Best Practices for HPC Workloads on Public Cloud Platforms

A guide for computational scientists to use public cloud for HPC workloads

Robert Walkup, Seetharami R. Seelam, Sophia Wen
IBM T. J. Watson Research Center
Yorktown Heights, NY 10514, USA
{walkup, sseelam, hfwen}@us.ibm.com

ABSTRACT

HPC (high performance computing) applications come with a variety of requirements for computation, communication, and storage; and many of these requirements can be met with commodity technology available in public clouds. In this article, we report on results for several well-known HPC applications on IBM public cloud, and we describe best practices for running such applications on cloud systems in general. Our results show that public clouds are not only ready for HPC workloads, but they can provide performance comparable to, and in some cases better than, current supercomputers.

CCS CONCEPTS

• High Performance Computing • Cloud • Distributed computing

KEYWORDS

HPC, Cloud, distributed computing, MPI

ACM Reference format:

Robert Walkup, Seetharami R. Seelam, and Sophia Wen. 2022. Best Practices for HPC Workloads on Public Cloud Platforms: A guide for computational scientists to use public cloud for HPC workloads. In *Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE'22), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA. 7 pages. https://doi.org/10.1145/3489525.3511693

1 Introduction

The widespread availability of public cloud systems is having an impact on many realms of computing, including traditional High-Performance Computing (HPC) [1-3]. In this article, we report on results for several well-known HPC applications on IBM public cloud, and we describe best practices for running such applications on cloud systems in general. We discuss some of the key factors that HPC cloud users need to consider, including the selection of basic building blocks or virtual-machine instances. Measurements on cloud systems provide insight into how to get the most benefit from compute and network resources, including how to manage multiple network interfaces, and what to do about hyper-threading. While we report results from IBM Cloud, our observations are applicable to public clouds in general. We take the point of view of a traditional computational scientist and address the challenges of public cloud adoption for their HPC workloads. We will use a broad collection of HPC workloads that vary in scale, with different compute, memory, network, and scalability characteristics to describe the best practices.

ICPE '22, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9143-6/22/04...$15.00
DOI: https://doi.org/10.1145/3489525.3511693

2 Use a single large instance or many small ones?

One of the first choices a cloud user is faced with is selection of an "instance type" to be used as a building block, analogous to a "node" in traditional HPC clusters. Most public clouds offer a wide variety of instance types, with varying amounts of memory, CPU resources, some specified limits on network connectivity, and a collection of storage and file system options. In traditional HPC data centers, typically users have a fixed node configuration, so the application is tuned to best use the resources of the node. In contrast, cloud provides a large collection of node configurations so the users can tweak the cloud to fit their application needs.

In this article we use generic names for virtual machines, where the number of virtual cpus follows after the letter "v", and the number of GBytes of memory follows the letter "m". For example, a virtual machine with 128 virtual cpus and 256 GB memory will be listed as v128m256.

The most cost-effective network solutions use Ethernet with TCP protocol, which comes with significant latency for communication between nodes or virtual machines, and with bandwidth limits in order to provide some guaranteed level of service. Current MPI (message-passing interface) implementations provide very efficient communication between processes on the same node or virtual-machine via shared memory. In fact, the latency for shared memory communication can be as much as ~100x lower than Ethernet with TCP protocol.

An example of this is illustrated in Figure 1, which shows the measured MPI latency, defined as the average value of half the round-trip time to exchange messages using MPI_Send followed by MPI_Recv, on a current cloud system. Shared memory is used to communicate between two processes on the same virtual machine, while TCP protocol and Ethernet, with virtio-net as the network virtualization software, is used to communicate between processes on different virtual machines. The data below was measured using mpich-3.4.2 built with the "ch4:ucx" device. Similar trends are observed with other MPI implementations. The network virtualization method is a key factor. Measurements in this article were made with virtio-net, which provides flexible control of network bandwidth, but adds significant latency.

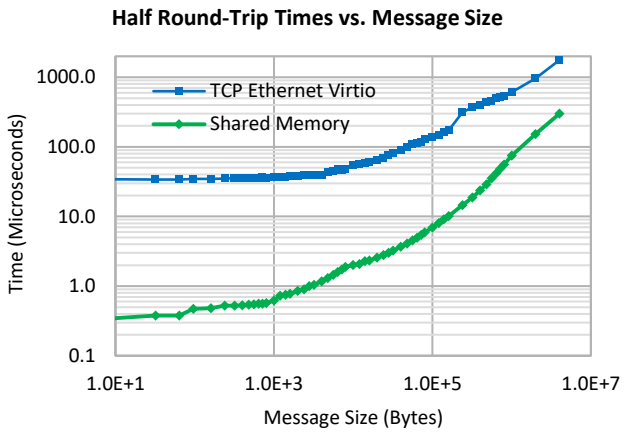

Figure 1. MPI latency is shown as a function of message size, using either shared memory (lower curve) or TCP protocol over Ethernet (upper curve) on a public cloud.

Latency sensitive applications that can fit in a large shared-memory system are best deployed on a single virtual machine with many cores and plenty of main memory. For example, on IBM public cloud there is a family of instance types with 128 virtual cpus (64 processor cores) and memory configurations of 256, 512, or 1024 GB. Other clouds have similar instance types. This is sufficient for many HPC use cases, including computational fluid dynamics with grids of order 10^7 cells or less and metropolitan area weather forecasts.

We investigated the performance of the weather model WRF [4,21] used by the Deep Thunder team in IBM [5]. Their test case used a sequence of four nested grids, where each grid contained 100x100 points in the lateral dimensions with resolutions of 18 km, 6 km, 2 km, and 0.667 km, and 51 vertical levels, and with history file outputs every 10 forecast minutes. The objective of this test case was to complete a 72-hour forecast in less than 3 wall-clock hours. Table 1 shows the measured run times on IBM cloud using either a single virtual machine with 64 cores (128 vcpus), 256 GB memory, with communication via shared memory (type v128m256), or a set of 12 virtual machines with 8 cores (16 vcpus) each (type v16m64) connected via Ethernet. The shared-memory configuration proved to be faster, even though there are significantly fewer cores.

Cost is another consideration. On our test cloud platform, cost was almost identical for configurations with the same total number of virtual cpus and the same aggregate amount of memory. For this local-area WRF case, it is more cost efficient to choose the large shared-memory instance type, because that provides better performance using fewer virtual cpus.

Table 1. Run times are shown for a 72-hour metropolitan area weather forecast using WRF on the cloud.

Instance type	#VMs	cores	Messaging	Run time
v16m64	12	96	Ethernet TCP	2.72 hrs
v128m256	1	64	shared mem	2.40 hrs

WRF is mainly Fortran code, and best performance was obtained using the Intel Fortran compiler, with options set for current generation Intel CPUs. Using a single large virtual machine in a cloud environment is essentially equivalent to using a dedicated x86 server with an equivalent configuration of cores and memory.

We also examined computational fluid dynamics (CFD) using OpenFOAM [6,22] and the motorbike steady-state flow tutorial. The key parameter in CFD simulations is the number of grid cells. Problems with up to ~50 M grid cells can fit within the memory available on a single large virtual machine. A comparison of run times for the motorbike tutorial is shown in the table 2, using 64 cores on the cloud from either (a) a single virtual machine of type v128m256, or (b) four virtual machines of type v32m128 connected via Ethernet. For grids with ~1 M cells and a total of 64 MPI ranks (one rank per physical core), communication is very important, and the single virtual machine is significantly faster. Communication via shared memory remains advantageous for larger grids, but the timing difference becomes less significant as the simulation becomes more compute bound. Again, cost is basically the same for configurations with the same number of virtual cpus and the same amount of memory, and so the large shared-memory system is more cost effective.

When the number of grid cells is ~10 M or greater, it may be desirable to scale out to larger core counts in order to reduce the time to solution. On Cloud systems that use TCP with virtio-net, it is best to limit scaling and keep the number of grid cells per core larger than ~10^5, otherwise latency in the Ethernet network becomes a substantial bottleneck. Cloud systems that provide lower latency networks can scale out further.

Table 2. Elapsed times for the simpleFoam motorbike tutorial are shown for different grid sizes using either a single virtual machine of type v128m256 or four instances of type v32m128 connected by Ethernet.

#Cells	v128m256	Four of v32m128
0.91 M	48.7 sec	88.1 sec
2.63 M	155.3 sec	195.4 sec
5.41 M	351.0 sec	420.6 sec

3 Managing Hyper-threading and using physical cores

Cloud instances usually expose virtual cpus (vcpus), which map to hyper-threads on the underlying server. Many HPC applications gain little if any benefit from hyper-threading, and so it is common practice to bind applications to just one hyper-thread per physical core or to turn it off at the guest virtual machine [20]. With virtio-net for network virtualization and TCP protocol [11] there is another reason to use just one hyper-thread per core. Messages are copied from the guest operating system in the user's VM to the operating system on the host, via "vhost" threads [7]. The "vhost" threads are active within the host operating system. In addition, there are kernel threads that service interrupts, both on the host operating system, and for the guest virtual machine. When network traffic is heavy, these threads need free slots for execution. If the user has tied up all the hyper-threads with computational work, the kernel threads will contend with user threads for execution slots, and messaging performance will suffer. We have conducted detailed experiments showing that this mechanism is a major factor that limits scaling and contributes to performance variability on cloud systems that use virtio-net.

Sources of performance variability can be identified using simple tests[28,29,30], including measurements of the memory bandwidth and core frequencies for every virtual machine. We have observed that virtual machines with 16 vcpus (8 cores) can be located on the underlying server hardware in multiple ways : some VMs fit inside one numa domain while others span two numa domains. In addition, there may be more than one VM allocated to the same numa domain, resulting in contention for memory bandwidth. This form of heterogeneity can result in job to job performance variations because different VMs of the same type have different performance characteristics, and the assignment of MPI ranks to VMs can vary from job to job. It is useful to make quantitative measurements of timing variability, using an artificial parallel application with carefully calibrated units of computation work combined with a simple communication pattern following completion of each work unit [28]. With such tests, one can map out the parallel efficiency as a function of the frequency and volume of communication between the ranks. Additional insight can be obtained by visualization of MPI time lines, which can show the delays in completion of the work unit caused by CPU cycle-stealing by competing processes. TCP communication results in significant CPU activity. One can manage activity by the guest operating system in the VM to some degree by carefully managing IRQ affinity assignments. However, with virtio-net as the network virtualization layer, there is also significant CPU activity in the host operating system, which is outside the realm of control by a cloud user. Our measurements show that CPU hot-spots resulting from message processing by the host OS constitutes a major source of timing variability and can limit scaling on cloud systems.

A dramatic example of this contention for CPU resources is provided by a high-resolution weather forecasting case, using a mix of MPI and OpenMP as described in a later section. When OpenMP threads are limited to one thread per physical core, the average time per time-step was ~0.28 seconds at 1536 cores. Doubling the number of OpenMP threads so that every hyper-thread was busy with OpenMP work resulted in an average time per step of ~3.2 seconds, or more than a 10x slow-down. Detailed performance analysis showed that the excess time is caused by delays in messaging. We have observed similar effects in many HPC applications. This problem can be avoided by either of two methods: (1) bind the application's threads to just one hyper-thread per physical core, or (2) disable hyper-threading inside the user's VM [20]. In both cases, all hyper-threads remain available to the host operating system, and so the "vhost" threads continue to have free slots for execution. Our measurements indicate that these two options provide equivalent performance. Normally we prefer to keep hyper-threading enabled inside the VM and take care to bind one user thread or process to each physical core, leaving half the vcpus free. The issue of host kernel threads contending with application threads for CPU resources does not arise when using only shared memory for communication, or when using methods that directly expose the network devices to the virtual machine.

4 Selecting instances and network interfaces for scaling out to large core counts

When scaling to large numbers of virtual machines and cores, the selection of an appropriate instance type depends on the available network options, which in turn depend on the number of virtual cpus in the instance. Our reference public cloud system uses virtio-net for network virtualization, where each virtual Ethernet interface can provide at most 16 Gbps bandwidth (single flow, unidirectional). Other public clouds follow similar patterns [23,24,25,26]. Depending on the number of virtual cpus in the instance, users can create additional Ethernet interfaces and obtain up to 80 Gbps bandwidth [26]. However, managing multiple interfaces adds complexity, and it is often not possible to realize the full potential benefit. Many HPC applications are rather loosely synchronous and have medium to small messages that do not benefit from striping across multiple adapters. In such cases, there is typically not much contention for shared adapter resources, and communication performance is not likely to improve with the addition of multiple Ethernet interfaces.

Our experience has been that those instances with ~16 vcpus and one or two Ethernet interfaces provide a good starting choice when scaling beyond the number of cores available in a single large VM. With 16 vcpus, there are 8 physical cores, which is normally enough to get some benefit from communication via shared memory, and the core count is small enough to ensure good network bandwidth per core for the messages that must go over Ethernet. These network considerations are unique to cloud systems, which offer many different instance types and network options. On traditional HPC clusters, users have full access to the hardware on each server, and all servers normally have identical configurations.

Many cloud providers use Ethernet with TCP protocol for networking, and there are tunable parameters that affect performance. We have noticed improved messaging performance on the cloud by setting the MTU to 9000 for the Ethernet interfaces,

and by using the options shown in the table 3, set in each virtual machine. Some of these parameters, such as the maximum number of socket connections, may need adjustment, depending on the scale of the parallel job. Similar TCP tuning parameters are suggested in the OpenMPI frequently asked questions [8], and in the performance tuning guides for Mellanox network adapters. Our experience has been that the increased TCP memory buffers reduce packet drops and re-transmits when the network load is heavy. These choices tend to help performance using any number of TCP interfaces.

The choice of MPI implementation can also make a difference, particularly when it comes to support for multiple Ethernet interfaces. The approach that we prefer uses MPI built on top of UCX [9]. UCX provides a convenient environment variable, UCX_NET_DEVICES, that can be used to select one or more network interfaces. The UCX layer is automatically included in recent releases of MPICH. We have tested mpich-3.4.2 configured with the built-in "ch4:ucx" device. One can also incorporate UCX into OpenMPI, and we have tested openmpi-4.1.1 built with ucx-1.11.1. Both MPI implementations can provide good performance and flexible control over multiple interfaces. The basic point-to-point messaging behavior is very similar in these implementations, because overhead is mostly in the TCP software layers, and not in MPI or UCX. Differences tend to arise in some of the collective communication operations, and so the best choice of MPI implementation is application dependent.

Table 3. TCP tuning parameters

net.ipv4.tcp_low_latency	1
net.ipv4.tcp_adv_win_scale	1
net.ipv4.tcp_timestamps	0
net.ipv4.tcp_sack	1
net.core.netdev_max_backlog	250000
net.core.rmem_max	16777216
net.core.wmem_max	16777216
net.core.rmem_default	16777216
net.core.wmem_default	16777216
net.core.optmem_max	4194304
net.ipv4.tcp_rmem	4096 87380 16777216
net.ipv4.tcp_wmem	4096 65536 16777216
net.core.netdev_budget	1200
net.core.somaxconn	2048

The exchange bandwidth between two virtual machines is shown in the figure 2, as a function of message size, using one or two Ethernet interfaces. These measurements were made using openmpi-4.1.1 built with ucx-1.11.1, with 8 MPI ranks per VM, and v32m64 instances (32 vcpus, 64 GB memory) on the cloud. To use just one of the two Ethernet interfaces, we set UCX_NET_DEVICES=eth0 for all MPI ranks, and the exchange bandwidth reaches a plateau at ~4 GB/sec (32 Gbps) for large messages (curve "ucx-1" in the Figure 2).

By setting UCX_NET_DEVICES=eth0,eth1 for all MPI ranks, large messages are striped across both virtual adapters, and the exchange bandwidth reaches a plateau at ~8 GB/sec (64 Gbps) (curve "ucx-striped"). Striping is effective only for rather large messages. For many purposes it is better to launch MPI jobs with a helper script that sets UCX_NET_DEVICES to eth0 for half of the ranks on each VM, and to eth1 for the other half. This results in the curve labeled "ucx-2", which reaches a plateau at ~8 GB/sec and provides better exchange bandwidth for medium sized messages. The same mechanism, setting UCX_NET_DEVICES, applies to MPICH versions that are configured with the "ch4:ucx" device.

With OpenMPI, one can alternatively select the built-in "ob1" point-to-point messaging layer (--mca pml ob1). One can specify a single Ethernet interface (--mca btl_tcp_if_include eth0), or request striping over two interfaces (--mca btl_tcp_if_include eth0,eth1). The performance characteristics are similar with either "ob1" or "ucx" for the point-to-point messaging layer; and again, striping is effective only for large messages.

For an example where multiple Ethernet interfaces can help, consider a large parallel 3D FFT. We used the P3DFFT package for these measurements [10], with an MPI-only build (no OpenMP). The P3DFFT package uses a 2D pencil decomposition, where the main communication pattern is MPI_Alltoallv or MPI_Alltoall on process rows or columns.

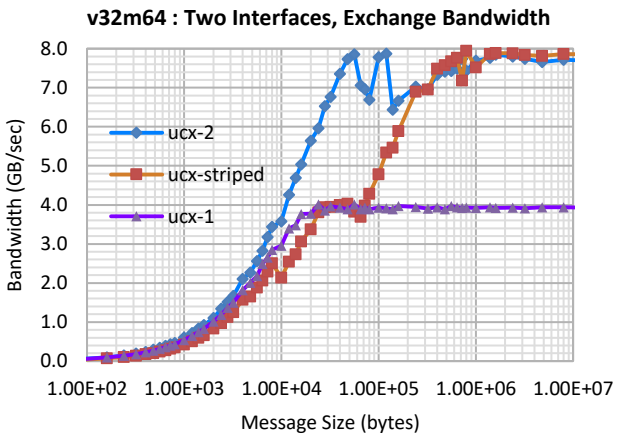

Figure 2. The measured exchange bandwidth between two virtual machines (8 MPI ranks per VM) is shown as a function of message size, using either one Ethernet interface (ucx-1), striping across two Ethernet interfaces (ucx-striped), or assigning half the ranks to use one interface while the other half use a different interface (ucx-2).

The entire data set is transposed multiple times in order to successively collect each dimension for 1D FFT transforms, keeping the other two dimensions distributed. The transposition method relies on network bandwidth when using large data sets. The data shown in the figure 3 is for a 3D FFT on a grid with 3072^3 points, distributed over 1536 MPI ranks, with one rank per physical core. We used 192 VMs of type v16m32 for these measurements, configured with two Ethernet interfaces, thus enabling the maximum available network bandwidth per core. We used openmpi-4.1.1 built on top of ucx-1.11.1, setting the environment variable UCX_NET_DEVICES to assign Ethernet interfaces. For simplicity, we chose to analyze one of the sample programs, test_rand_f.x, which uses random numbers on the 3D grid. We instrumented the code to roughly partition time into "messaging" and "computation" components. The figure 3 shows timing data for 10 iterations of the forward and backward 3D FFT, using either just one Ethernet interface (the left-most bar), or using a helper script set UCX_NET_DEVICES to eth0 for even numbered ranks and to eth1 for odd numbered ranks (the bar in the middle), or we request striping over both interfaces by setting UCX_NET_DEVICES=eth0,eth1 (the right-most bar). At this scale, message sizes in MPI_Alltoallv were ~3-5 MBytes, performance is limited by network bandwidth, and the messages are large enough to benefit from striping, as shown in our earlier Figure 2. Using two Ethernet interfaces clearly reduces the time associated with messaging, but not by a factor of two, and the two different methods for using both network interfaces provide roughly equivalent performance.

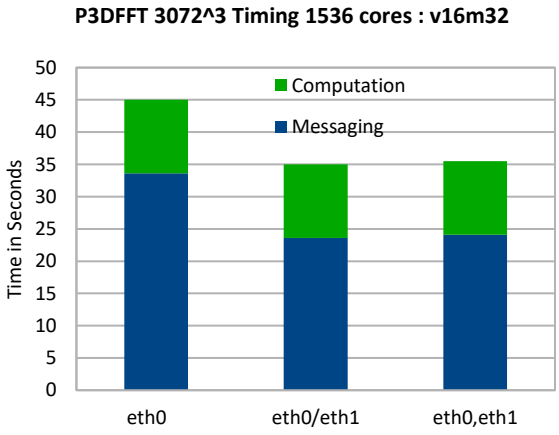

Figure 3. The effect of using two Ethernet interfaces is shown for 10 iterations of forward and backward 3D FFTs on a grid with 3072^3 points distributed over 1536 MPI ranks on the cloud.

We have examined the impact of enabling two Ethernet interfaces on a substantial number of HPC applications, and in most cases the performance improvement is minimal. There are exceptions, for example when performance depends critically on collective communication with quite large messages, as in the 3D FFT example.

5 HPC workloads on Cloud vs a Supercomputer

A high-resolution weather forecast model provides an interesting test case. We made extensive measurements using WRF version 4.1.5 and a grid covering the continental US at 2.5 km resolution. WRF is an explicit time-stepping code, where the main communication pattern is boundary exchange with nearest neighbors on a 2D Cartesian grid. There are no globally synchronizing functions in the main time-step loop. WRF uses a very effective method to pack and unpack message buffers. This results in message sizes of ~10 KB to ~500 KB in the relevant parts of the scaling curve. As a result, this high-resolution weather model is not very sensitive to latency in the network, and it is well suited for cost effective Ethernet networks using TCP protocol on public cloud systems. One can achieve simulation speeds of ~50 forecast hours per elapsed hour using ~1500 cores on the cloud, with the standard single-interface Ethernet configuration, as shown in the scaling curve in Figure 4. We used v16m32 instances (16 vcpus, 32 GB memory), and a mix of MPI plus OpenMP: four MPI ranks per VM, with two OpenMP threads, taking care to bind just one OpenMP thread per physical core.

The scaling curve for the same test case on the Summit supercomputer at Oak Ridge National Laboratory is shown in Figure 4 for comparison.

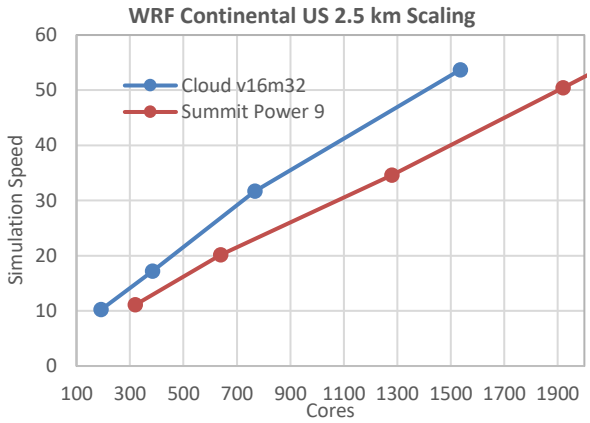

Figure 4. Scaling curves are shown for WRF 4.1.5 using a grid covering the continental US at 2.5 km resolution. The upper curve is for the cloud using v16m32 instance types with a single 16 Gbps Ethernet interface, and the lower curve shows data from the Summit supercomputing system at Oak Ridge.

The Summit system has two EDR Infiniband adapters per node and includes a very low noise environment to ensure excellent scaling. This WRF model used only CPUs (not GPUs), and the current x86 processors on the cloud provide higher performance per core compared to the IBM Power 9 processors on Summit. Our measurements on the cloud show no improvement for this WRF test case when enabling two Ethernet interfaces. In fact, best performance was with OpenMPI and the "ob1" point-to-point messaging layer, using just one Ethernet interface. This is

consistent with expectation because the messages are too small to benefit from striping, and WRF has plenty of load imbalance, which tends to reduce contention for access to shared network adapters. As mentioned earlier in this article, it is essential to bind just one OpenMP thread to each physical core on the cloud system. If OpenMP threads are bound to every virtual cpu, there will be contention for execution slots between user threads and kernel threads in the guest and host operating systems, resulting in communication delays and ~10x lower performance at scale. This type of contention occurs with virtio-net as the network virtualization layer. Different behavior is expected for different methods of network virtualization (such as SR-IOV).

SNAP is a well-known HPC benchmark used by U.S. Dept. of Energy Labs as a proxy for solution of deterministic Boltzmann transport models for neutral particles. We investigated the SNAP benchmark used to evaluate "Commodity Technology Systems" [12]. The test case has constant work per MPI rank (weak scaling), using a local domain with dimensions of 640x4x4 grid cells. This is a very communication intensive benchmark, because most of the grid cells are on domain boundaries, so the ratio of communication to local computation is relatively high. Measurements for SNAP are shown in figure 5, comparing a public cloud system with the Summit supercomputing system at Oak Ridge, using only the CPUs for computation (not GPUs).

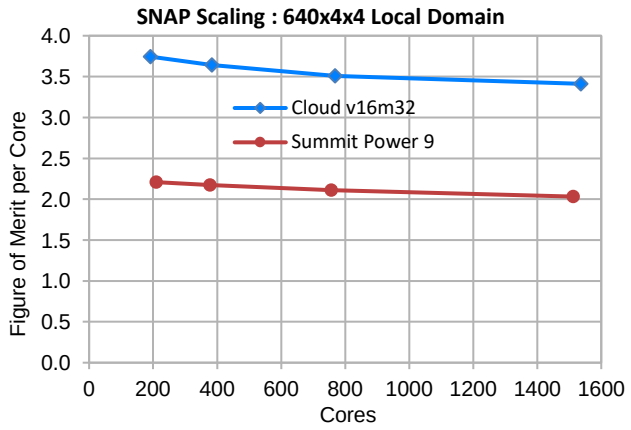

Figure 5. Weak-scaling curves are shown for the SNAP benchmark, using a local domain with 640x4x4 grid cells per MPI rank. The upper curve is for the cloud using v16m32 instance types with two 16 Gbps Ethernet interfaces, and the lower curve shows data from the Summit supercomputing system at Oak Ridge.

The cloud system was using virtual machines with 16 vcpus (8 cores) and two ethernet interfaces per VM. The Summit system has 42 cores and two EDR Infiniband adapters per node. Figure 5 shows the performance metric (figure of merit) per MPI rank, so linear scaling would appear as a flat line. This is another case where the performance per core is higher on the public cloud system. SNAP has one main "hot" routine, and the x86 processors used by the cloud system benefit more from SIMD instructions, and thus provide more performance per core. The scaling behavior is very similar on both systems: there is a slight reduction in parallel efficiency as one scales out beyond ~1000 cores. On the public cloud system, using two Ethernet interfaces per VM resulted in a ~10-15% performance improvement, relative to using a single Ethernet interface.

We have investigated other well-known HPC applications including the molecular dynamics package LAMMPS [13]. The performance characteristics of LAMMPS depend on the force field, and we selected solid copper with embedded-atom forces for this study. On the public cloud system, we limited our measurements to CPUs, but on the Summit supercomputing system, we made measurements using (1) just the Power 9 CPUs, or (2) the NVIDIA V100 GPUs enabled with the Kokkos package [14]. In order to compare CPU and GPU systems, we show performance versus the number of compute units, where a compute unit is either a CPU socket, or a GPU device, and we used weak scaling, with a constant number of copper atoms per compute unit (~4*10^5 atoms). On Summit, each CPU socket has 21 Power 9 cores, and there are six V100 GPUs and two EDR Infiniband adapters per node. On the public cloud system, we used virtual machines with 16 vcpus (8 cores) and just one 16 Gbps Ethernet interface, and for purposes of comparison, we assume that each "compute unit" on the cloud contains 24 cores, as expected for the underlying x86 servers.

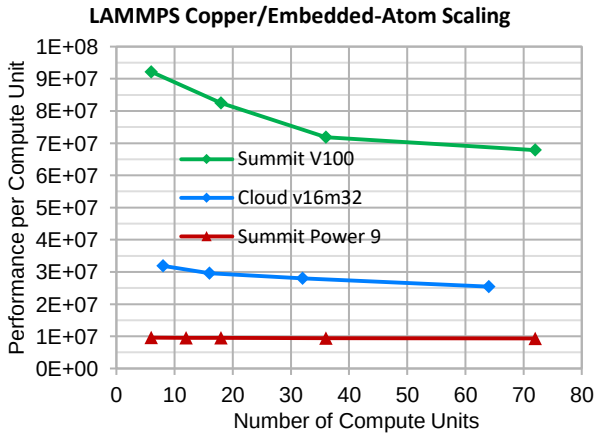

Figure 6. Weak-scaling curves are shown for LAMMPS using embedded-atom forces for solid copper. The x-axis indicates the number of compute units, which represent either a GPU or a CPU socket. The V100 GPUs on Summit, top curve, provide the best performance per compute unit. The two lower curves indicate performance using CPUs on the cloud (middle curve) or on Summit (lower curve).

The measured performance per compute unit is shown in figure 6, with data extending to ~1500 cores for the CPU cases. Perfect linear scaling would appear as a flat line in the figure. With these parameters, the Summit system using Power 9 CPUs provides very close to the ideal scaling behavior. On the public cloud system, we used the Intel package [15], which exploits SIMD instructions available for the x86 processors. This results in better performance per core (and per compute unit) compared to the Power 9 processors on Summit. On the cloud system, performance per compute unit

decreases by a modest amount, indicating less efficient scaling. The NVIDIA V100 GPUs provide significantly more performance per compute unit, as shown in the top curve. However, scaling is less than ideal in the GPU case because it takes less time to do the computation per time step, and there is more overhead in the messaging layers. To get the most benefit from GPUs, a cloud system would need good support for messaging using RDMA from GPU memory.

6 Discussion

In this paper, we reported best practices for compute, memory and network intensive workloads with moderate requirements for storage in applications such as WRF checkpoints. Public clouds offer a collection of storage and file system technologies such as host attached storage, network attached block storage, NFS file systems and parallel HPC file systems. For I/O intensive HPC workloads, careful evaluation with the different storage and file systems in cloud will be necessary to leverage cloud economically and to achieve scalable performance.

7 Conclusion

Many cloud providers use Ethernet with TCP protocol for networking [16,17,18,19], which comes with high latency, whereas shared memory provides very efficient communication. As a result, it is often best to choose a single large virtual machine if the HPC workload fits. When scaling to large numbers of cores, a good starting choice is to use virtual machine instances with ~16 virtual cpus and one or two Ethernet interfaces. Multiple network interfaces offer marginal performance improvement when messages are relatively small and application behavior is loosely synchronous, but applications that rely on collective communication with large messages can benefit. With careful choices for managing the compute and networking resources, current public clouds can handle many HPC workloads, and provide performance comparable to, and sometimes better than, current supercomputing systems.

ACKNOWLEDGMENTS

Authors would like to thank Paul Mazzurana, Augie Mena, Greg Mewhinney, Suraksha Vidyarthi from IBM Cloud for their help and collaboration for experimentation on IBM Cloud [27].

REFERENCES

[1] Giulia Guidi, Marquita Ellis, Aydin Buluc, Katherine Yelick, David Culler, *10 Years Later: Cloud Computing is Closing the Performance Gap*, ICPE '21: Companion of the ACM/SPEC International Conference on Performance Engineering April 2021 Pages 41–48 https://doi.org/10.1145/3447545.3451183

[2] Constantinos Evangelinos and Chris Hill. 2008. Cloud computing for parallel scientific HPC applications: *Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2*. CCA-08, Vol. 2, 2.40 (2008), 2--34.

[3] Katherine Yelick, Susan Coghlan, Brent Draney, and Richard S. Canon. 2011. *The Magellan Report on Cloud Computing for Science*. US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Vol. 3.

[4] https://www.mmm.ucar.edu/weather-research-and-forecasting-model

[5] https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepthunder

[6] https://openfoam.org

[7] https://www.redhat.com/en/blog/introduction-virtio-networking-and-vhost-net

[8] https://www.open-mpi.org see the frequently asked questions

[9] https://openucx.org

[10] https://p3dfft.net

[11] Using Google Virtual NIC, https://cloud.google.com/compute/docs/networking/using-gvnic

[12] https://hpc.llnl.gov/cts-2-benchmarks

[13] https://www.lammps.org

[14] https://docs.lammps.org/Speed_kokkos.html

[15] https://docs.lammps.org/Speed_intel.html

[16] https://cloud.google.com/vpc/docs/create-use-multiple-interfaces#max-interfaces

[17] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

[18] https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-network-interface-vm?context=/azure/virtual-machines/context/context

[19] https://cloud.ibm.com/docs/vpc?topic=vpc-using-instance-vnics

[20] Seetharami Seelam, Disable Hyper-Threading in IBM Cloud https://www.ibm.com/cloud/blog/disable-hyper-threading-in-ibm-cloud

[21] Paul Mazzurana, Augie Mena, Robert Walkup, Weather Research and Forecasting Model Workload Evaluation on IBM Cloud, https://www.ibm.com/cloud/blog/weather-research-and-forecasting-model-workload-evaluation-on-ibm-cloud

[22] Robert Walkup, Greg Mewhinney, Running OpenFOAM on IBM Cloud, https://www.ibm.com/cloud/blog/running-openfoam-on-ibm-cloud

[23] AWS Elastic Network Interfaces: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

[24] GCE Multiple network interfaces: https://cloud.google.com/vpc/docs/create-use-multiple-interfaces

[25] Microsoft Azure: Multiple Network interface: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/multiple-nics

[26] Rei Odaira, Saju Mathew, Weiming Gu, Multiple Virtual Network Interfaces Maximizing Throughput in IBM Cloud VPC, https://www.ibm.com/cloud/blog/multiple-virtual-network-interfaces-maximizing-throughput-in-ibm-cloud-vpc

[27] Suraksha Vidyarth, IBM Spectrum LSF Is Now Available on IBM Cloud, https://www.ibm.com/cloud/blog/announcements/ibm-spectrum-lsf-is-now-available-on-ibm-cloud

[28] Robert Walkup, OS Noise tool -- Utility to characterize parallel application scaling issues caused by effects including OS noise. https://github.com/IBM/osnoise

[29] Robert Walkup, MPI Trace tool: https://github.com/IBM/mpitrace

[30] Seetharami Seelam, Liana Fong, Asser Tantawi, John Lewars, John Divirgilio, Kevin Gildea. Extreme scale computing: Modeling the impact of system noise in multicore clustered systems. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.