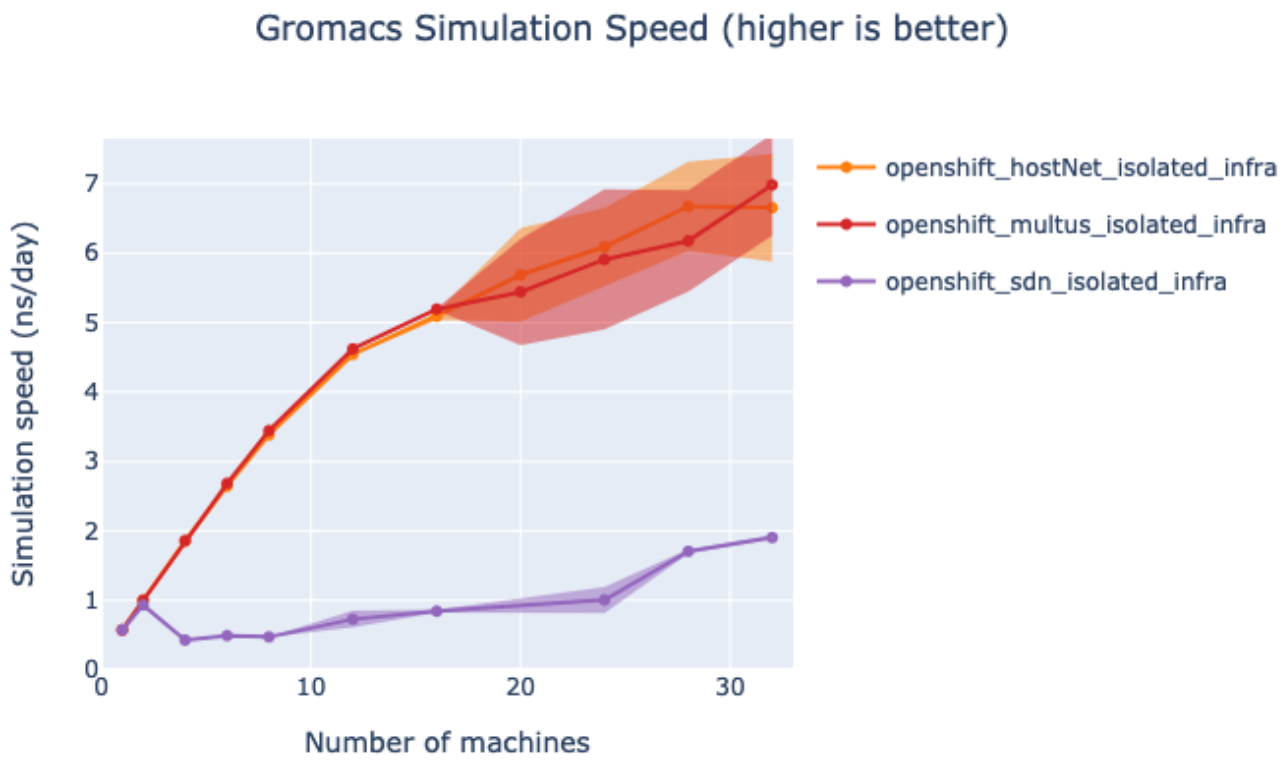
GROMACS

The results for GROMACS are collected from running the "water" benchmark 5 times. We display the average and highlight the range within one standard deviation, to visualize how the performance varies more at higher node counts.

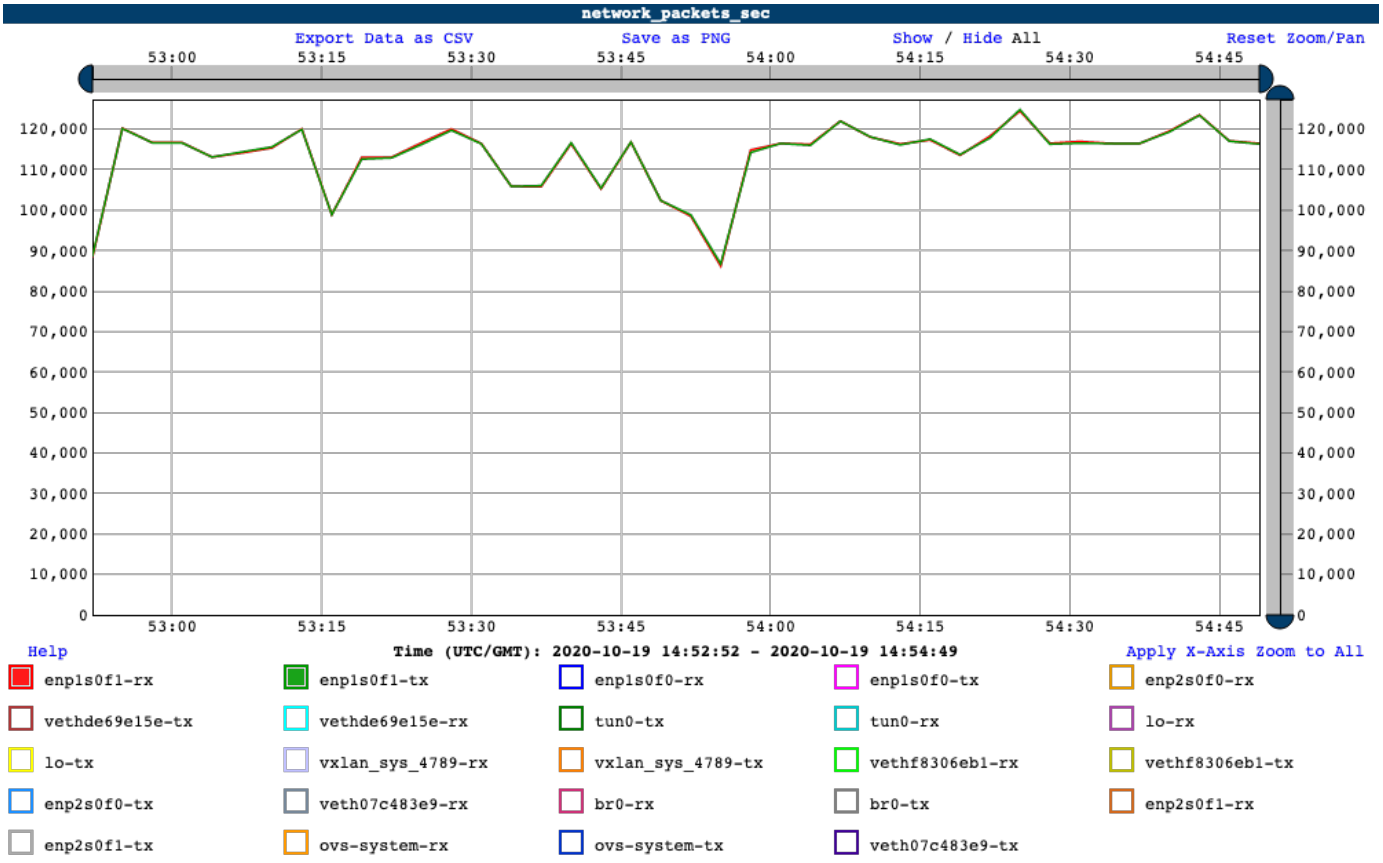
One of the first things we experimented with is isolating infrastructure pods to two specific nodes, to reduce some of the overhead on the worker nodes running the workload. We moved the router pod (ingresscontroller), the image registry, and the monitoring stack (Prometheus, Grafana, AlertManager) to these two infrastructure nodes. As the following graph shows, this reduced variability and improved performance. All of the following results were gathered with infrastructure pods isolated to two specific nodes which were not used for the benchmark run.



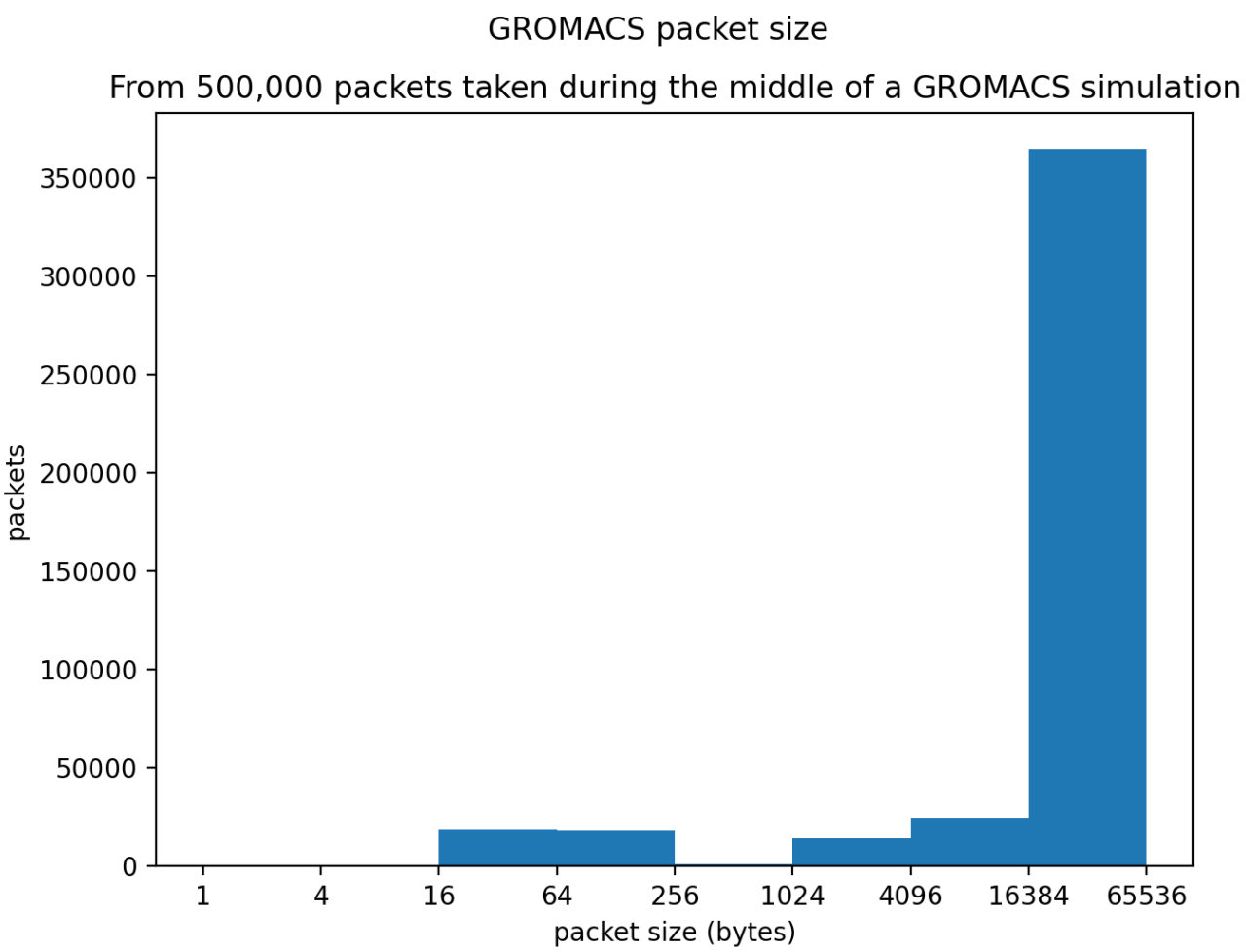
As mentioned previously, we can use Multus to bypass the SDN. The difference between GROMACS performance with the default SDN, and using a secondary network or ĥostNetwork: truè is very significant. At some node counts, using Multus gives over a 7x improvement in performance.



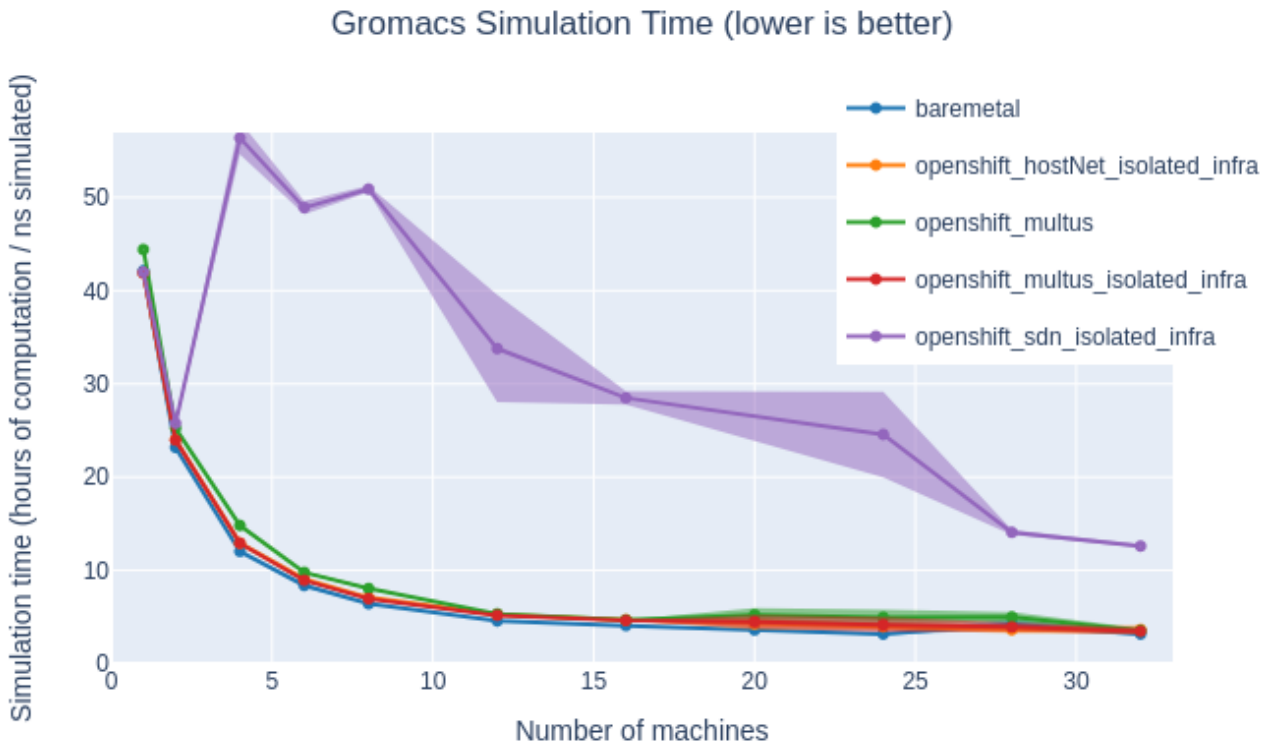
Note that the SDN overhead for GROMACS on this bare-metal cluster is more significant than our initial results on AWS. We have not yet fully determined the reason for this, however, we have gathered some information about the network traffic of multi-node parallel GROMACS test using Pbench. Results from sàr reveal that our nodes are sending/receiving over 100,000 packets/second, and the number increases as the number of nodes increases.

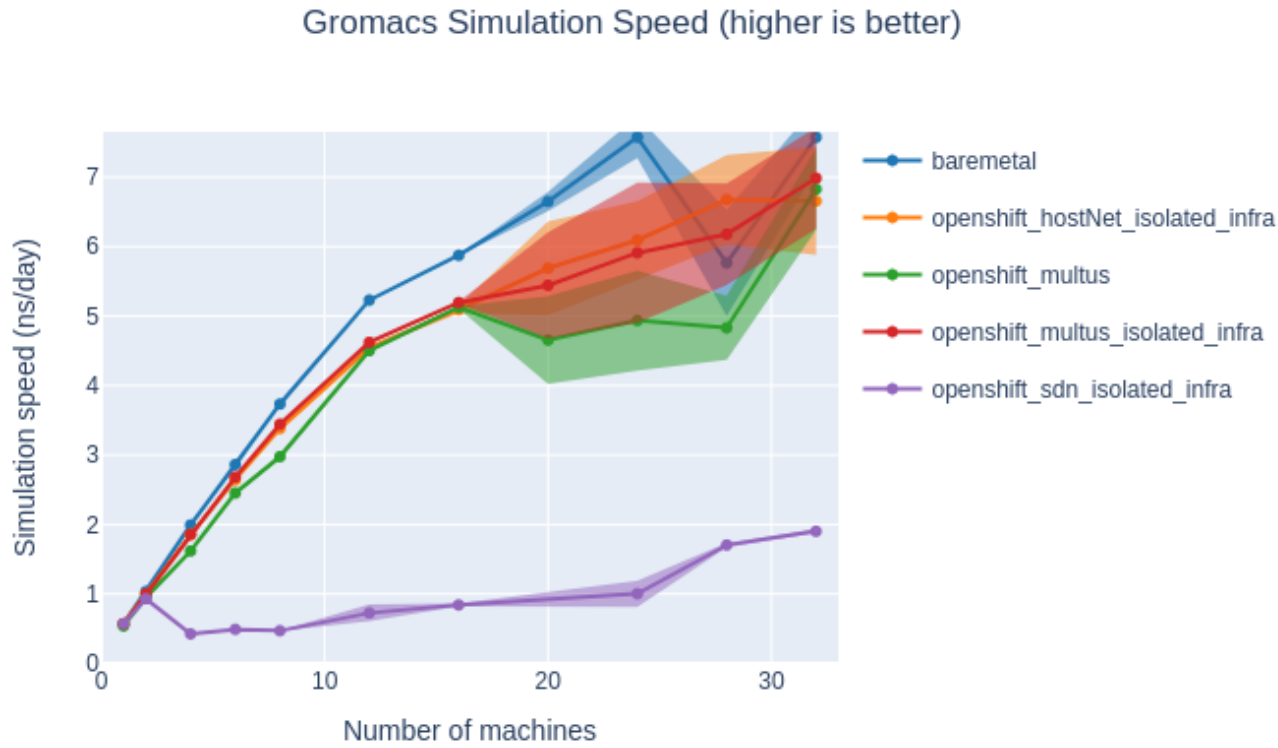


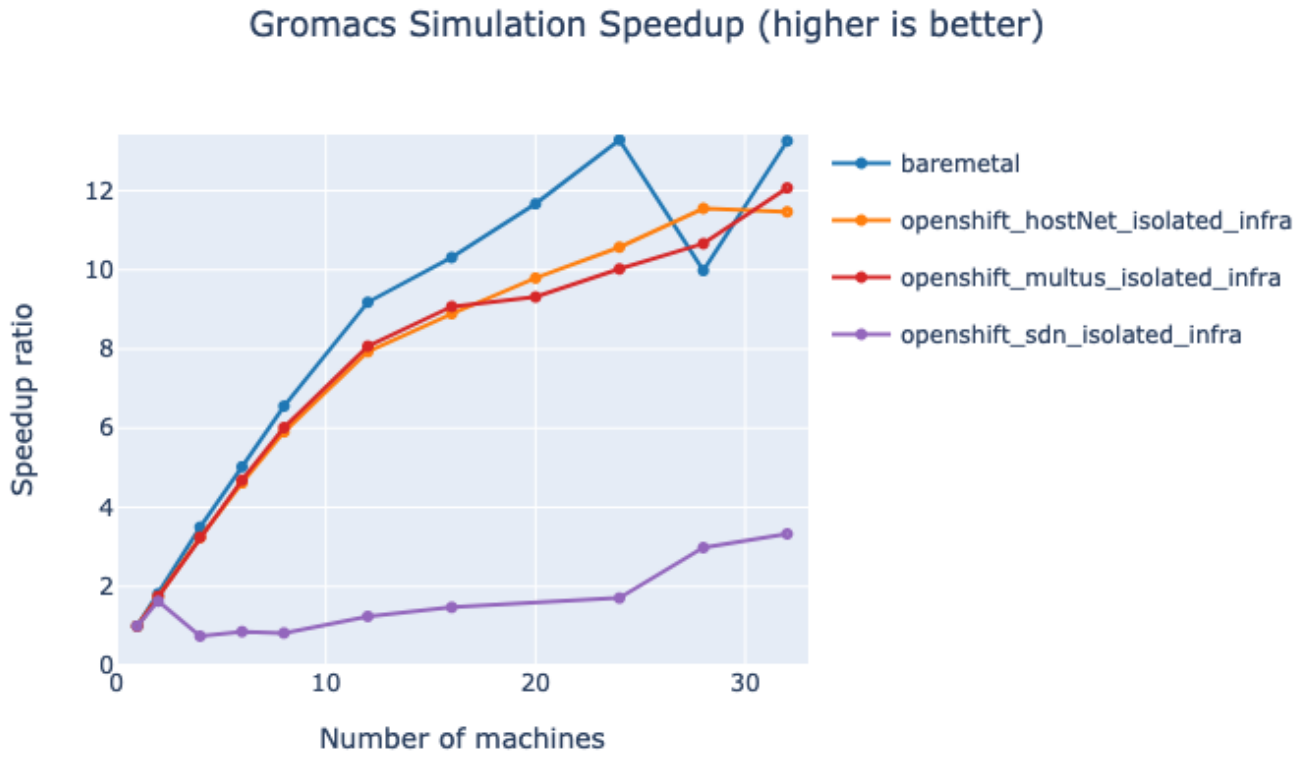
Furthermore, we examined the size of the packets being transmitted using tcpdump (thanks again to Pbench), and saw that the vast majority of the packets are 64Kb in size. From previous benchmarking done by members of the Performance and Scale team, we know that this type of traffic (Request and Response with large packets) is the most impacted by SDN overhead. As we will show in the following section with the results for SPECFEM, not all workloads are highly impacted.

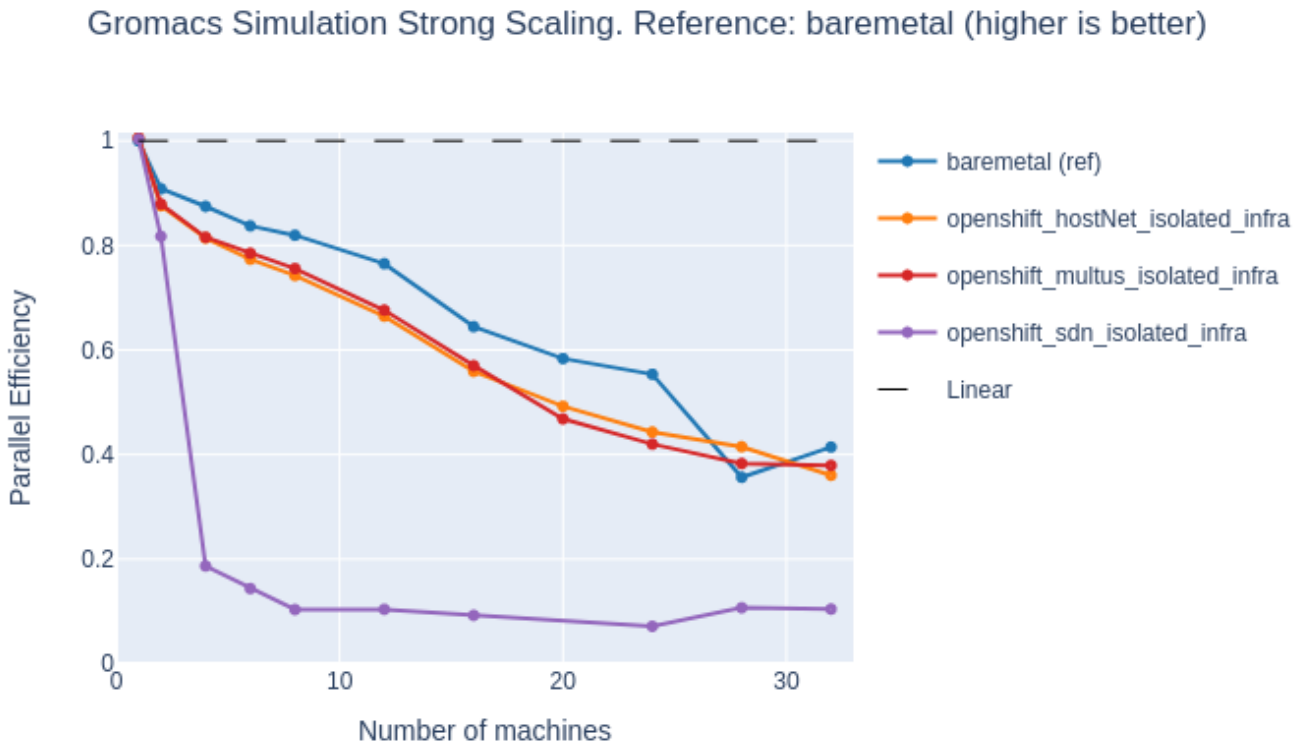


The main goal of this PoC was to see how the performance of HPC workloads running on OpenShift would compare to bare-metal. The complete set of GROMACS results using the default OpenShift SDN, hostNetwork, Multus, and bare-metal are in the following set of graphs. We show the computation time to simulate one nanosecond, the performance in ns/day simulated, the parallel speedup, and the parallel efficiency.

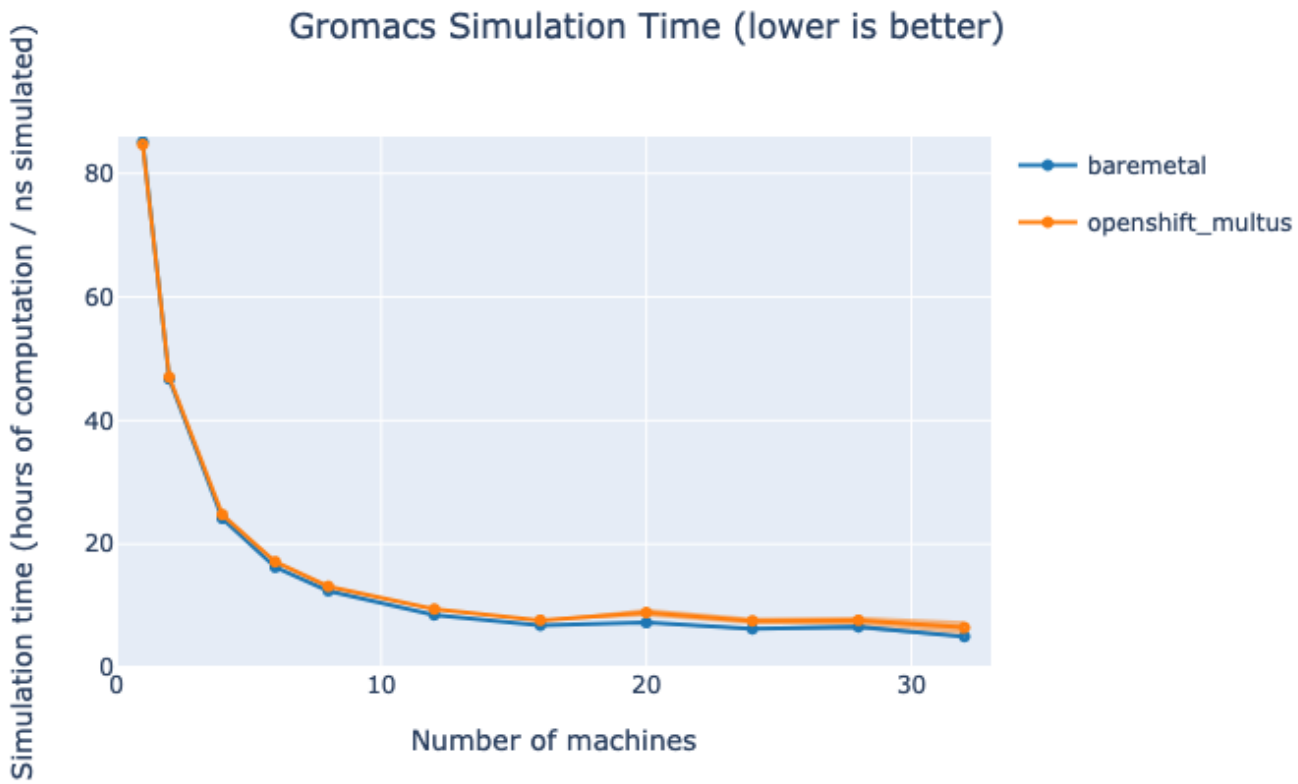

Gromacs Simulation Time (lower is better)
Simulation time (hours of computation / ns simulated)
Number of machines
baremetal
openshift_hostNet_isolated_infra
openshift_multus
openshift_multus_isolated_infra
openshift_sdn_isolated_infra
0
10
20
30
40
50
0
5
10
15
20
25
30


Gromacs Simulation Speed (higher is better)
Simulation speed (ns/day)
Number of machines
baremetal
openshift_hostNet_isolated_infra
openshift_multus
openshift_multus_isolated_infra
openshift_sdn_isolated_infra
0
1
2
3
4
5
6
7
0
10
20
30


Gromacs Simulation Speedup (higher is better)
Speedup ratio
Number of machines
0
2
4
6
8
10
12
0
10
20
30
baremetal
openshift_hostNet_isolated_infra
openshift_multus_isolated_infra
openshift_sdn_isolated_infra



These results show that throughput can be improved significantly by scaling up the nodes. Up to 16 nodes, the performance on OpenShift using the secondary network is within a few percent of the performance on bare-metal. With more than 16 nodes, the performance on both platforms becomes much more variable, but on average the results stay similar, and throughput continues to improve with more nodes. Looking at the simulation time, the number of hours of computation needed to simulate one nanosecond of this simulation does flatten out at about 16 nodes. This aligns with what we have read about running GROMACS on multiple nodes with MPI. To get optimal scaling out of GROMACS, high-speed networking hardware is necessary. Increasing the problem size by doubling the molecules in the simulation results in slightly better scaling, not flattening out until about 20 nodes.



SPECFEM3D_Globe

For the second HPC workload, we ran Specfem3D_Globe with the default model. Let us describe how we configured the benchmark parameters:

Problem size (NEX): we used the "*number of elements at the surface along the two sides of the first chunk*" (NEX_XI and NEX_ETA values) to control the size of the problem solved during the simulation. NEX_XI and NEX_ETA were kept identical, with values 64, 128, 192 and 256. These problems took between a few minutes and 3.5 hours to solve.

Number of processes: the number of processes (controlled by NPROC_XI x NPROC_ETA) that can be used to run Specfem depends on the NEX value. More exactly, *"NEX must be multiple of 16 and 8 x multiple of NPROC"*. In this benchmark, we kept NPROC_XI and NPROC_ETA equal, and we used 1, 2, 4, 16, 32, 36 or 64 processes for the different NEX problem sizes.

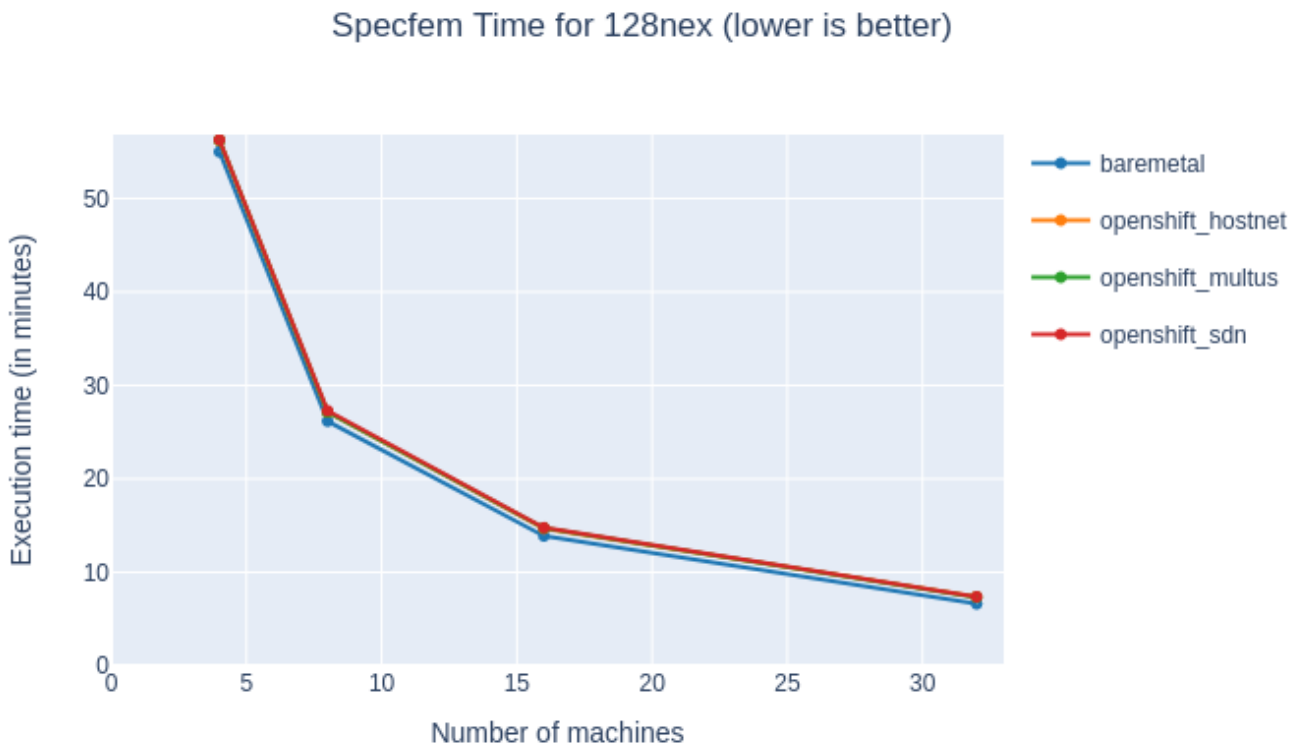
Number of machines, process per machine and threads: Specfem is able to efficiently exploit shared-memory cores with the help of OpenMP threads. We chose to use one or two processes per machine (OpenMPI slots parameter), with either four threads (if one process) or two threads (if two processes) so that each of the node's physical cores runs one-and-only-one

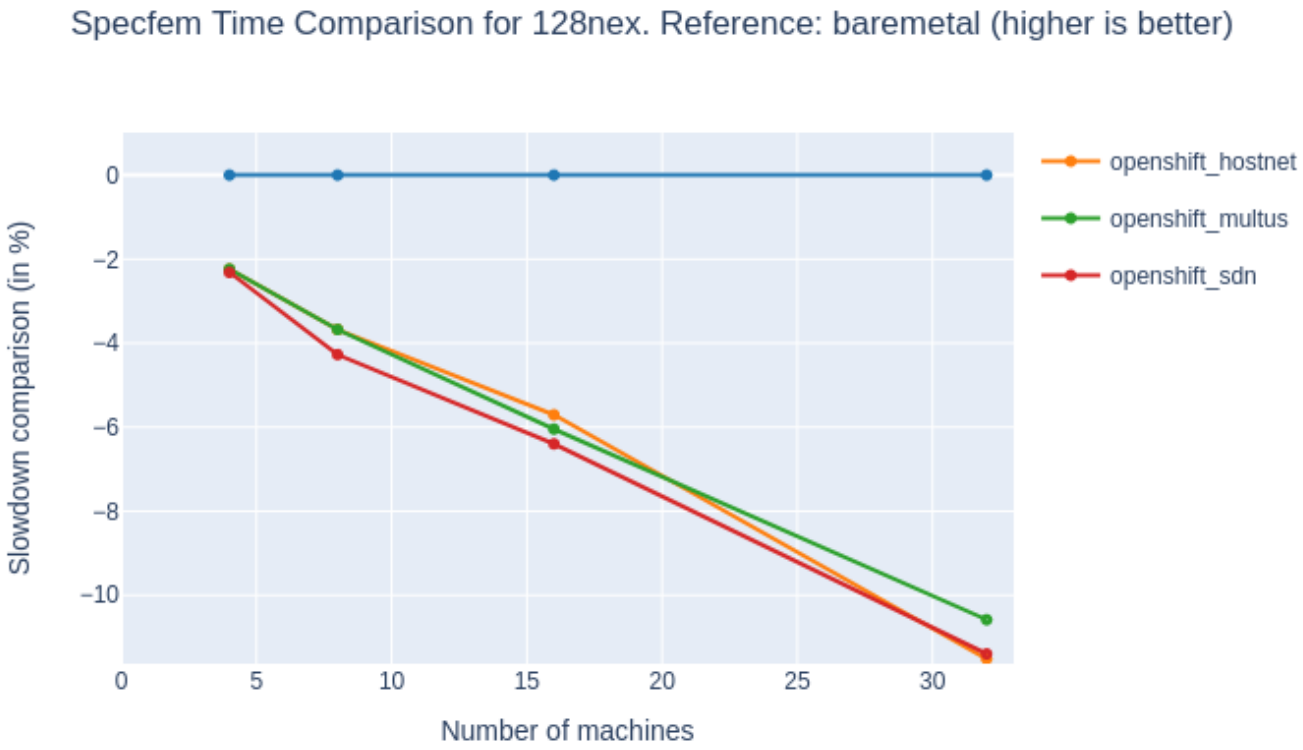
thread.

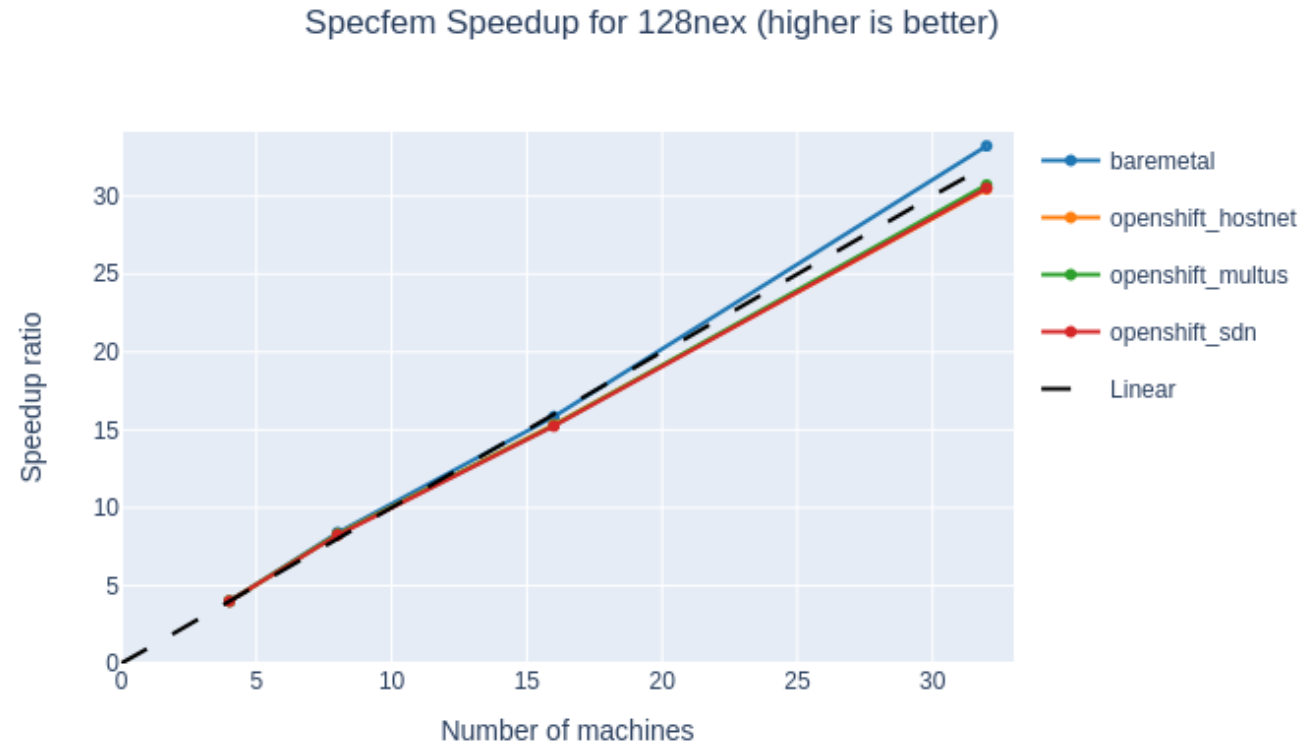
Note that by default, OpenMPI binds each process it spawns to a single CPU core, which prevents any speedup when using multiple threads. We passed the -bind-to none option to mpirun to prevent this CPU pinning.

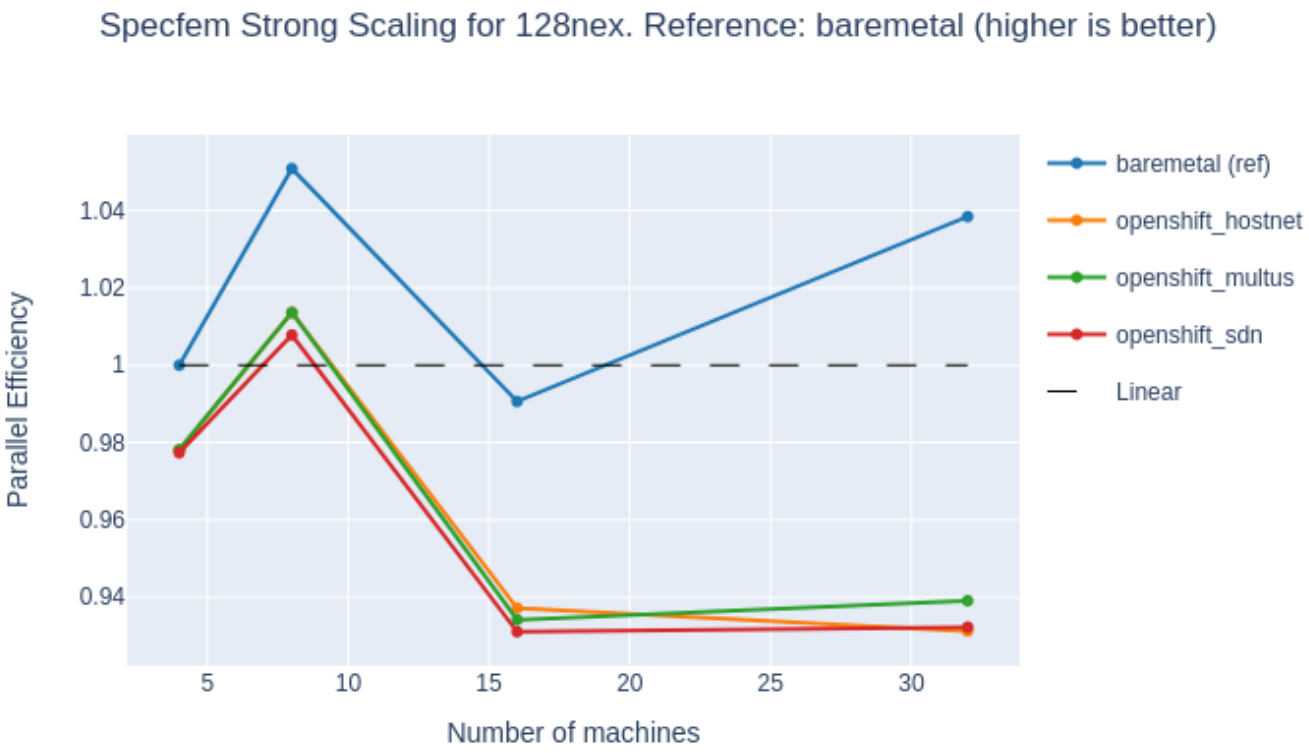
Strong scaling: running SPECFEM3D_Globe with 128 NEX

The results below present Specfem benchmarks with a fixed problem size (128 NEX), running on 4 to 32 processes.




Specfem Time Comparison for 128nex. Reference: baremetal (higher is better)
Slowdown comparison (in %)
0
−2
−4
−6
−8
−10
0
5
10
15
20
25
30
Number of machines
openshift_hostnet
openshift_multus
openshift_sdn


Specfem Speedup for 128nex (higher is better)
Speedup ratio
0
5
10
15
20
25
30
0
5
10
15
20
25
30
Number of machines
baremetal
openshift_hostnet
openshift_multus
openshift_sdn
Linear

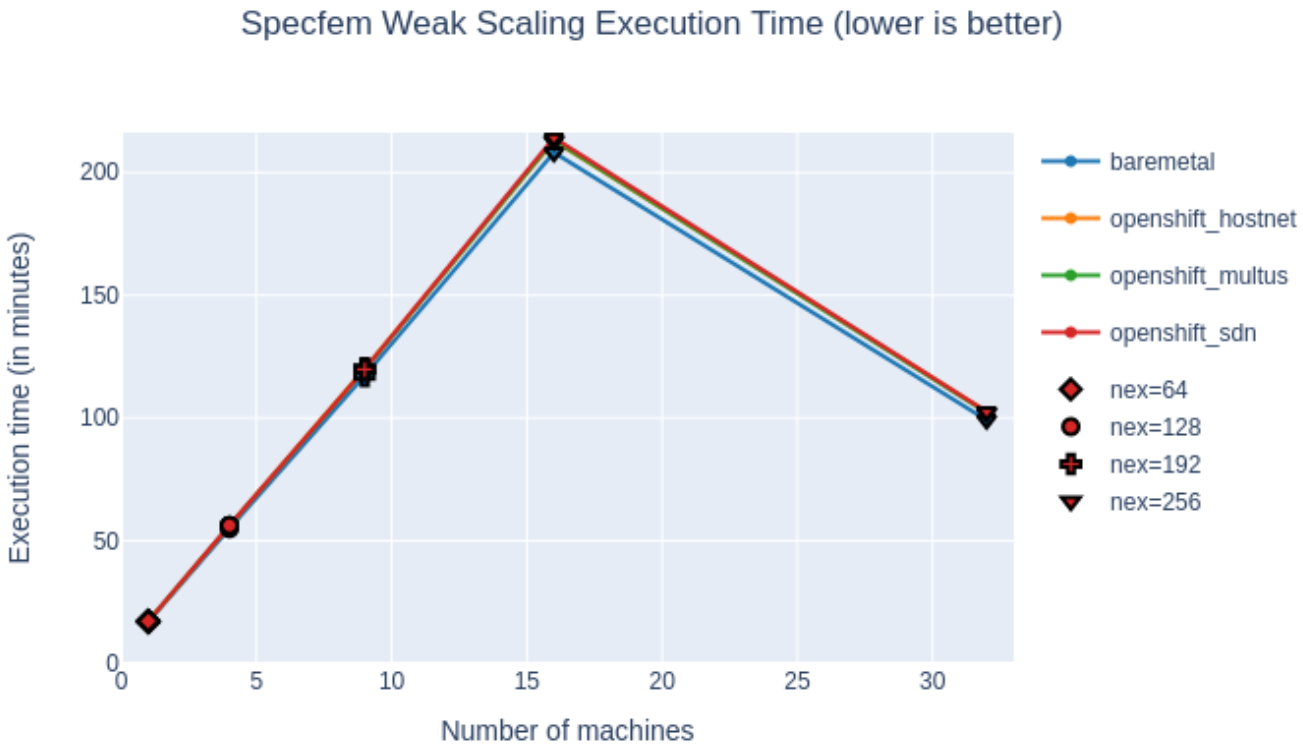


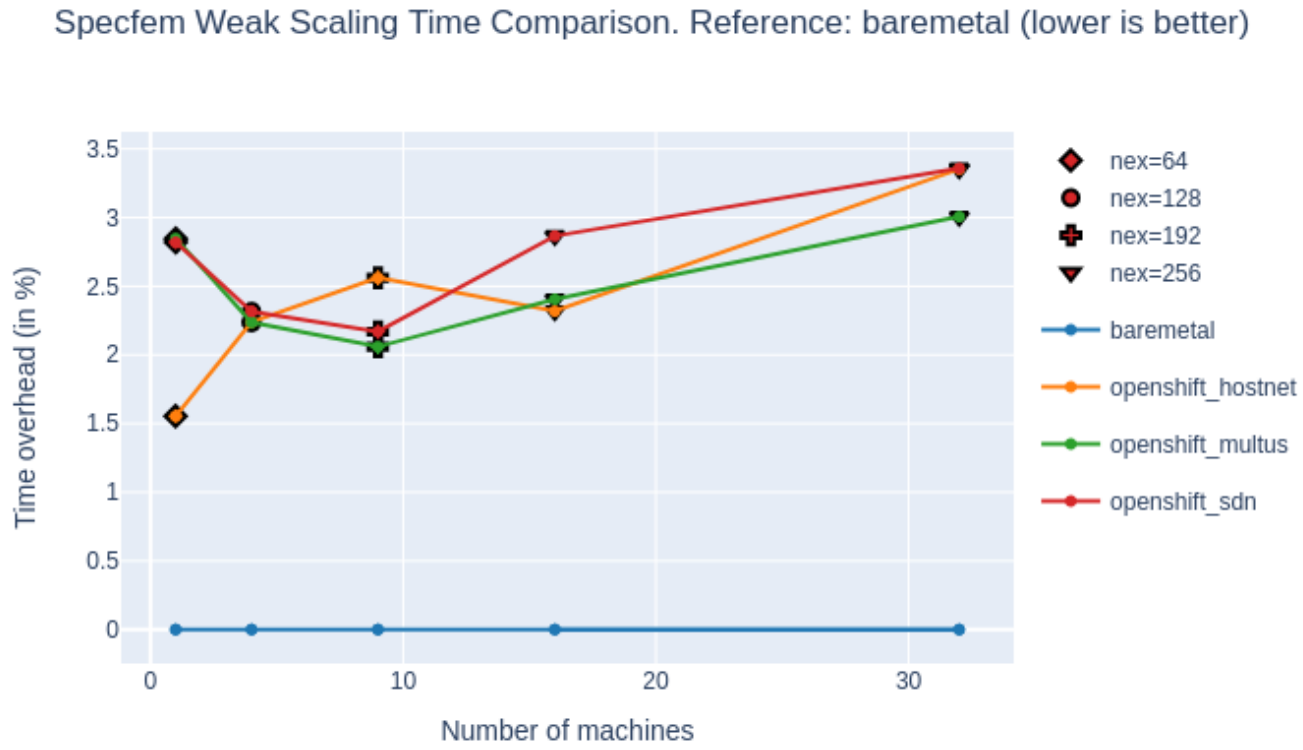
This problem took 55 minutes to execute on four machines, and only 8 minutes on 32 machines (see the **time** graph). For four machines, the slowdown of OpenShift was only 2%, and it grew up to 12% when OpenShift overhead took over the extra computing power on 32 nodes (see the **comparison** graph). But on the **strong scaling** graph, we can see that the efficiency of OpenShift remained very good, not far below the bare-metal reference and the linear time.

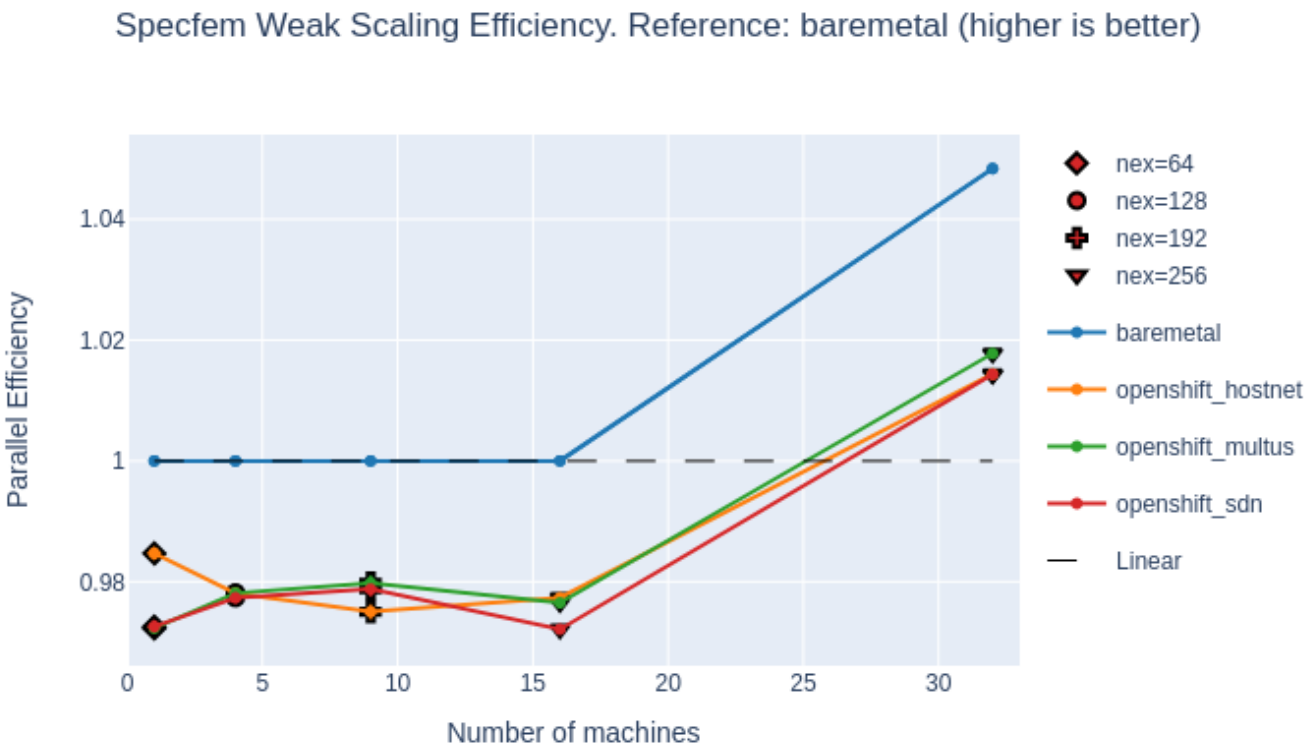
We can also note that Specfem performance is virtually independent of the network types.

Weak Scaling: Running SPECFEM3D_Globe with multiple problem sizes

The graphs below present the results of Specfem benchmark with multiple problem sizes (64 to 256 NEX), running on 4 to 32 processes.


Specfem Weak Scaling Execution Time (lower is better)
Execution time (in minutes)
Number of machines
0
50
100
150
200
0
5
10
15
20
25
30
baremetal
openshift_hostnet
openshift_multus
openshift_sdn
nex=64
nex=128
nex=192
nex=256


Specfem Weak Scaling Time Comparison. Reference: baremetal (lower is better)
Time overhead (in %)
Number of machines
0
0.5
1
1.5
2
2.5
3
3.5
0
10
20
30
nex=64
nex=128
nex=192
nex=256
baremetal
openshift_hostnet
openshift_multus
openshift_sdn



We can see that when we feed bigger problems to the cluster, we get back very close to the bare-metal reference, with less than 3.5% of overhead for most of the runs. For problems big enough (192 NEX and more), this applies up to 32 nodes, including the super-linear efficiency.

Conclusions

Overall our tests suggest that when running these high-performance scientific workloads on OpenShift, there is very low CPU overhead relative to bare-metal, as we have come to expect from containers. Additionally, when using Multus to enable a secondary network, there is very little impact on the network performance. By isolating infrastructure pods to specific nodes, we were also able to decrease variability and slightly increase the average performance of our workloads. Because we saw low overhead up to 32 nodes we are hopeful that workloads like these would continue to scale well to higher node counts. In this PoC testing, we were able to get performance within a few percent of bare-metal with two scientific HPC workloads while taking advantage of some of the benefits of containers and OpenShift.

We believe that Linux containers and container orchestration engines, most notably Kubernetes,

are well positioned to power future software applications spanning multiple industries. Red Hat has embarked on a mission to enable some of the most critical workloads like machine learning, deep learning, artificial intelligence, big data analytics, high-performance computing, and telecommunications, on Red Hat OpenShift. By developing the necessary performance and latency sensitive application platform features of OpenShift, the PSAP team is supporting this mission across multiple footprints (public, private and hybrid cloud), industries and application types.

About the authors

More like this

Blog post

Red Hat and Sylva unify the future for telco cloud

Blog post

NetApp's certified OpenShift operator for Trident

Original podcast

Crack the Cloud_Open | Command Line Heroes

Original podcast

Edge computing covered and diced | Technically Speaking

Browse by channel

Explore all channels →

Automation

The latest on IT automation for tech, teams, and environments

Artificial intelligence

Updates on the platforms that free customers to run AI workloads anywhere

Open hybrid cloud

Explore how we build a more flexible future with hybrid cloud

Security

The latest on how we reduce risks across environments and technologies

Edge computing

Updates on the platforms that simplify operations at the edge

Infrastructure

The latest on the world's leading enterprise Linux platform

Applications

Inside our solutions to the toughest application challenges

Virtualization

The future of enterprise virtualization for your workloads on-premise or across clouds

LinkedIn
YouTube
Facebook
X

Platforms

Red Hat AI
Red Hat Enterprise Linux
Red Hat OpenShift
Red Hat Ansible Automation Platform
See all products

Tools

Training and certification
My account
Customer support
Developer resources
Find a partner
Red Hat Ecosystem Catalog
Documentation

Try, buy, & sell

Product trial center
Red Hat Store
Buy online (Japan)
Console

Communicate

Contact sales
Contact customer service
Contact training
Social

About Red Hat

Red Hat is an open hybrid cloud technology leader, delivering a consistent, comprehensive foundation for transformative IT and artificial intelligence (AI) applications in the enterprise. As a trusted adviser to the Fortune 500, Red Hat offers cloud, developer, Linux, automation, and application platform technologies, as well as award-winning services.

- - - - - - - - - - - - - - -

Our company

How we work

Customer success stories

Analyst relations

Newsroom

Open source commitments

Our social impact

Jobs

Change page language

English

About Red Hat
Jobs
Events
Locations
Contact Red Hat
Red Hat Blog
Inclusion at Red Hat
Cool Stuff Store
Red Hat Summit