Analyzing and Tuning SPECAccel Performance for GPU Workloads on Lenovo ThinkSystem Servers

Introduces the accelerator programing paradigm standard, OpenACC and OpenCL

Describes the Analysis runtime system behavior of SPECAccel benchmark

Introduces the Lenovo ThinkSystem SR650 server

Provides a tuning recipe to optimize the SPECAccel performance for Lenovo ThinkSystem servers

Jimmy Cheng

LENOVO PRESS

Abstract

The use of GPU high-performance accelerators are especially important in High Performance Computing (HPC) workloads. As a result, it is critical to ensure that the GPUs provide the most processing power possible by adjusting key performance and measuring the results.

The SPECAccel benchmark uses the OpenCL and OpenACC programing paradigm to provide a comparative measure of parallel computing performance among systems equipped with an accelerator. Through runtime characteristic analysis, this paper provides a best practices recipe for Lenovo® ThinkSystem™ servers to obtain the best performance for GPU workloads, as well as other applications that have characteristics similar to the SPECAccel benchmark.

This paper is intend for ThinkSystem end users and technical sales representatives who want to understand how to tune GPU performance. The paper assumes readers are familiar with Linux and have basic experience with programming languages such as C/C++.

At Lenovo Press, we bring together experts to produce technical publications around topics of importance to you, providing information and best practices for using Lenovo products and solutions to solve IT challenges.

See a list of our most recent publications at the Lenovo Press web site:

http://lenovopress.com

Do you have the latest version? We update our papers from time to time, so check whether you have the latest version of this document by clicking the **Check for Updates** button on the front page of the PDF. Pressing this button will take you to a web page that will tell you if you are reading the latest version of the document and give you a link to the latest if needed. While you're there, you can also sign up to get notified via email whenever we make an update.

Contents

Introduction

The revolution of Artificial Intelligence (AI), machine learning (ML) and deep learning (DL) technology requires massive amount of computational power for workloads like training on deep neural networks (DNNs) and convolutional neural networks (CNNs)

A traditional data center computing environment that primarily uses the basic system CPUs (homogeneous) can become a bottleneck that constrains project development speed. Conversely, a heterogeneous computing environment equipped with accelerators (e.g., GPUs, FPGAs, and DSPs) has the ability to perform operation in parallel, which can significantly improve computing performance.

Accelerator Programing Paradigm

Traditionally, when adopting a proprietary heterogeneous accelerator architecture, users need to rewrite and compile the existing source code via the SDK (e.g., NVIDIA CUDA) provided by each vendors. To increase the source code portability, standard programing paradigms, such as OpenACC and OpenCL, were introduced to port existing source code across a wide-variety of heterogeneous hardware architectures.

OpenCL

The Open Computing Language (OpenCL) is a cross-platform programing framework that provides Application Programming Interfaces (APIs) and C-like language to optimize accelerators within a heterogeneous environment.

There are four models programmers need to understand before using OpenCL:

- Platform Model
- Memory Model
- Execution Model
- Host API

OpenCL uses the Platform Model to formulate the internal components of each accelerator (GPU, DSP, or FPGA hardware). As Figure 1 on page 4 shows, a heterogeneous host containing multiple accelerators is called a Compute Device in OpenCL. It may include one or more Compute Units, and a Compute Unit can be further divided into one or more Process Element (PE).

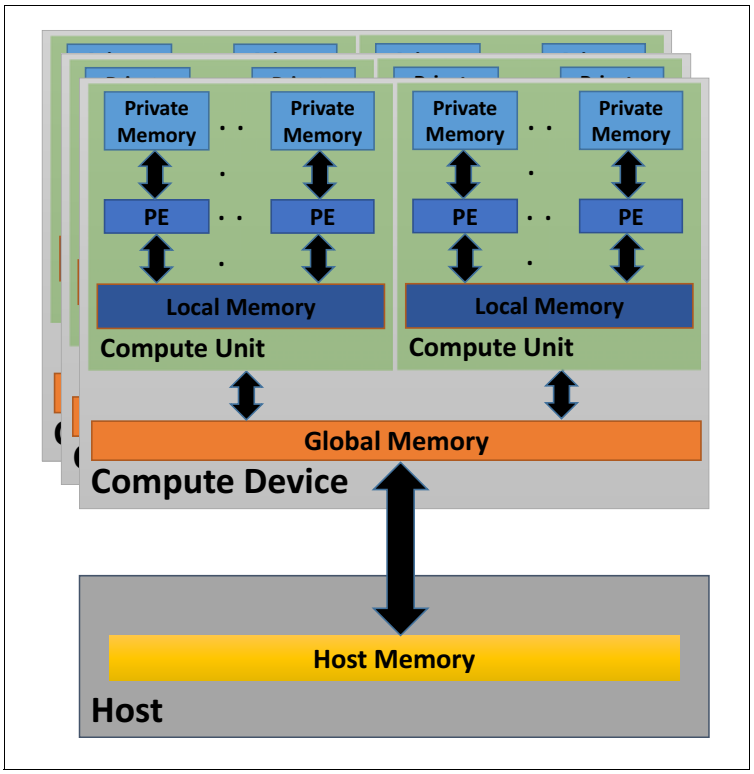

Figure 1 Memory model of OpenCL

The Memory Model defines the memory structure and behavior of a given system, as follows:

- Global Memory can be access by all the Process Elements (PEs)
- All PEs in the same work-group can access shared Local Memory
- Each PE has its own Private Memory region, which is not visible to others
- Host Memory is the memory region directly available to the host

The data flow from host to device must flow from Host Memory to Global Memory, Global Memory to Local Memory, and Local Memory to Private Memory, or vice versa.

The Execution Model is a framework that defines the execution environment, including how a Host Program submits Work-Items to a Compute Device. Each Work-Item is a basic unit executed on a PE. A collection of several Work-Items form a Work-Group. Depending on a Work-Group's memory requirement, several concurrent Work-Groups can reside on a Compute Unit. The Host Program also defines Context, which is the environment including the devices, memory, and command queues being used.

The Host APIs are the set of API functions OpenCL provides to control the system.

Including the Compute Devices exploration in the system, memory allocation, transportation on both the Host and the Compute Device, and launch kernel execution on Compute Devices.

Table 1 shows an example of a serial Matrix Addition modified for use with OpenCL.

Table 1

Serial Matrix Addition	OpenCL Matrix Addition

`void MatrixAdd(int n, int *A, int *B, int *C)`
`{`
`  int i;`
`for(i=1; i<=n; i++)`
`  C[i] = A[i] + B[i]`
`}`
	`_kernel void MatrixAdd(_global int *A,`
	`_global int *B, _global int *C)`
	`{`
	`  int idx = get_global_id(0);`
	`  C[idx] = A[idx] + B[idx];`
	`}`

Because the Serial Matrix Addition code calculates a result by running the calculation over and over serially, the execution time is highly dependent on the size of the matrix. The OpenCL version, on the other hand, creates work items with the matrix size optimized to enable kernel functions to execute in parallel, resulting in consistently minimized execution time.

For more information about OpenCL, visit the following web page:

https://www.khronos.org/opencl/

OpenACC

OpenACC is a directive-based programming model, using high-level pragmas to offload workload to the accelerators in the system. A pragma is a form of code annotation that gives instructions to the compiler on how to compile the code.

The simple syntax of OpenACC is as follows:

```
#pragma acc Directive Clauses
<Source Code>
```

The syntax is as follows:

- The OpenACC pragma start with the # symbol
- `acc`: Instruct compiler it's an OpenACC directive
- *`Directive`*: OpenACC commands to altering the code
- *`Clauses`*: Specifiers or additions to directives.

In the following OpenACC example, two pragmas are used for a single loop in C language:

```
#pragma acc parallel
{
  #pragma acc loop independent
  for (int i=0; i<100; i++)
    a[i]=0
}
```

The `parallel` directive marks the code region needed for parallel execution. The `loop` directive with `independent` clauses tells the compiler that all iterations of the loop are independent.

For more information about OpenACC, visit the following web site:

https://www.openacc.org/

SPECAccel benchmark

The SPEC ACCEL benchmark contains a suite that focuses on parallel computing performance using the OpenCL 1.1, OpenMP 4.5, and OpenACC 1.0 standards. The suite exercises the performance of the accelerator, host CPU, memory transfer between host and accelerator, support libraries and drivers, and compilers.

In this section, we focus on OpenCL and OpenACC, which are used to offload the workload to the accelerator.

Table 2 and Table 3 list the application areas and programing languages and suites for each sub-benchmark.

Table 2 OpenACC

Benchmark	Language	Application Area
303.ostencil	C	Thermodynamics
304.olbm	C	Computational Fluid Dynamics, Lattice Boltzmann Method
314.omriq	C	Medicine
350.md	Fortran	Molecular Dynamics
351.palm	Fortran	Large-eddy simulation, atmospheric turbulence
352.ep	C	Embarrassingly Parallel
353.clvrleaf	Fortran, C	Explicit Hydrodynamics
354.cg	C	Conjugate Gradient
355.seismic	Fortran	Seismic Wave Modeling
356.sp	Fortran	Scalar Penta-diagonal solver
357.csp	C	Scalar Penta-diagonal solver
359.miniGhost	C, Fortran	Finite difference
360.ilbdc	Fortran	Fluid Mechanics
363.swim	Fortran	Weather
370.bt	C	Block Tridiagonal Solver for 3D PDE

Table 3 OpenCL

Benchmark	Language	Application Area
101.tpacf	C++	Astrophysics
103.stencil	C++	Thermodynamics
104.lbm	C++	Fluid Dynamics
110.fft	C	Signal processing
112.spmv	C++	Sparse Linear Algebra
114.mriq	C	Medicine
116.histo	C	Silicon Wafer Verification

Benchmark	Language	Application Area
117.bfs	C	Electronic Design Automation, Graph Traversals
118.cutcp	C	Molecular Dynamics
120.kmeans	C++	Dense Linear Algebra, Data Mining
121.lavamd	C	N-Body, Molecular Dynamics
122.cfd	C++	Unstructured Grid, Fluid Dynamics
123.nw	C++	Dynamic Programming, Bioinformatics
124.hotspot	C	Structured Grid, Physics Simulation
125.lud	C++	Dense Linear Algebra, Linear Algebra
126.ge	C++	Dense Linear Algebra, Linear Algebra
127.srad	C	Structured Grid, Image Processing
128.heartwall	C	Structured Grid, Medical Imaging
140.bplustree	C	Graph Traversal, Search

For more information, please refer to this web page:

https://www.spec.org/accel/

ThinkSystem SR650

We used the Lenovo ThinkSystem SR650 (Figure 2) for our testing in the lab. The SR650 two-socket server offers industry-best system performance with the Intel Xeon Processor Scalable Family processors with up to: 28-cores, 38.5 MB of last level cache, 2933 MHz memory, and two Ultra Path Interconnect (UPI) links between processors that operate at up to 10.4 GT/s.

Figure 2 Lenovo ThinkSystem SR650

The SR650 offers configuration flexibility, powerful computing power, and high-end GPU upgrade capability for SPECAccel-like applications in the data center.

For more information about the SR650, see the Lenovo Press product guide:

https://lenovopress.com/lp1050-thinksystem-sr650-server-xeon-sp-gen2

The configuration we used in our lab consisted of the following:

- 1x Lenovo ThinkSystem SR650 server
- 2x Intel Xeon Gold 6240 Processors (18 cores, 2.60 GHz)
- 768 GB memory (24x 32G B RDIMMs running at 2933 MHz)
- 1x 1 TB 12Gbps SAS 2.5" SSD
- 1x NVIDIA Tesla V100 16GB
- Red Hat Enterprise Linux Server release 7.6, Kernel 3.10.0-957.el7.x86_64

Analysis of CPU performance

Figure 3 and Figure 4 illustrate how busy each CPU core was during OpenACC and OpenCL testing, measuring accumulated CPU utilization (unit: percentage) every second. Unlike a homogeneous system where the CPU handles all workload, the SPECAccel workload was offloaded from the CPU to the GPU. As a result, only a few CPU cores were busy during SPECAccel execution.

We provide GPU profiling information in “Analysis of NVIDIA V100 GPU performance” on page 13.

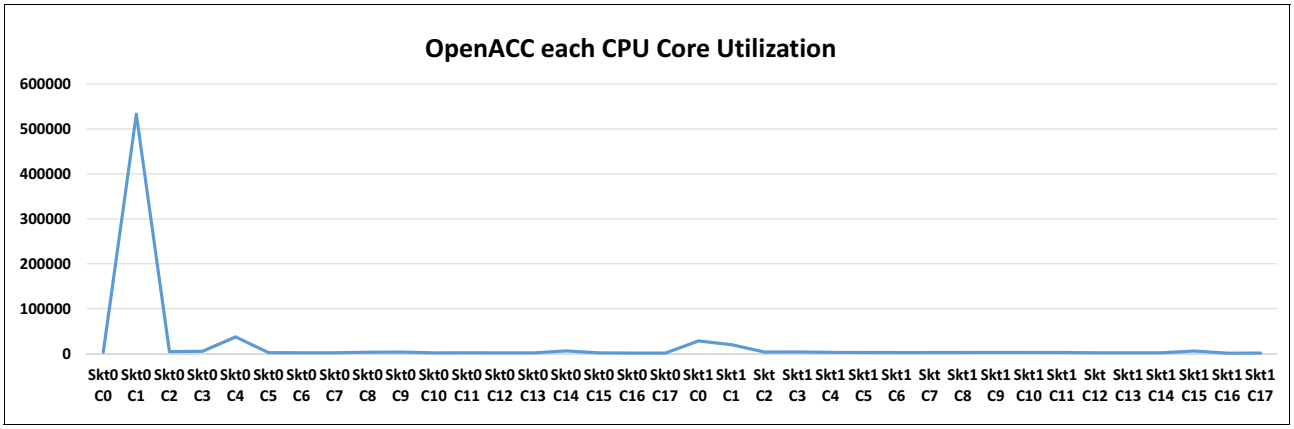

Figure 3 CPU utilization when running OpenACC

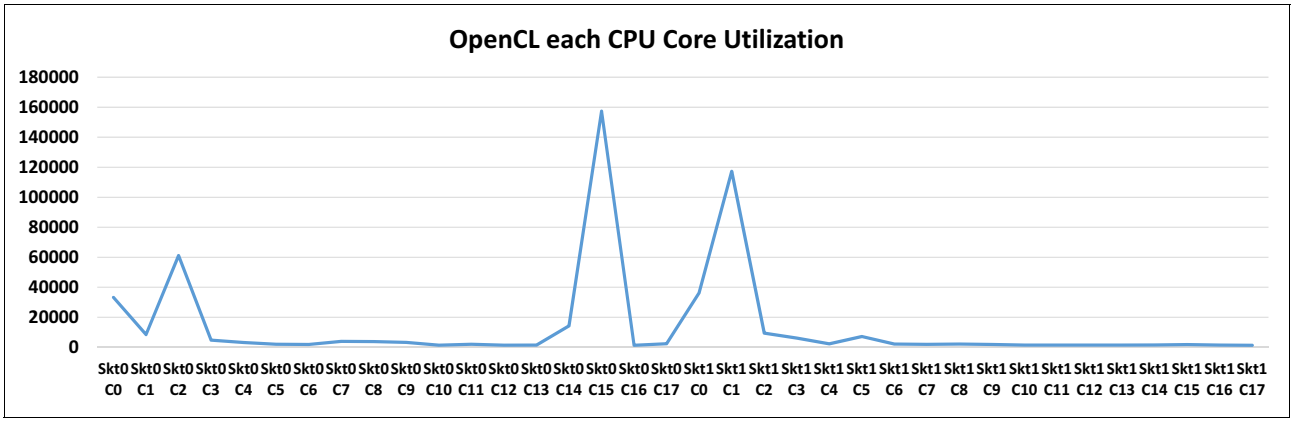

Figure 4 CPU utilization when running OpenCL

Analysis of memory performance

Since the 1980s, processor speed has increased significantly faster than memory speed. As a result, the processor-memory performance gap has become a primary obstacle of system performance especially for memory-bound workloads. In this section, we will focus on memory characteristic analysis of SPECAccel through memory footprint and memory bandwidth.

Memory bandwidth

The theoretical memory bandwidth can be calculated by the following formula:

`Memory frequency × (data width ÷ 8) × number of channels × number of processor sockets × 90%`

Our lab test ran on the ThinkSystem SR650, with a memory frequency of 2933 MHz. Each of the two memory controllers contained three memory channels with a data width of 64 bits.

Using the formula, the highest theoretical memory bandwidth is calculated as follows:

`2.933 GHz × (64 bits ÷ 8) × 6 channels × 2 processor sockets × 90% = 253.4 GB/second`

Figure 5 shows the system memory read and write bandwidth while running OpenACC. The OpenACC is not a bandwidth-intensive workload, using at most 10% of the system theoretical bandwidth. The system memory read and write ratio is well balanced at approximately 1:1 (53:47).

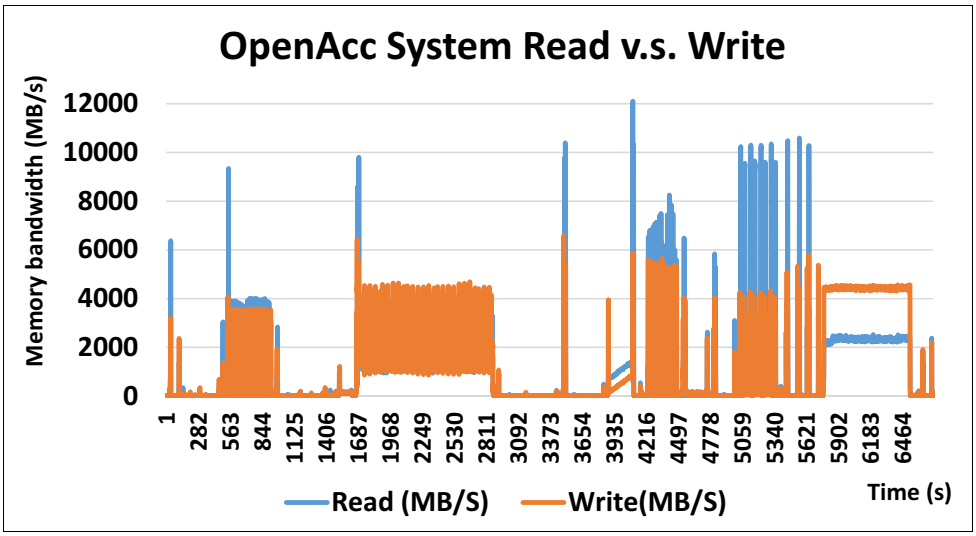

Figure 5 System memory bandwidth of OpenACC

Table 4 shows the memory access pattern of socket 0 and socket 1 of OpenACC. Socket 0 used most of the memory bandwidth because the GPU is connect to CPU0.

Table 4 OpenACC Memory read and write bandwidth on different sockets

	Read	Write
SKT0	91.45%	93.71%
SKT1	8.55%	6.29%

Figure 6 shows the system memory read and write bandwidth while running OpenCL. OpenCL is not a bandwidth-intensive workload, using no more than 20% of the system theoretical bandwidth. The system memory read and write ratio is almost 2:1 (63:37).

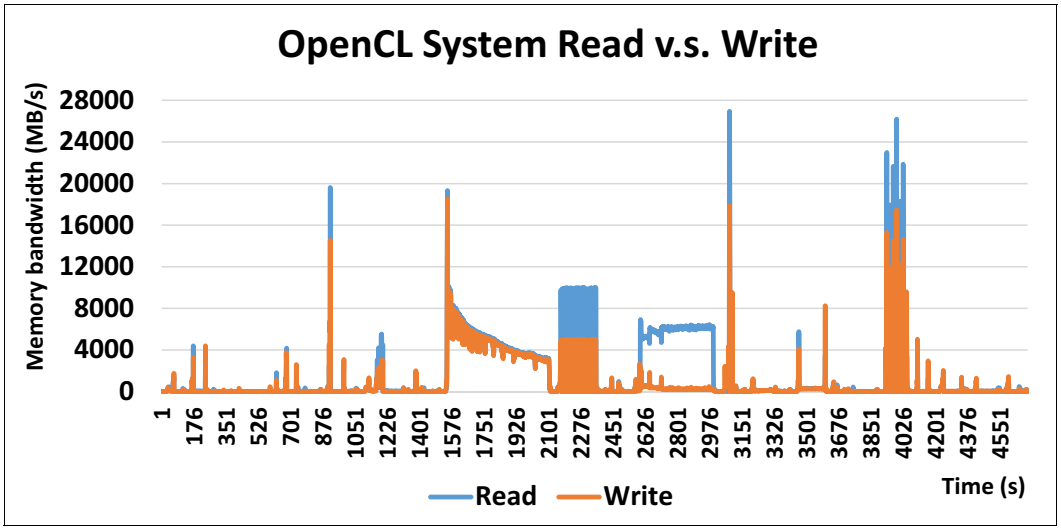

Figure 6 System memory bandwidth of OpenCL

Table 5 shows the memory access pattern of socket 0 and socket 1 of OpenCL. Socket 0 used most of memory bandwidth because the GPU is connect to CPU0.

Table 5 OpenCL Memory read and write bandwidth on different sockets

	Read	**Write**
Socket 0	91.45%	93.71%
Socket 1	8.55%	6.29%

Memory footprint

Virtual memory (VM) is a memory management technique implemented in modern operating systems. It usually simulates dynamic random access memory (DRAM) on a hard disk drive or solid-state drive to extend memory capacity when physical memory capacity is not sufficient for the application. Although VM solves spatial issues in physical memory, the lower bandwidth affects the application performance significantly. Table 5 shows the performance difference between memory and various type of drives.

Table 6 The theoretical bandwidth comparison between DRAM and different types of hard drives

Hardware Type	**Theoretical bandwidth**
SAS 15K RPM hard disk drive (HDD)	200 MB/s
2.5-inch solid-state drive (SSD)	500 MB/s
PCIe NVMe solid-state drive	2.8 GB/s
DDR4 2933 MHz memory (6 memory channels)	140.7 GB/s (6 channels)

To ensure SPECAccel performance was not affected by the use of virtual memory, we used the Linux command **`free -h`** to diagnose the memory footprint and SWAP memory (VM in Linux called SWAP) usage.

Figure 7 shows the result of using this command.

```
[root@cyborg01os NFS]# free -h
              total        used        free      shared  buff/cache   available
Mem:           755G        5.6G        749G        121M        651M        748G
Swap:          4.0G          0B        4.0G
```

Figure 7 free -h command

As shown in Figure 8, SPECAccel does not use SWAP when it is running, and the requirement of system memory capacity is low; less than 1% of system memory is used during its operation.

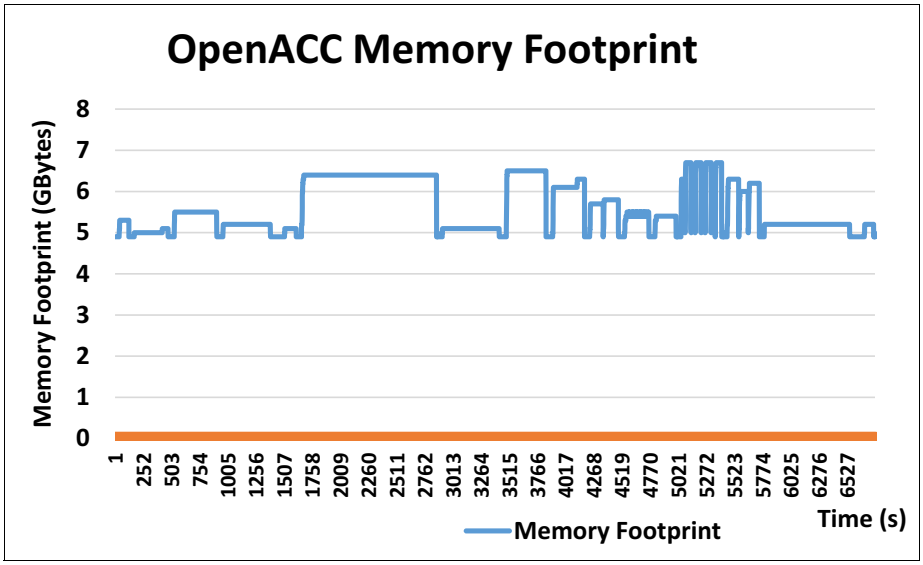

Figure 8 Memory footprint and SWAP memory usage of SPECAccel

Analysis of PCIe performance

The industry-standard PCIe is major interconnect interface for high-speed peripheral to address the growing appetite for high-speed peripheral bandwidth. Each physical PCIe slot can offers scalable lane widths for different bandwidth requirements, such as GPUs, Ethernet cards, InfiniBand cards, and high-end NVMe storage.

Table 7 shows the PCIe theoretical bandwidth of each generation, showing the different lane widths.

Table 7 PCIe bandwidth of each generation on different lanes width

Bandwidth (GB/s)	**PCIe Lanes Width**				
	x1	**x4**	**x8**	**x16**	**x32**
PCIe 1.0	0.25	1	2	4	8
PCIe 2.0	0.5	2	4	8	16
PCIe 3.0	1	4	8	16	32
PCIe 4.0	2	8	16	32	64
PCIe 5.0	4	16	32	64	128

SPECAccel offloads the workload from the CPUs to the NVIDIA V100 GPU by exchanging data with the system memory; therefore, the connection bandwidth becomes a potential bottleneck to overall performance. In our test environment, we installed the NVIDIA Tesla V100 16GB GPU in a PCIe 3.0 x16 slot to maximize the connection bandwidth. The CPU and GPU topology are shown in Figure 9.

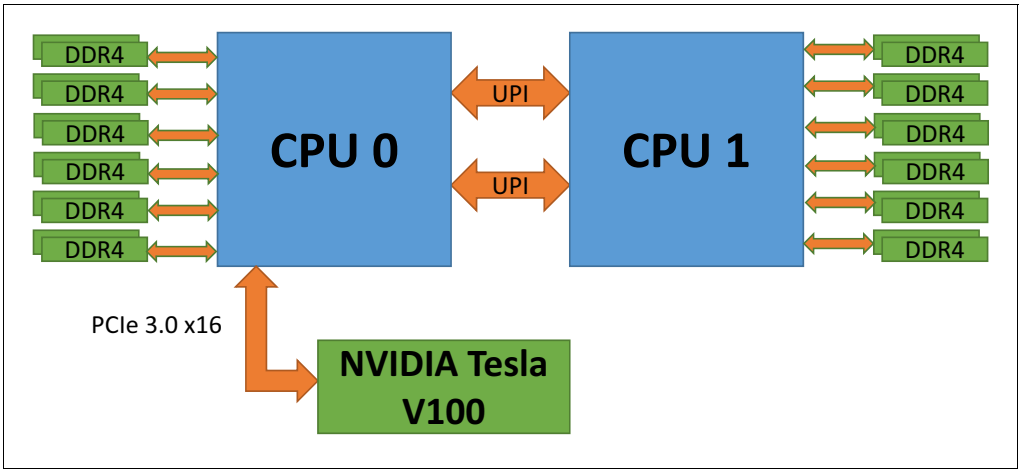

Figure 9 NVIDIA Tesla V100 connection topology in the ThinkSystem SR650

Figure 10 on page 12 and Figure 11 show the NVIDIA V100 PCIe read and write bandwidth when running SPECAccel with OpenACC and OpenCL respectively.

The peak PCIe bandwidth of OpenACC and OpenCL is only about 35.6% and 23.5% of the PCIe 3.0 x16 slot's theoretical bandwidth. This demonstrates that the connection bandwidth between one of the CPUs and the NVIDIA GPU is not a performance bottleneck for the SPECAccel benchmark.

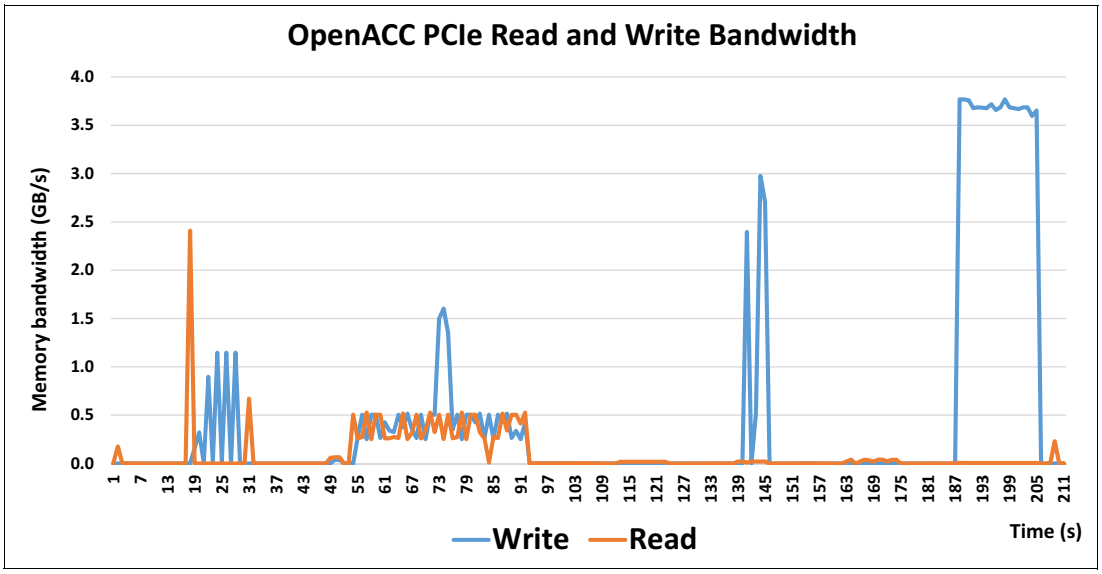

Figure 10 NVIDIA V100 read/write bandwidth of OpenACC

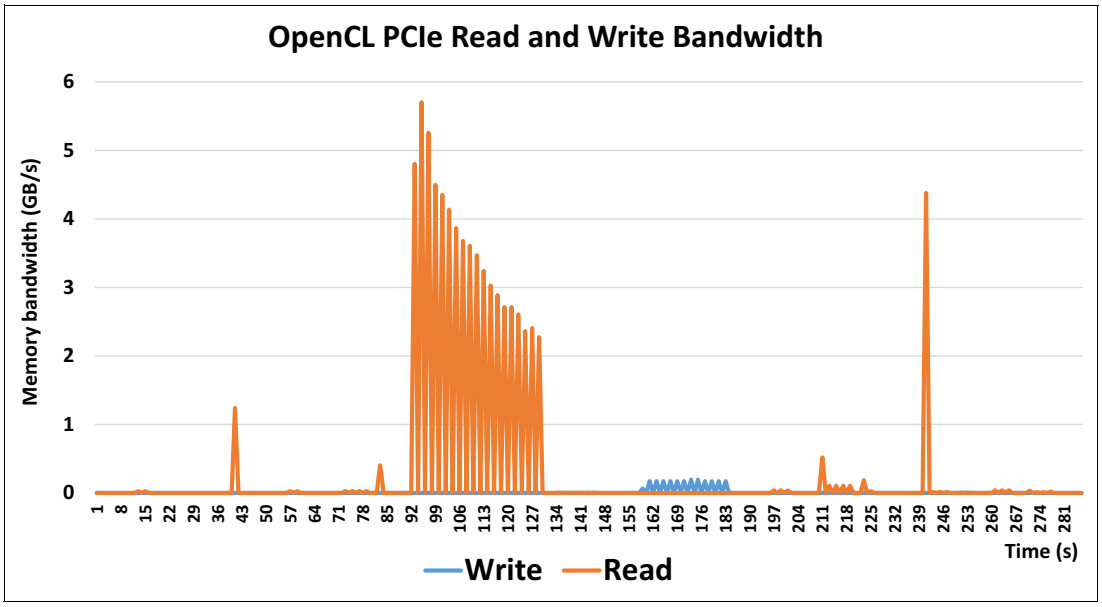

Figure 11 NVIDIA V100 read/write bandwidth of OpenCL

Analysis of NVIDIA V100 GPU performance

As a target for deep learning algorithms and frameworks, the NVIDIA V100 GPU is composed of multiple GPU Processing Clusters (GPCs), Texture Processing Clusters (TPCs), Streaming Multiprocessors (SMs), and memory controllers.

The NVIDIA V100 GPU consists of:

- 6x GPU Processing Clusters (GPCs), each with:
  - 7x Texture Processing Clusters (TPCs)
  - 14x Streaming Multiprocessors (SMs), 2 for each TPC
- 84x Volta SMs, each with:
  - 64x FP32 cores
  - 64x INT32 cores
  - 32x FP64 cores
  - 8x Tensor Cores
  - 4x texture units
- Eight 512-bit memory controllers (4096 bits total)

In our lab, we sampled the NVIDIA V100 GPU and its local memory every second while running SPECAccel, to observe the runtime behavior. Figure 12 and Figure 13 on page 14 illustrate the GPU SM running frequency and utilization of OpenACC and OpenCL, respectively.

Due to the powerful computing power of NVIDIA V100 GPU, each sub-benchmark of ACCEL workload can be consumed quickly, without occupying the SM for long time. On the other hand, the operating frequency reached its maximum limit (1380MHz) most of the time while the workload was running and then returned to its lowest frequency (135MHz) when the workload finished.

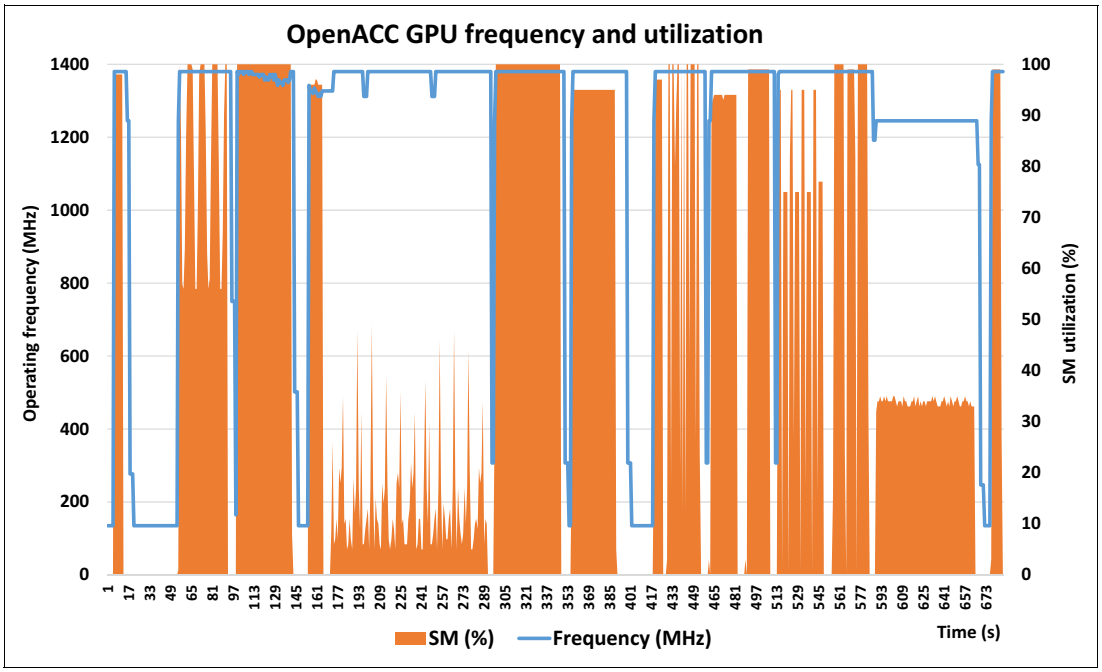

Figure 12 GPU frequency and utilization when running OpenACC

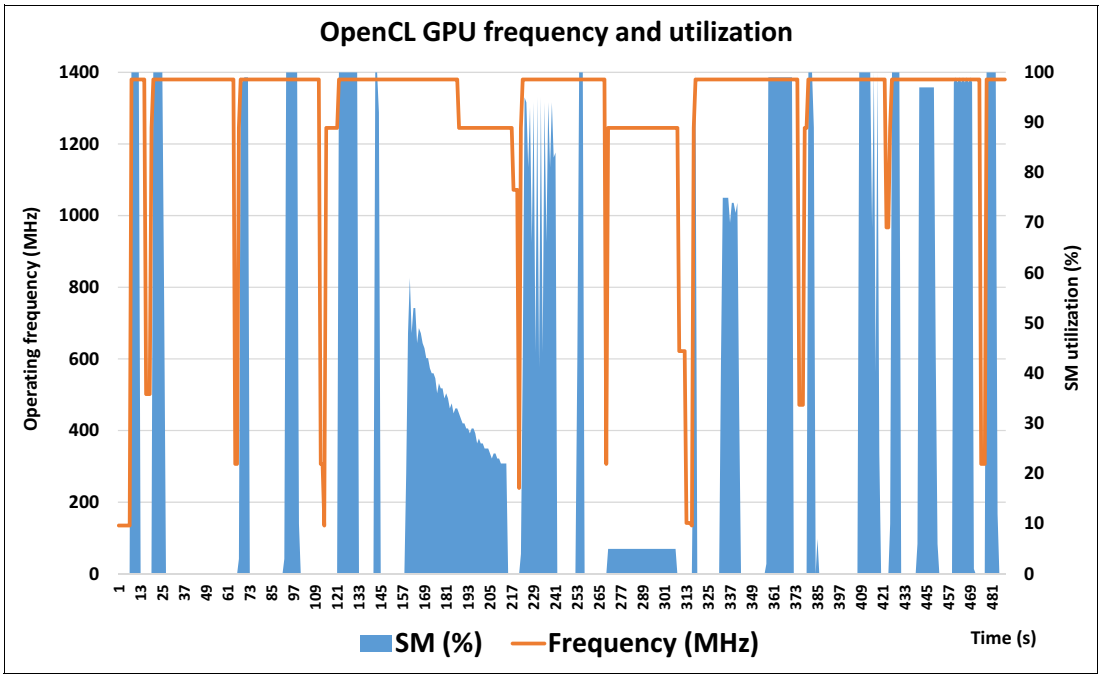

Figure 13 GPU frequency and utilization when running OpenCL

The memory capacity of the NVIDIA V100 GPU is 16GB, far less than the system memory (768GB), however, the high-bandwidth memory (HBM2) of the GPU can deliver up to 900 GB/s memory bandwidth within the four memory stacks inside the NVIDIA V100.

As Figure 14 on page 15 and Figure 15 show, the GPU memory requirement for SPECAccel seldom reaches 100%. In other words the amount of GPU memory installed may not be a major bottleneck for an OpenAccel-like workload.

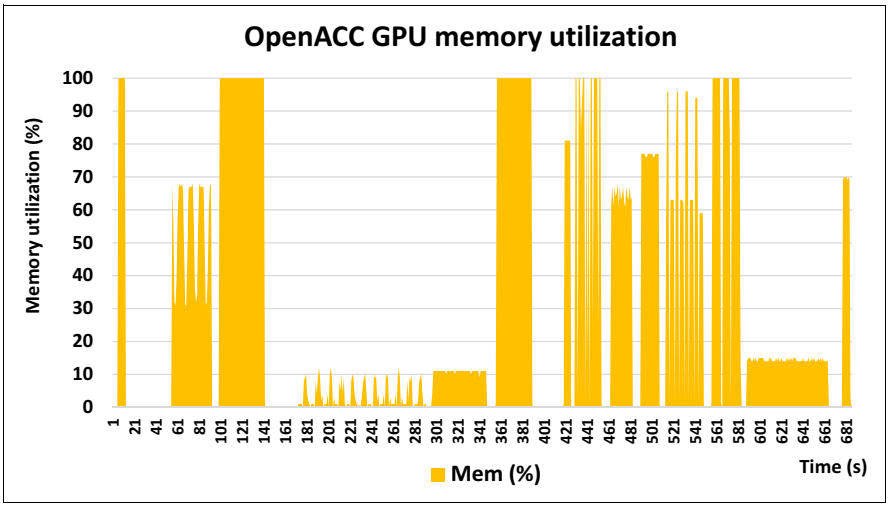

Figure 14 GPU memory utilization of OpenACC

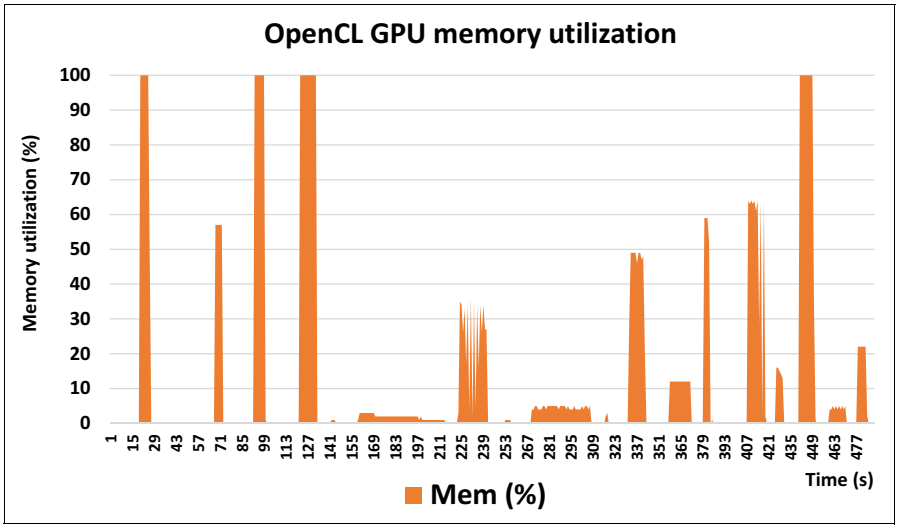

Figure 15 GPU memory utilization of OpenCL

Performance tuning

SPECAccel offloads workloads from the CPUs to the GPU, so only a few CPU cores are required at the highest CPU frequency. As a result, for computing-intensive parallel applications and other SPECAccel-like workloads, we recommend the ThinkSystem SR650 be configured as follows:

- Intel Xeon Gold 6240 Processor
- Maximum Performance mode (in UEFI)
- C-State Autonomous (in UEFI)
- Binding the application threads to the correct CPU core and memory
- Enable Persistence Mode for the GPU

Due to a thermal restrictions, the SR650 supports only processors up to 150W TDP when the NVIDIA V100 GPU is used. For SPECAccel, we suggest the use of the Intel Xeon Gold 6240

processor, which provides sufficient number of CPU cores. The Turbo Boost frequency of up to 3.90 GHz helps to speed up the offload process.

UEFI settings

Lenovo provides different preset Operating Modes which can be selected in UEFI to adapt to different scenarios. The operating mode Maximum Performance is recommend for the SPECAccel workload.

1. Boot the server and enter System Setup by pressing F1 when prompted
2. From the menus, select **System Settings**, then **Operating Modes**.
3. Navigate to the Choose Operating Mode field and press Enter. Figure 16 appears.

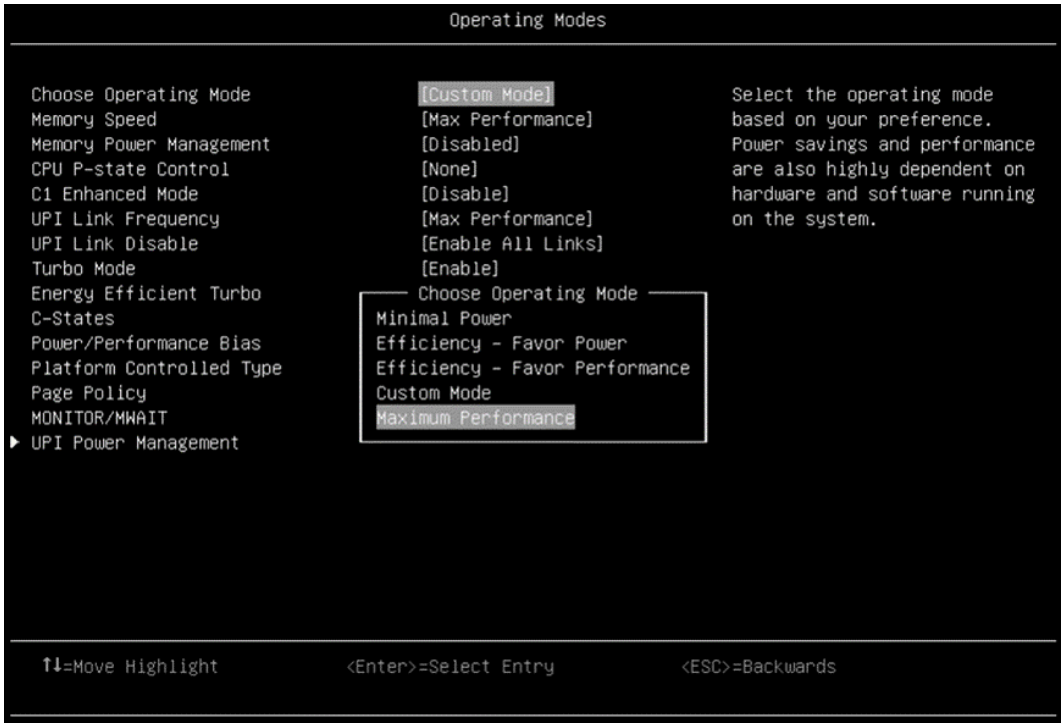

Figure 16 Selecting Maximum Performance mode in UEFI

ThinkSystem servers like the SR650 offer five different operating modes:

- **Minimal Power** mode strives to minimize the absolute power consumption of the system while it is operating. The trade-off is that performance may be reduced in this mode, depending on the application that is running.
- **Efficiency: Favor Power** mode maximizes the performance/watt efficiency with a bias towards power savings. It provides the best features for reducing power and increasing performance in applications where maximum bus speeds are not critical.
- **Efficiency: Favor Performance** mode optimizes the performance/watt efficiency with a bias towards performance.
- **Maximum Performance** mode maximizes the absolute performance of the system without regard for power. In this mode, power consumption is not taken into consideration. Attributes like fan speed and heat output of the system may increase in addition to power consumption. Efficiency of the system may go down in this mode, but the absolute performance may increase depending on the workload that is running.

– **Custom Mode** allows the user to individually modify any of the low-level settings that are preset and unchangeable in any of the other four preset modes.

4. Select **Maximum Performance** and press Enter.
5. Select the same menu again, and this time, select **Custom Mode** and press Enter as shown in Figure 17.

   We are using Maximum Performance mode but will then be tweaking it to change the C-States setting by then changing the mode to Custom Mode. Custom Mode inherits the settings from the mode previously selected.

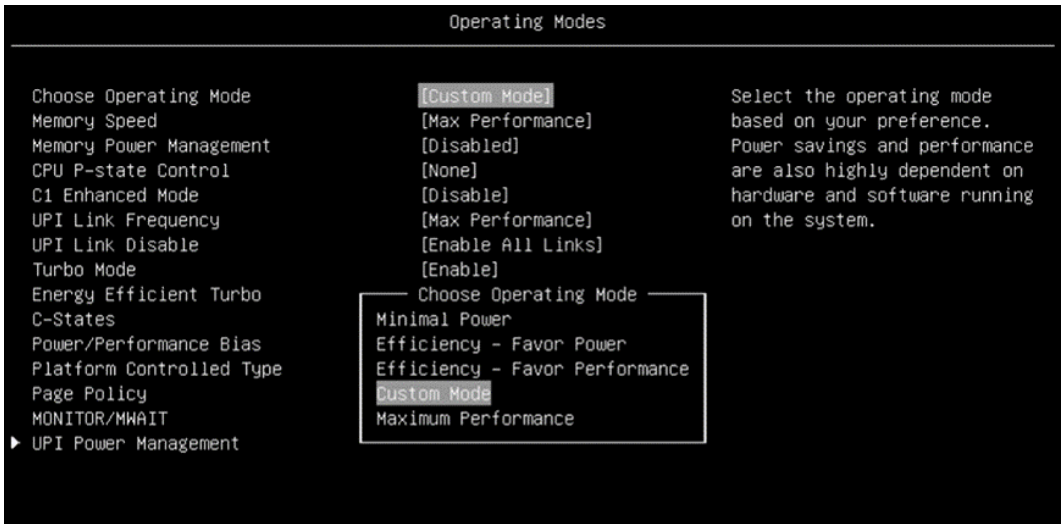

Figure 17 Custom Mode in UEFI menu

6. Select the C-States field in the same page and press Enter, Figure 18.

Figure 18 C-States in UEFI menu

The C-states setting offers three choices:

- **Legacy** — The operating system initiates the C-state transitions. Some operating system software may defeat the ACPI mapping (e.g., the intel_idle driver).
- **Autonomous** — HALT and C1 requests get converted to C6 requests in hardware.
- **Disable** — Only C0 and C1 are used by the operating system. C1 gets enabled automatically when an OS autohalts.

7. Select **Autonomous** and press Enter.

   The Autonomous C-States mode reduces the power consumption of the CPU in idle state so that active CPU cores have more thermal margin to reach higher CPU turbo frequency and stay there longer.

8. Exit UEFI and reboot the server.

Binding application threads to specific CPU cores and memory

The automatic NUMA balancing mechanism implemented in modern operating systems move tasks closer to the CPU core and memory domain they are accessing. The intent is to improve the application performance running on NUMA architecture systems such as the Intel Xeon Scalable processor-based ThinkSystem servers.

The status of automatic NUMA balancing can be checked using the Linux command below:

```
# cat /proc/sys/kernel/numa_balancing
```

The output shows the status:

- 0: disable
- 1: enable

However, the automatic algorithm may not be suitable for all scenarios. Whenever an application thread is moved to another processor core, there is a performance cost associated with the move. If such context switches occur too frequently, it can be more efficient to disable OS automatic NUMA binding and manually bind all threads to specific cores and the corresponding memory domain.

There are three steps to bind an application manually:

1. First check the CPU and memory topology by using the following Linux command:

   ```
   numactl -H
   ```

   As shown in Figure 19, the command output lists the CPU cores of different nodes and the memory domain it is attached to.

```
# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
node 0 size: 392894 MB
node 0 free: 382540 MB
node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
node 1 size: 393216 MB
node 1 free: 382386 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

Figure 19 Output of the numactl -H command

2. The GPU and CPU connect topology also need to be considered for SPECAccel to complete the manual binding. The following NVIDIA utility is used here:

   ```
   nvidia-smi topo-m
   ```

   Figure 20 shows that the command displays the CPU cores connected to the NVIDIA V100 GPU.

```
# nvidia-smi topo -m
        GPU0    CPU Affinity
GPU0     X      0-15

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks
```

Figure 20 Output of the nvidia-smi command

3. With this information about the CPU, memory, and GPU topologies, we can use numactl with the `-C` parameter to assigned CPU cores and the `-m` parameter for the memory domain, followed by the application we plan to run. The syntax is as follows:

   ```
   # numactl -C x -m y APP
   ```

   where:

   - `x` is the CPU number in the range 0-15 [CPU Affinity= 0~15]
   - `y` is 0 [node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
   - `APP` is the application we want to bind on GPU0

Figure 21 demonstrates a 4.2% performance advantage with the correct binding versus an incorrect binding on OpenACC. In fact, performance could be worse on other SPECAccel-like applications if the wrong binding is applied.

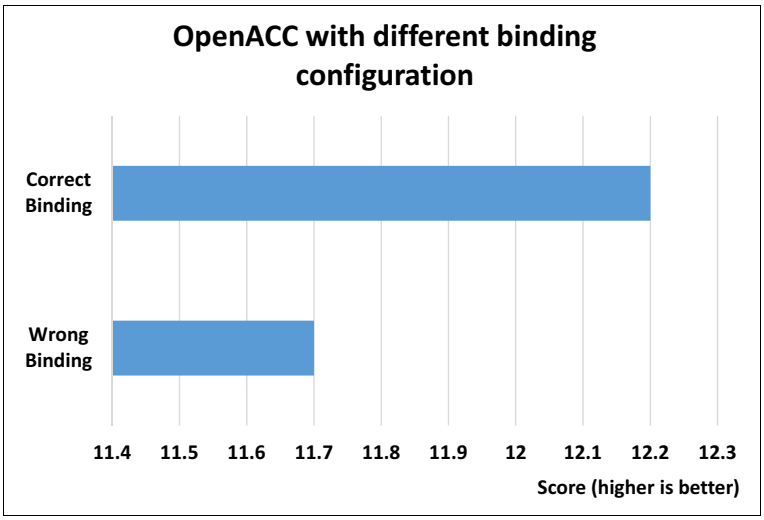

Figure 21 OpenACC performance comparison with different binding configurations

Enable Persistence Mode for the GPU

Enabling Persistent Mode on the NVIDIA GPU prevents the driver from releasing the device state when the device is not in use, which can improve the startup time and reduce the latency when new workload is going to run on the GPU.

NVIDIA provides the following nvidia-smi command to setup persistence mode:

```
# nvidia-smi -pm X
```

Where:

- X = 1 means that Persistent Mode is enabled
- X = 0 means that Persistent Mode is disabled

For example, to enable PM:

```
# nvidia-smi -pm 1
Enabled persistence mode for GPU 00000000:2F:00.0.
All done.
```

Figure 22 and Figure 23 demonstrate the performance advantage when Persistent Mode (PM) is enabled (PM=1) vs disabled (PM=0) on each sub-benchmark of OpenACC and OpenCL, respectively. The green line shows the delta between the orange bar (PM disabled) and the blue bar (PM enabled).

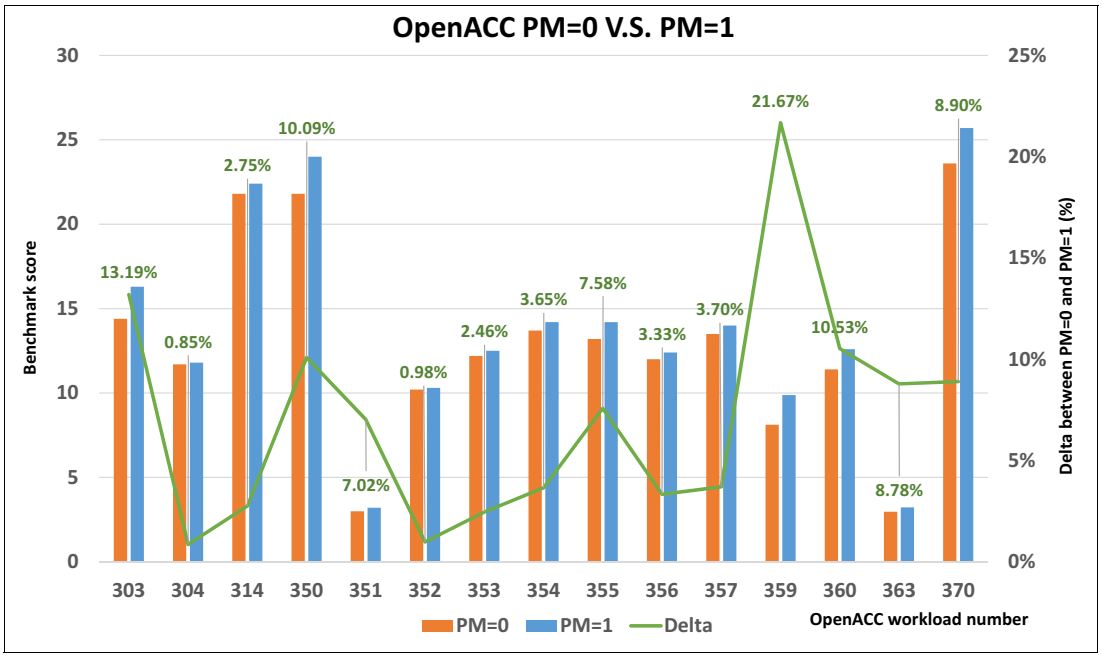

Figure 22 OpenACC performance comparison with different Persistence Modes

Tip: The x-axis in the figure corresponds to the OpenACC workloads as defined in the benchmark. See Table 2 on page 6 for a summary.

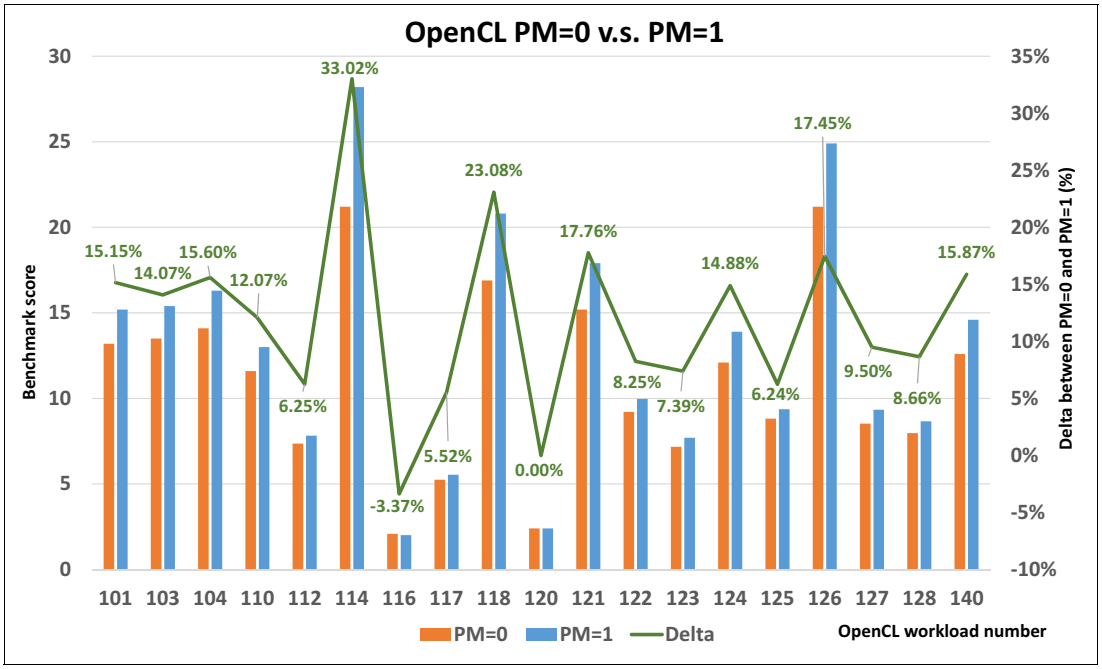

Figure 23 OpenCL performance comparison with different Persistence Modes

Tip: The x-axis in the figure corresponds to the OpenCL workloads as defined in the benchmark. See Table 3 on page 6 for a summary.

Performance world records

Applying all the tuning steps mentioned here, the ThinkSystem SR650 set a new performance world record on SPECAccel with both OpenACC and OpenCL offloading APIs.

The Lenovo benchmark results are described at the following pages:

- OpenACC benchmark report:

  https://lenovopress.com/lp1103-sr650-1-node-specaccel-acc-2019-04-02
- OpenCL benchmark report:

  https://lenovopress.com/lp1104-sr650-1-node-specaccel-ocl-2019-04-02

Conclusion

When workload processing is offloaded from the server processor to an accelerator such as a GPU, the way a system is tuned for maximum performance must also be changed. In addition to traditional analysis on CPU frequency, utilization, CPI (cycle per instruction), memory bandwidth, and footprint, the accelerator and its connection bandwidth also need to be considered to identify the obstacles to system performance.

Author

Jimmy Cheng is a performance engineer and Lenovo SPEC HPG representative in the Lenovo Data Center Group Laboratory located at Taipei, Taiwan. Jimmy joined Lenovo in 2016. Prior to this, he worked on IBM POWER as a system assurance and validation engineer, as the ATCA system integration engineer and automation developer at Advantech, and he focused on network performance when he worked at Hewlett Packard Enterprise. Jimmy holds a Master's Degree in Electronic and Computer Engineering from National Taiwan University of Science and Technology in Taiwan, and a Bachelor's Degree in Computer Science and Engineering from Yuan-Ze University, Taiwan.

Thanks to the following people for their contributions to this project:

- David Watts, Lenovo Press
- Mark T. Chapman, Lenovo Data Center Group

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document was created or updated on May 24, 2019.

Send us your comments via the **Rate & Provide Feedback** form found at
http://lenovopress.com/lp1146

Trademarks