LEADERSHIP PERFORMANCE WITH 2ND-GENERATION INTEL® XEON® SCALABLE PROCESSORS

New Features and Tools to Maximize Your HPC, AI, and Analytics Applications

Amarpal S. Kapoor, Technical Consulting Engineer; Rama Kishan V. Malladi, Performance Modeling Engineer; and Avinash Karani and Nitya Hariharan, Application Engineers; Intel Corporation

April 2019 saw the launch of the 2nd-generation **Intel® Xeon® Scalable processor** (formerly codenamed Cascade Lake), a server-class processor. This new processor family has already set 95 performance world records, earning performance leadership[1]. New features include Intel® Deep Learning Boost (Intel® DL Boost) for AI deep learning inference acceleration and support for **Intel® Optane™ DC** (data center) persistent memory. These processors will continue to deliver leadership performance with up to 56 cores per CPU socket and 12 DDR4 memory channels per socket—making them ideal for a wide variety of HPC, AI, and analytics applications with high-density infrastructures.

The Intel Xeon Scalable processor is designed to address a range of compute needs and demands, including more than 50 workload-optimized solutions and a variety of custom processors. The 8200 series offers up to 28 cores (56 threads), while the 9200 series has up to 56 cores (112 threads). Each processor core has a 1MB dedicated L2 cache and a non-inclusive shared L3 cache of up to 38.5 MB. On each socket, there are up to three Intel® Ultra Path Interconnect (Intel® UPI) links operating at 10.4 GT/s for cross-die (multi-socket) communication. The processor memory interface now supports up to six channels (on the 8200 series) and 12 channels (on the 9200 series) of DDR4 memory, operating at 2,933 MT/s. Also, the processor supports up to 4.5 TB of memory per socket using the Intel Optane DC persistent memory modules. To help improve the performance of DL applications, Intel Xeon Scalable processors have 512-bit VNNI (vector neural network instructions), which help in processing up to 16 DP/32 SP/128 INT8 MAC (multiply accumulate) instructions per cycle per core. To address some side-channel security issues, Intel Xeon Scalable processors implement hardware mitigations, which have smaller overhead compared to software-based methods[2]. These processor features apply in multiple computational domains, some of which we'll discuss below.

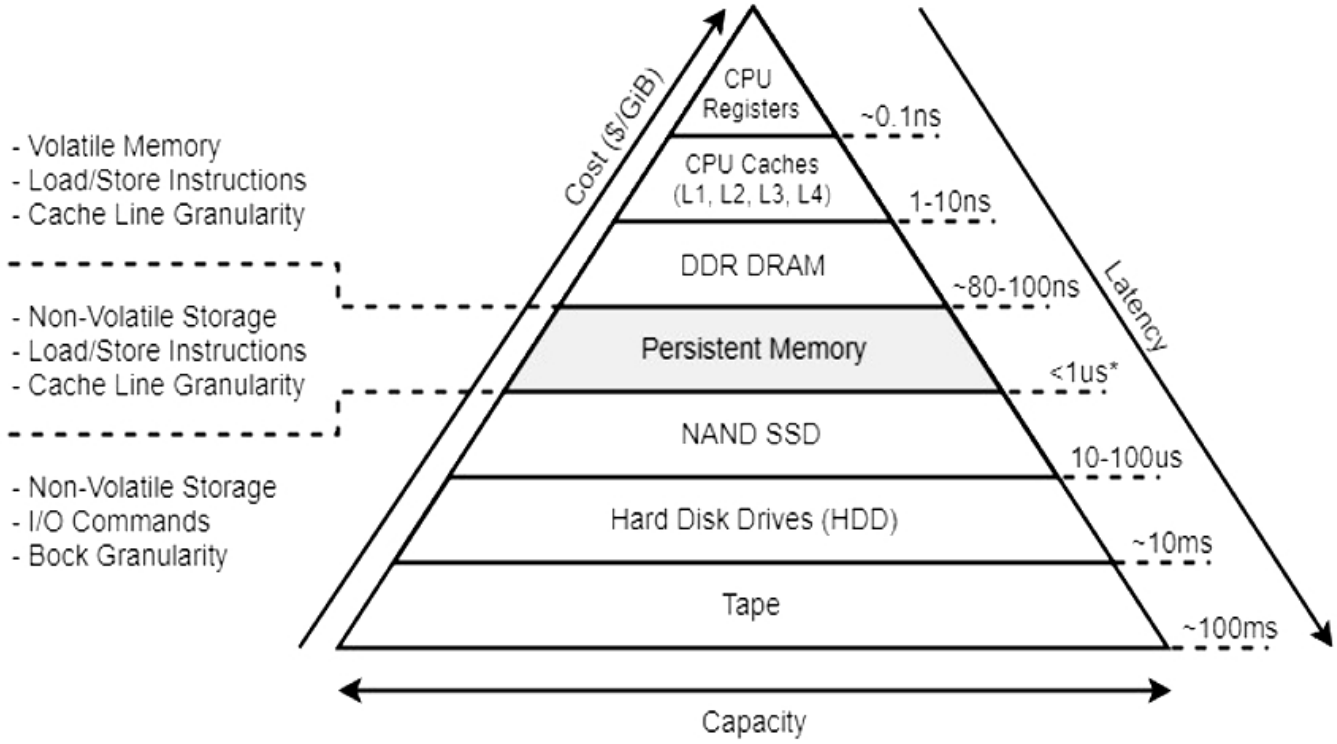
We'll also discuss working with Intel Optane DC persistent memory, Intel® AVX-512 Vector Neural Network Instructions (VNNI) for faster DL inference, and relative performance gains achieved in HPC applications on the Intel Xeon Scalable processor.

[Editor's note: We discuss ways to determine how applications can best utilize this new memory in **Using the Latest Performance Analysis Tools to Prepare for Intel® Optane™ DC Persistent Memory** in this issue.]

Intel® Optane™ DC Persistent Memory

Intel Optane DC persistent memory is a new type of non-volatile, high-capacity memory with near-DRAM latency, offering affordable, high-capacity data persistence. **Figure 1** shows latency estimates for different classes of memory and storage devices. Note the new tier that Intel Optane DC persistent memory creates between SSD and conventional DRAM.

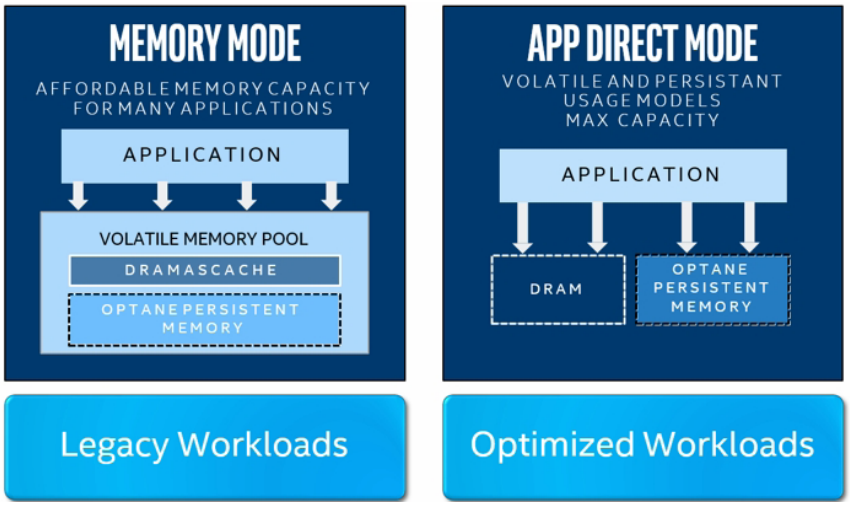
Intel Optane DC persistent memory is available in the same form factor as DRAM and is both physically and electrically compatible with DDR4 interfaces and slots. Intel Xeon Scalable processor-based machines must be populated with a combination of DRAM and Intel Optane DC persistent memory. (It's not possible to just have Intel Optane DC persistent memory on an Intel Xeon Scalable processor-based machine, since DRAM is necessary to serve system activities.)



1 **Latency estimates for different storage and memory devices**

Intel Optane DC persistent memory can be used in two different modes (**Figure 2**):

- Memory mode
- App Direct mode



2 **Modes of operation for Intel® Optane™ DC persistent memory**

Memory Mode

This is the simplest mode for using Intel Optane DC persistent memory, since existing applications can benefit without any source changes. In this mode, a new pool of volatile memory becomes visible to the operating system and user applications. The DRAM acts as a cache for hot (frequently accessed) data, while Intel Optane DC persistent memory provides a large volatile memory capacity. Memory management is handled by the Intel Xeon Scalable processor memory controller. When data is requested from memory, the memory controller first checks the DRAM cache. If data is found, the response latency is identical to DRAM latency. If data isn't found in the DRAM cache, it's read from Intel Optane DC persistent memory, which has higher latency. Memory controller prediction mechanisms aid in delivering better cache hit rates by fetching the required data in advance. However, workloads with random access patterns over a wide address range may not benefit from the prediction mechanisms and would experience slightly higher latencies compared to DRAM latency[3].

App Direct Mode

For data to persist in memory, Intel Optane DC persistent memory should be configured for use in App Direct mode. In this mode, the operating system and user applications become aware of both DRAM and Intel Optane DC persistent memory as discrete memory pools. The programmer can allocate objects in either of the memory pools. Data that needs to be fetched with the least latency must be allocated in DRAM (this data will be inherently volatile). Large data, which might not fit in DRAM, or data that needs to be persistent, must be allocated in Intel Optane DC persistent memory. These new memory allocation possibilities are the reason behind the need for source code changes in App Direct mode. Interestingly, in App Direct mode, it's also possible to use Intel Optane DC persistent memory as a faster storage alternative to conventional HDD/NVMe storage devices.

Configuration

Switching between Memory and App Direct modes requires changes in BIOS settings. `ipmctl` is an open source utility available for configuring and managing Optane persistent memory modules (PMM)[4]. Here are some useful management commands:

```
$ ipmctl show -topology
$ ipmctl show -memoryresources
```

Provisioning PMMs is a two-step process. First, you specify a goal and store it on the PMMs. Then the BIOS reads it at the next reboot.

Memory mode
```
$ ipmctl create -goal MemoryMode=100
```

App Direct mode
```
$ ipmctl create -goal PersistentMemoryType=AppDirect
```

Mixed mode
```
$ ipmctl create -goal MemoryMode=60
```

In the Mixed mode, a specified percentage of Intel Optane DC persistent memory can be used in memory mode and the remaining memory can be used in App Direct mode. The command above will assign 60% of available persistent memory in memory mode and 40% in App Direct mode. (For details, see references 4 and 5.)

Persistent Memory Development Kit (PMDK)

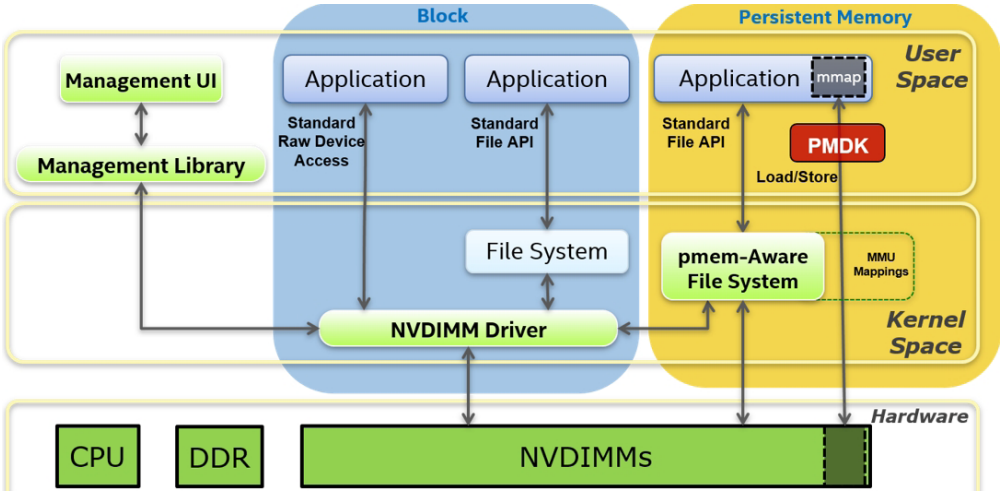
Applications can access persistent, memory-resident data structures in place, as they do with traditional memory, eliminating the need to page blocks of data back and forth between memory and storage. Getting this low-latency direct access requires a new software architecture that allows applications to access ranges of persistent memory[6]. The Storage Network Industry Association (SNIA) Programming Model comes to our rescue here, as shown in **Figure 3**.

PMDK is a collection of libraries and tools that system administrators and application developers can use to simplify managing and accessing persistent memory devices. These libraries let applications access persistent memory as memory-mapped files. **Figure 3** shows the SNIA model, which describes how applications can access persistent memory devices using traditional POSIX standard APIs such as `read`, `write`, `pread`, and `pwrite`, or load/store operations such as `memcpy` when the data is memory-mapped to the application. The Persistent Memory area represents the fastest possible access because the application I/O bypasses existing filesystem page caches and goes directly to or from the persistent memory media[6].

PMDK contains the following libraries and utilities to address common programming requirements with persistent memory systems:

PMDK libraries
`Libpmem`
`Libpmemobj`
`Libpmemblk`
`Libpmemlog`
`Libvmem`
`Libvmmalloc`
`Libpmempool`
`Librmem`
`Libvmemcache`

PMDK libraries
`pmempool`
`pmemcheck`



3 **SNIA Programming Model[6]**

PMDK Example

This section demonstrates the use of PMDK through the `libpmemobj` library, which provides a transactional object store, providing memory allocation, transactions, and general facilities for persistent memory programming. This example demonstrates two applications:

- **`writer.c`**, which writes a string to persistent memory
- **`reader.c`**, which reads that string from persistent memory

Code snippets with comments are shown in **Table 1**. The complete source for this example is available in reference 7.

Table 1. Working with strings in persistent memory

writer.c	reader.c

`int main(int argc, char *argv[1]`
`{`
`PMEMobjpoo] *pop = pmemobj_create(argv[1],`
`LAYOUT_NAME,`
`PMEMOB]_MIN_POOL, 0666),`
`if (pop == NULL) {`
`Perror("pmemobj_create");`
`return 1;`
`}`
`...`
`...`
`pmemobj_close(pop);`
`return 0;`
`}`
	`int main(int argc, char *argv[1]`
	`{`
	`PMEMobjpoo] *pop = pmemobj_open(argv[1],`
	`LAYOUT_NAME,`
	`if (pop == NULL)`
	`{`
	`perror("pmemobj_open");`
	`return 1;`
	`}`
	`...`
	`...`
	`pmemobj_close(pop);`
	`return 0;`
	`}`

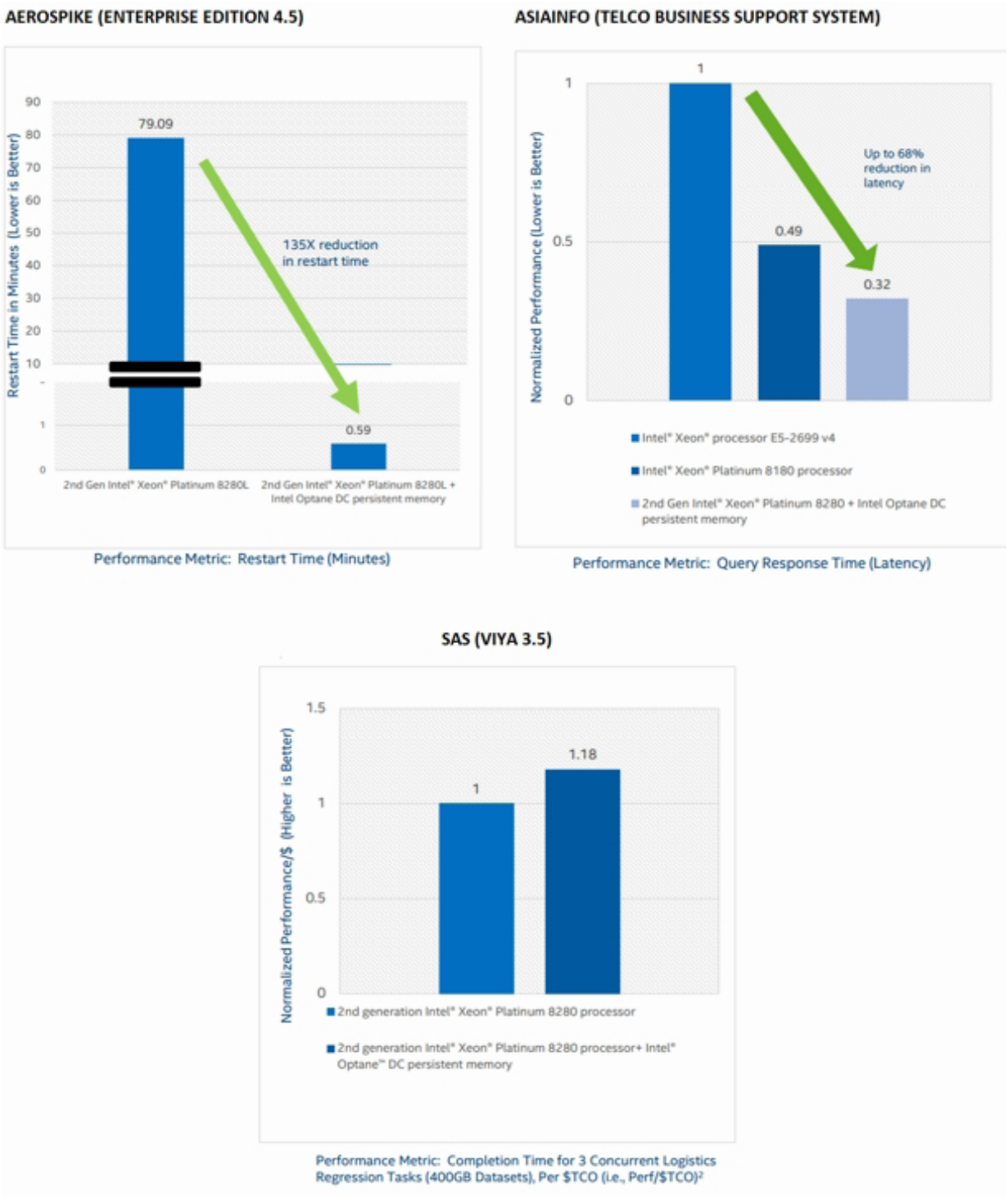
The `pmemobj_create` API function takes the usual parameters you would expect for a function creating a file plus a layout, which is a string of your choosing that identifies the pool.	In the reader, instead of creating a new pool, we open the pool we have created in the writer code using the same layout.
`PMEMoid root = pmemobj_root(pop, sizeof (struct my_root));` `struct my)root *rootp = pmemobj_ direct(root);`	`PMEMoid root = pmeobj_root(pop, sizeof (structr my_root));` `struct my_root *rootp = pmeobj_direct(root);`
It is required to keep a known location for the application in the memory pool, called the root object. It is the anchor to which all the memory structures can be attached. In the above code, we are creating a root object using `pmemobj_root` in the `pop` memory pool. We are also translating the `root` object to a usable, direct pointer using `pmemobj_direct`.	Since we already created the root object in the pool, **`pmeobj_root`** returns the root object without initializing it with zeros. It will contain whatever string the writer was tasked with storing.
`char buf[MAX_BUT_LEN];` `scanf("X9s", buf):`	`if (rootp->len == strlen(rootp->buf))` `printrf("%s\n", rootp->buf);`
A maximum of 9 bytes are then read to the temporary buffer.	The above section reads the string from persistent memory.

`root->len = strlen(bvuf);`
`pmemmobj_persistr(pop, &rootrs->len, sizeof`
`(rootrp->len));`
`pmemobvj_memcpy_persistr(pop, rootp->buf,`
`my_buf, rootp->len);`	

In the above section, we force any changes in the range (&rootp->len &rootp->len+sizeof(rootp->len))to be stored durably in persistent memory using pmeobj_persist and we copy the string from local buffer to persistent memory using pmeobj_ memcopy_persist.	

Performance Gains from Intel Optane DC Persistent Memory

This section presents the performance gains achieved in enterprise applications like Aerospike*, Asiainfo's benchmark, and SAS VIYA* using Intel Optane DC persistent memory (**Figure 4**). Aerospike is a NoSQL* key-value database application which saw a 135x reduction in restart times with the use of Intel Optane

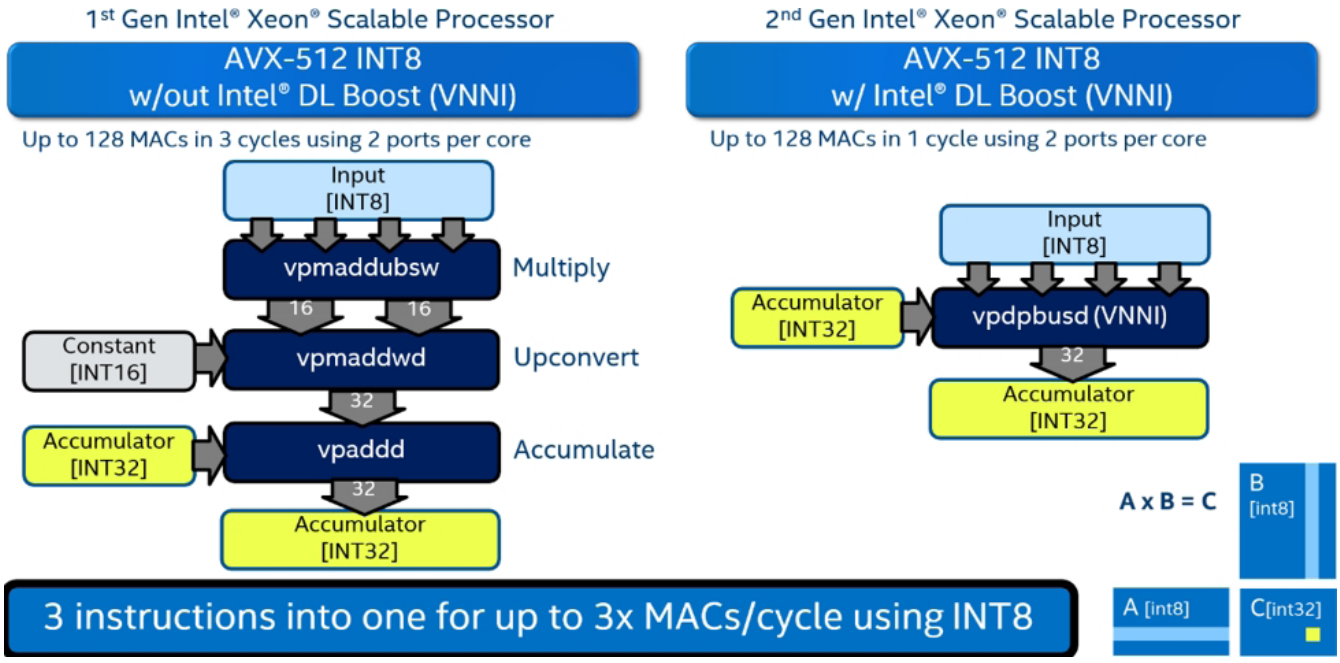


4 **Better performance in enterprise applications using Intel Optane DC persistent memory**

DC persistent memory in App Direct mode. This helps Aerospike restart in seconds instead of hours, allowing for more frequent software and security updates, while significantly reducing disruption. One of Asiainfo's benchmarks saw a 68% reduction in latency due to the combined effects of Intel Xeon Scalable processors and Intel Optane DC persistent memory in App Direct mode. The performance gains were attributed to the ability to store more data in memory with reduced spillover to slower SSDs. SAS VIYA* is a unified, open analytics platform with AI capabilities deployed on the cloud. Using Intel Optane DC persistent memory mode, larger datasets needed for gradient boosting models could be placed in memory, with little or no performance degradation, at reduced costs. The performance gain was up to 18%.

Faster AI Inference with Intel® AVX-512 VNNI

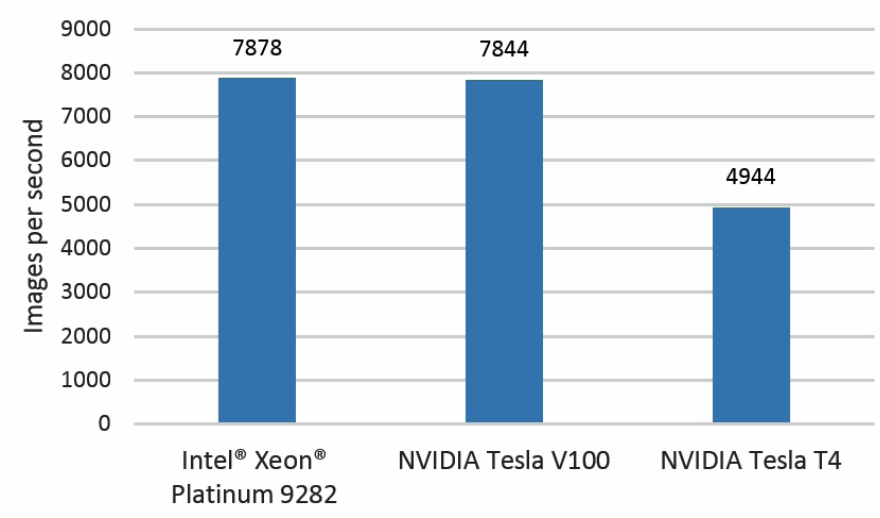
Neural networks require several matrix manipulations, which may be realized using MAC instructions. In the previous generation of Intel Xeon Scalable processors, multiplying two 8-bit (INT8) values and accumulating the result to 32 bits required three instructions. In the latest generation of Intel Xeon Scalable processors, this is done in one instruction[8]. This instruction count reduction represents a performance gain, which is accomplished by having simultaneous execution of the MAC instructions on both port 0 and 5 of the execution pipeline (**Figure 5**).



5 **Intel® DL Boost technology**

Currently, Intel® compilers support VNNI instructions through intrinsics and inline assembly only. For users intending to leverage VNNI capabilities without using intrinsics or assembly, **Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)**[9] and **BigDL**[10] are the recommended alternatives. Intel MKL-DNN is a collection of highly optimized DL primitives for traditional HPC environments, while BigDL (powered by Intel MKL-DNN) provides similar optimized DL capabilities for big data users in Apache Spark*.

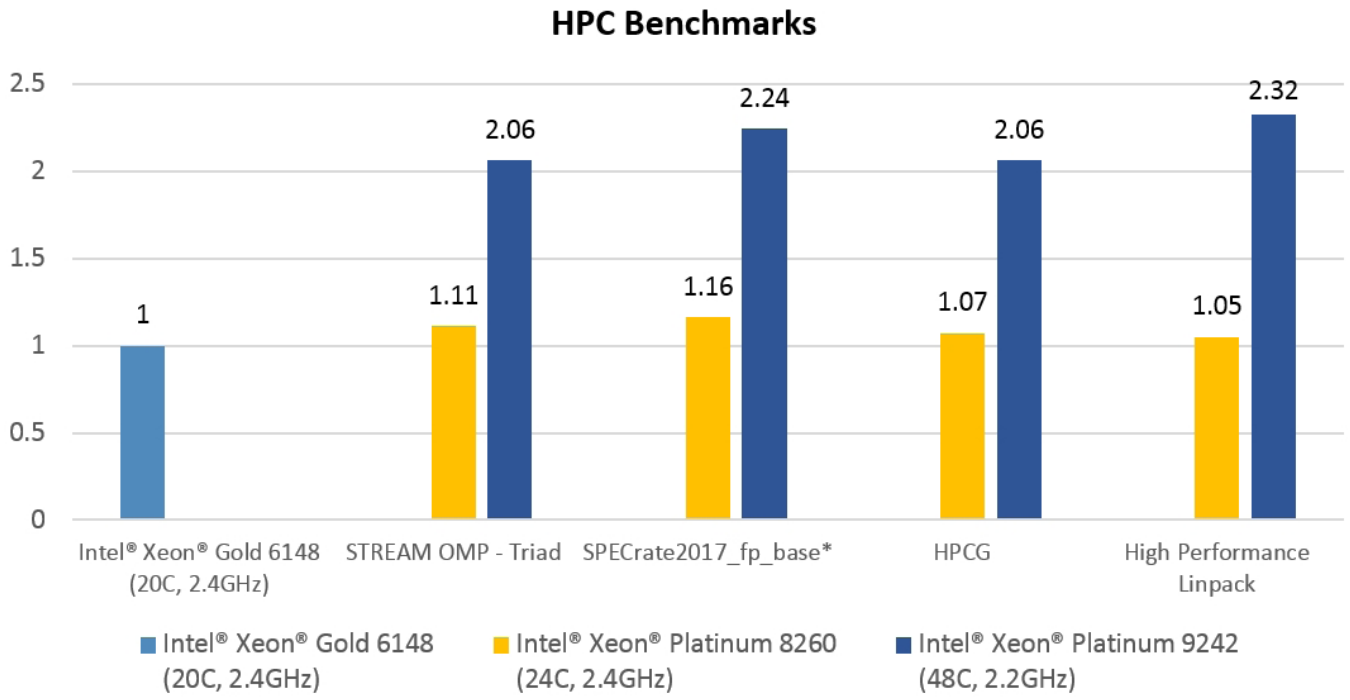
Caffe* 1.1.3 optimized with Intel MKL-DNN gives 14x better inference throughput on a dual-socket Intel® Xeon® Platinum 8280 processor and 30x better inference throughput on a dual-socket Intel Xeon Platinum 9282 processor, in comparison to the previous-generation Intel Xeon Scalable processors[11]. A similar study with Intel Xeon Platinum 9282 processor for Caffe ResNet-50* demonstrated better inference throughput than NVIDIA* GPUs (**Figure 6**)[12]. Other popular frameworks like Chainer*, DeepBench*, PaddlePaddle*, and PyTorch* also use Intel MKL-DNN for better performance.



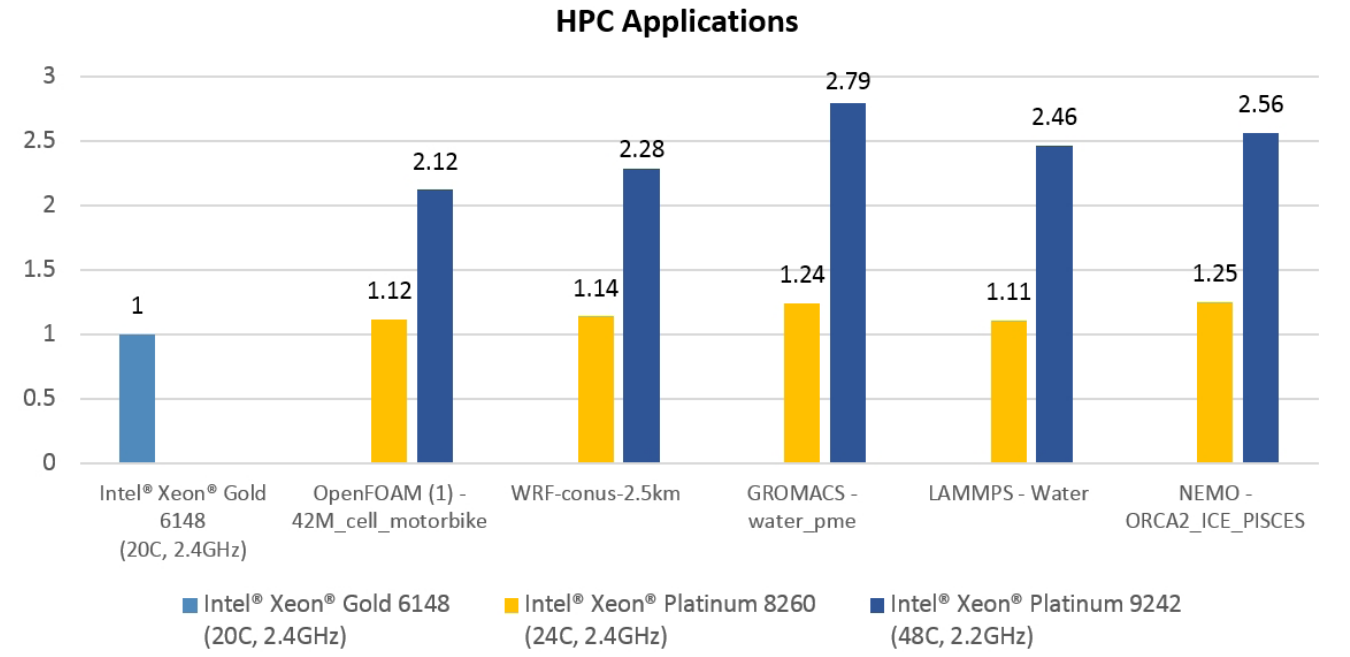
6 **CPU-GPU inference throughput comparison for Caffe ResNet-50* (higher is better)**

HPC Application Performance on Intel Xeon Scalable Processors

The increased core count and higher bandwidth available on Intel Xeon Scalable 8200 and 9200 processors provide substantial gain for HPC applications. Performance gains are shown for industry standard benchmarks in **Figure 7**, and for real-world applications in **Figure 8**. LAMMPS* and GROMACS* benefit from AVX-512, higher core count, and hyperthreading. The higher available bandwidth on Intel Xeon Scalable processors shows a positive gain for memory bandwidth-bound codes like OpenFOAM*, WRF*, and NEMO*.



7 **Relative performance of HPC industry standard benchmarks**



8 **Relative performance of HPC applications**

OpenFOAM Disclaimer: This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software via **www.openfoam.com**, and owner of the OPENFOAM® and OpenCFD® trademark.

Improving HPC, AI, and Analytics Application Performance

New hardware features in the latest Intel Xeon Scalable processors enable developers to improve performance for a wide variety of HPC, AI, and analytics applications. Intel continues to innovate in processor technologies. The upcoming Cooper Lake architecture will introduce bfloat[16] for enhanced AI training support. Also, Intel recently released 10th-generation Intel® Core™ processors on 10nm, delivering better performance and density improvements.

References

1. **Data-Centric Business Update Media Briefing**
2. **Addressing Hardware Vulnerabilities**
3. **Intel® Optane™ DC Persistent Memory Operating Modes Explained**
4. **ipmctl for Intel Optane DC Persistent Memory**
5. **Configure Intel® Optane™ DC Persistent Memory Modules on Linux***
6. **Persistent Memory Development Kit**
7. **PMDK on GitHub**
8. **Intel® Architecture Instruction Set**
9. **Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)**
10. **BigDL: Distributed Deep Learning Library for Apache Spark***
11. **2nd-Generation Intel® Xeon® Scalable Processors, Solutions**
12. **Intel® CPU Outperforms NVIDIA* GPU on ResNet-50 Deep Learning Inference**

Appendix

Configuration: Single-Node Intel® Xeon® Generational HPC Performance

Intel® Xeon® 6148 processor: Intel Reference Platform with 2S 6148 Intel processors (2.4GHz, 20C), 12x16GB DDR4-2666, 1 SSD, Cluster File System: Panasas (124 TB storage) Firmware v6.3.3.a & OPA based IEEL Lustre, BIOS: SE5C620.86B.00.01.0015.110720180833, Microcode: 0x200004d, Oracle Linux Server release 7.6 (compatible with RHEL 7.6) on a 7.5 kernel using ksplice for security fixes, Kernel: 3.10.0-862.14.4.el7.crt1.x86_64, OFED stack: OFED OPA 10.8 on RH7.5 with Lustre v2.10.4.

Intel® Xeon® Platinum 8260 processor: Intel Reference Platform with 2S 8260 Intel processors (2.4GHz, 24C), 12x16GB DDR4-2933, 1 SSD, Cluster File System: Panasas (124 TB storage) Firmware v6.3.3.a & OPA based IEEL Lustre, BIOS: SE5C620.86B.0D.01.0286.011120190816, Microcode: 0x4000013, Oracle Linux Server release 7.6 (compatible with RHEL 7.6) on a 7.5 kernel using ksplice for security fixes, Kernel: 3.10.0-957.5.1.el7.crt1.x86_64, OFED stack: OFED OPA 10.9 on Oracle Linux 7.6 (Compatible w/RHEL 7.6) w/Lustre v2.10.6.

Intel® Xeon® Platinum 9242 processor: Intel Reference Platform with 2S Intel Xeon 9242 processors (2.2GHz, 48C), 24x16GB DDR4-2933, 1 SSD, Cluster File System: 2.12.0-1 (server) 2.11.0-14.1 (client), BIOS: PLYXCRB1.86B.0572.D02.1901180818, Microcode: 0x4000017, CentOS 7.6, Kernel: 3.10.0-957.5.1.el7.x86_64

Intel® Xeon® 6148/8260/9242 Processors

Application	Workload	Intel® Compiler	Intel® Software CoSoftware	BIOS Settings
STREAM OMP 5.10	Triad	2019u2		HT=ON, Turbo=OFF, 1 thread per core
HPCG 2018u3	Binary included MKL	2019u1	MPI 2019u1, MKL 2019u1	HT=ON, Turbo=OFF, 1 thread per core
SPECrate2017 _ fp_base	Best published result as of June 20, 2019: • **6148** • **8260** • **9242**			
HPL 2.1	Binary included MKL	2019u1	MKL 2019, MPI 2019u1	HT=ON, Turbo=OFF, 2 threads per core
WRF 3.9.1.1	conus-2.5km	2018u3	MPI 2018u3	HT=ON, HT=ON, 1 threads per core
GROMACS 2018.2	All workloads	2019u2	MKL 2019u2, MPI 2019u2	HT=ON, Turbo=OFF, 2 threads per core
LAMMPS 12 Dec 2018	All workloads	2019u2	MPI 2019u2	HT=ON, Turbo=ON, 2 threads per core
OpenFOAM 6.0	42M_cell_motorbike	2019u1	MPI 2018u3	HT=ON, Turbo=OFF, 1 thread per core
NEMO v4.0	ORCA2_ICE_PISCES	2018u3	MPI 2018u3	HT=off, TURBO=ON, pure MPI run