Running an HPC MPI Workload Using Intel oneAPI on IBM Cloud

October 2022

Authors:
I.S. Sutton
R. Walkup

Running an HPC MPI Workload Using Intel oneAPI on IBM Cloud

Introduction

As the HPC landscape shifts away from large specialized clusters and towards on-demand scalable cloud platforms, the providers of such platforms are met with the task of satisfying the customer's performance expectations. This task is particularly challenging for high performance computing applications, where the expectation is that performance is gated by some hardware property instead of being limited by inefficient software. In this document we analyze the performance of a scalable HPC workload that depends heavily on communication via the Message Passing Interface (MPI). We use Intel's MPI implementation and the Intel compilers provided in its oneAPI HPC Toolkit. Intel oneAPI is now included in the IBM Spectrum LSF on IBM Cloud offering via the operating system images used to provision the cluster nodes.

HPC on IBM Cloud with IBM Spectrum LSF

Our tests used IBM Cloud's Spectrum LSF offering, a cloud-based cluster solution using virtual machines (VMs) as the storage, management, and compute nodes. The cluster exists in a Virtual Private Cloud (VPC) network. A shared NFS file system is made available to all compute nodes using a cloud-provided storage volume managed by a storage node.[1] With the Spectrum LSF solution, each compute node has one Ethernet interface. We used virtual machines with 8 cores (16 virtual cpus) for the compute nodes because that configuration provides a good balance between computational capability and network performance. Larger VMs would provide less network bandwidth per core, and smaller VMs would result in bandwidth throttling of the Ethernet interface.

The Message Passing Interface and Intel MPI

The Message Passing Interface (MPI) is provided as a software library used for communication between different processes located either on the same virtual machine or on different virtual machines. MPI has been widely adapted by the HPC community and it plays a key role in achieving scalable performance across many nodes or VMs in a cluster. HPC users can choose from a number of MPI implementations including Intel MPI and open-source versions such as Open MPI and MPICH.
Communication between processes on the same VM is through shared memory, while in our case communication between processes on different VMs is over Ethernet with TCP protocol. The software overhead associated with TCP communication is large compared to the overhead within the MPI layer. As a result, one can expect similar performance characteristics for basic point-to-point communication over Ethernet with any of the available MPI implementations. However, performance differences can be significant for collective communication, where implementation details and algorithm choices can make a difference.

The compute nodes in IBM Cloud are built on top of Intel servers. The current generation uses primarily Cascade Lake CPUs. With Intel's oneAPI HPC Toolkit, highly optimized C, C++, and Fortran compilers are included along with Intel MPI. The oneAPI HPC Toolkit is particularly convenient for Fortran applications that use MPI modules. Intel's Fortran compiler generates very efficient SIMD code for Intel CPUs, and Intel MPI automatically includes the MPI module files for the Intel Fortran compiler. This combination is very convenient for ensuring efficient computation and

1 Although the Spectrum LSF offering provides a powerful clustered file-system (GPFS), an NFS file-system is often sufficient for HPC workloads such as the one considered here.

communication for HPC applications. Also, the "mpirun" command included with Intel MPI has built-in support
for the LSF scheduler. This makes building and running HPC applications on IBM Cloud very similar to using traditional HPC centers, thus easing the transition to Cloud environments.

SNAP

We chose SNAP as the communication-intensive HPC application to explore in this article. SNAP serves as a proxy for deterministic Boltzmann transport solvers[2,3], and it is used to evaluate commodity HPC clusters by the U.S. Department of Energy Labs: Livermore, Los Alamos, and Sandia. The particular version of SNAP that we used is available as a benchmark for commodity HPC systems (https://hpc.llnl.gov/cts-2-benchmarks). SNAP uses the discrete ordinates method, where the three-dimensional momentum space is discretized using a specified number of energy groups and a discrete set of angular intervals. SNAP uses a regular Cartesian grid in three dimensions for the spatial coordinates. The parallelization scheme is domain decomposition using a two-dimensional process grid. For a given parallel job the user specifies the number of MPI ranks in the Y and Z dimensions (npey and npez respectively). The total number of MPI ranks is the product of these two values. We studied SNAP in a weak-scaling scenario, where the size of the local domain remains constant, and the global domain increases as one increases the number of ranks in the Y and Z dimensions. With the normal benchmark inputs, each MPI process owns a narrow pencil of the three-dimensional spatial domain : typically 4 grid cells in each of the Y and Z dimensions and 640 grid cells in the X dimension. The fact that there are only 4 grid cells in each of the Y and Z dimensions makes the simulation very sensitive to communication, because most of the grid cells are on a domain boundary, and so the ratio of communication to computation is high. Input parameters for a job using a total of 64 MPI ranks is shown below, where there are 8 ranks in the Y dimension (npey=8), and 8 ranks in the Z dimension (npez=8). To maintain weak scaling, one increases the global number of grid cells in the Y and Z dimensions (input parameters ny and nz) keeping them proportional to the number of MPI ranks in the Y and Z dimensions (npey and npez). SNAP uses an iterative solution method with a sequence of "sweeps" along each angular direction. The computation in SNAP is dominated by the dim3_sweep() routine. This routine can achieve high computational performance because it is structured for good cache re-use and has good potential for SIMD code generation. SNAP is written in Fortran-90, and the Intel Fortran compiler is particularly effective at generating efficient code for SNAP.

SNAP output includes a solve time which depends on many things including the number of iterations and the total number of unknowns. The code also outputs a "grind time", which is the solve time divided by the number of iterations and the number of unknowns. The Figure of Merit (FOM) for the SNAP benchmark is one over the "grind time". The FOM is a direct indicator of the performance of the system. If you solve the same problem in half the time, the FOM increases by 2x, and if you solve a problem with 2x more unknowns in the same time per iteration, the FOM increases by 2x. We studied weak-scaling, where the size of the global domain increases linearly with the number of MPI ranks. For ideal scaling, the solve time per iteration would remain constant, and so the FOM would increase linearly with the number of MPI ranks.

2 https://www.lanl.gov/projects/codesign/proxy-apps/lanl/index.php

```
! Input from namelist
&invar
  npey=8
  npez=8
  ichunk=64
  nthreads=1
  nnested=1
  ndimen=3
  nx=640
  lx=64.0
  ny=32
  ly=3.2
  nz=32
  lz=3.2
  nmom=4
  nang=48
  ng=54
  epsi=1.0E-4
  iitm=5
  oitm=100
  timedep=1
  tf=0.02
  nsteps=2
  mat_opt=1
  src_opt=1
  scatp=0
  it_det=0
  fluxp=0
  fixup=1
  soloutp=0
  popout=0
  swp_typ=0
  angcpy=1
  multiswp=1
/
```

Figure 1. Example SNAP input parameters for an 8x8 simulation

Observations and Analysis

A summary of the performance measurements is provided in Table 2, which shows the solve time and the Figure of Merit for SNAP, scaling from 8 cores (1 compute node) to 504 cores (63 compute nodes) on IBM Cloud. In all measurements we used one MPI rank per core without OpenMP threading. When using a single compute node, all communication is through shared memory, which is very efficient. As one scales out to an increasing number of compute nodes, more of the communication is over the Ethernet interface using TCP, and a larger fraction of the solve time is spent on MPI communication, resulting in somewhat lower performance per node. However, the aggregate performance as indicated by the Figure of Merit continues to show significant improvement over the full range of compute nodes available in our LSF cluster, as can be seen in Figure 2.

MPI Process Grid	Cores Utilized	Solve Time (sec)	Figure of Merit	FOM per node
2 X 4	8	111.9	0.571	0.571
4 X 4	16	148.4	0.908	0.454
8 X 8	64	222.0	2.490	0.311
16 X 16	256	260.4	8.602	0.269
21 X 24	504	312.7	15.221	0.242

Table 2. Performance results for SNAP on IBM Cloud using a local domain with dimensions 640x4x4 grid cells.

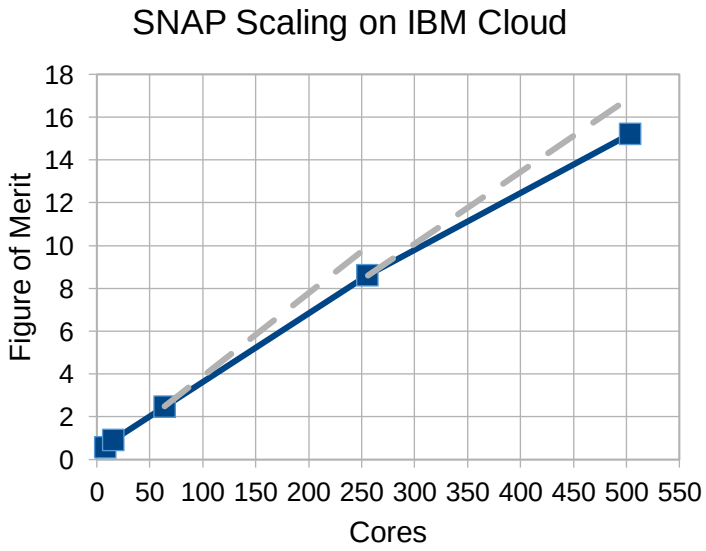

Figure 2. Scaling curve for SNAP on IBM Cloud using a local domain with dimensions 640x4x4 grid cells. The dashed lines indicate perfect linear scaling relative to measured values at 64 cores or 256 cores.

With the multi-sweep option specified in the input file, the main communication in SNAP is a novel boundary exchange method involving nearest neighbors on the two-dimensional process grid. This method uses non-blocking MPI calls, MPI_Isend/MPI_Irecv, where completion of these routines is checked by non-blocking calls to MPI_Test and MPI_Testsome. This approach provides improved messaging performance compared to the more common method where blocking routines, MPI_Wait or MPI_Waitall, are used to ensure completion of the MPI_Isend/MPI_Irecv requests. For our benchmark parameters, boundary exchange is with message sizes of ~100 KBytes, which is large enough to stress network bandwidth more than latency. This is beneficial for cloud-based platforms with TCP communication. Since the main communication in SNAP is point-to-point messaging, we expect rather minor differences in performance when using different MPI implementations, and we verified this experimentally. There is one issue with the SNAP code that we encountered. SNAP makes extensive use of message tags, and the formulas used by SNAP to compute tags can result in values that exceed the limits in some MPI implementations, including Intel MPI (https://github.com/lanl/SNAP/issues/6). The MPI specification requires that the upper bound for tag values must be at least 32767, so MPI applications including SNAP should ensure that they use message tags with smaller values.

Conclusions

The IBM Spectrum LSF on IBM Cloud offering now includes the Intel oneAPI HPC Toolkit, and it provides an environment for building and running HPC applications that is very similar to traditional HPC clusters, thus lowering the barrier to cloud deployment. We demonstrated good scaling for a communication-intensive HPC application, SNAP, on an environment deployed on IBM Cloud using that offering.

The observed successful executions of the SNAP workload occurred using a stock environment with no special preparation; the default image installed on LSF worker nodes includes the Intel oneAPI MPI library. We attribute this success to several factors. SNAP's tolerance of inter-node latency suits the cloud environment which understandably does not operate with latencies in

the sub-1µs range found in specialized InfiniBand clusters. The compute hardware substantiating the worker nodes is of appropriate efficacy to run this workload efficiently -- the executions mentioned above all occurred using Cascade Lake generation processors. High computational efficiency is provided by Intel compilers, and solid messaging performance is provided by Intel MPI. Finally, the integration of the workload manager software (LSF) and MPI allows for dispatch of the jobs in a manner that ensures that the MPI environment variables are set correctly for optimal execution of the jobs.

Trademarks

IBM®, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

SoftLayer® is a registered trademark of SoftLayer, Inc., an IBM Company.

Other company, product, or service names may be trademarks or service marks of others.

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.