Opportunities for Partitioning Non-Volatile Memory DIMMs between Co-scheduled Jobs on HPC Nodes

Brice Goglin, Andrès Rubio Proaño

Opportunities for Partitioning Non-Volatile Memory DIMMs between Co-scheduled Jobs on HPC Nodes

Brice Goglin and Andrès Rubio Proaño
Inria, LaBRI, Univ. Bordeaux – France
brice.goglin@inria.fr, andres.rubio@inria.fr

Abstract

The emergence of non-volatile memory DIMMs such as Intel Optane DCPMM blurs the gap between usual volatile memory and persistent storage by enabling byte-accessible persistent memory with reasonable performance. This new hardware supports many possible use cases for high-performance applications, from high performance storage to very-high-capacity volatile memory (terabytes). However the numerous ways to configure the memory subsystem raises the question of how to configure nodes to satisfy applications' needs (memory, storage, fault tolerance, *etc.*).

We focus on the issue of partitioning HPC nodes with NVDIMMs in the context of co-scheduling multiple jobs. We show that the basic NVDIMM configuration modes would require node reboots and expensive hardware configuration. Moreover it does not allow the co-scheduling of all kinds of jobs, and it does not always allow locality to be taken into account during resource allocation.

Then we show that using 1-Level-Memory and the Device DAX mode by default is a good compromise. It may be easily used and partitioned for storage and memory-bound applications with locality awareness.

Keywords: Non Volatile Memory DIMM; NVDIMM; DAX; Partitioning; Co-scheduling; Locality.

1 Introduction

Computing nodes are increasing complex, with tens of cores. Co-scheduling multiple jobs on such nodes is a useful strategy for making sure all powered-on cores are used in HPC centers. However sharing nodes between multiple jobs also comes with issues such as contention in the memory subsystem or cache pollution. Resource partitioning is an interesting way to avoid such issues thanks to operating system features such as Linux Cgroups.

The emergence of non-volatile memory DIMMs such as recently announced Intel Optane DC Persistent Memory brings new possible strategies for data management in HPC applications. Indeed they support multiple hardware and software configurations spanning from huge volatile capacities to high-performance storage, that may be used as burst buffers or for recovery after fault.

We focus in this paper on the co-scheduling of jobs with different needs, and on the partitioning of these new hardware resources between them. We compare the possible hardware configurations and advocate for the use of the 1-Level-Memory mode with namespaces and explicit NUMA memory management.

The rest of this paper is organized as follows. We present the upcoming NVDIMM hardware in Section 2 and discuss its possible hardware and software configurations. Co-scheduling jobs with different requirements is then discussed in Section 3 before we explain how to partition resources between them in Section 4. Before concluding, related works are discussed in Section 5.

2 Background

Non-volatile memory DIMMs is a promising emerging technology that is expected to blur the longstanding separation between usual volatile memory and persistent storage [5]. It supports both with good performance and offers multiple ways to be used by software.

2.1 Hardware

Non-volatile memory DIMMs have been available for several years as DDR DIMMs with a battery so as to save data to a flash backup on power loss. However software support was not ready until recently. Intel recently announced the availability of Optane DataCenter Persistent Memory Module (DCPMM) and competitors are working on offering similar technologies in the near future. These memory DIMMs are inserted in usual memory slots just like normal DIMMs (DDR) as depicted in Figure 1.

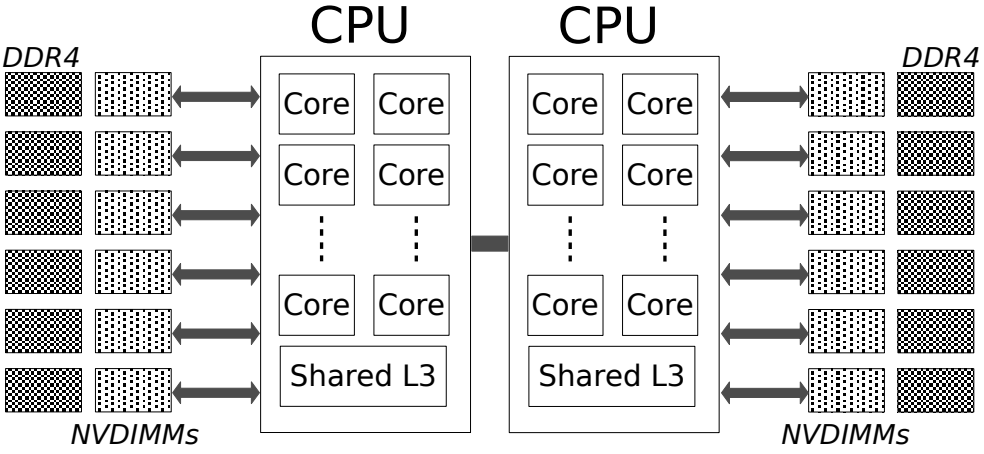

Figure 1: Dual-socket Xeon platform with 6 channels per processor, with one Optane DCPMM and one DDR each.

Optane DCPMMs can be configured as individual *Regions* or as *Interleaved Regions*. Interleaving implies that the entire region data is lost whenever a single NVDIMM fails. However, interleaving is still expected to be used by default because it increases the memory bandwidth by using multiple channels simultaneously. Non-interleaved regions are expected to be useful for separating small, independent jobs such as virtual machines.

Besides regions, Intel hardware introduces in latest Xeon processor (*Cascade Lake*) a way to use Optane DCPMM as normal (volatile) memory [2]. Each DCPMM can be partitioned between *Memory Mode* (to be used as a large pool of volatile memory) and *App Direct* (for persistent stor-

age) [14]. This configuration is performed in the BIOS or using tools such as `ipmctl` and requires a reboot.

2.2 Memory Mode and 2-Level-Memory

Latest Xeon processors may be configured in *2-Level-Memory* mode (2LM). It exposes the *Memory Mode* part of NVDIMMs as volatile memory and uses DDR as a *Memory-side Cache* in front of it, as shown in Figure 2. This mode is convenient for applications that require lots of RAM (up to 6x 512GB DCPMM per socket with current hardware).

Figure 2: 2-Level-Memory mode (2LM) uses DDR as a *Memory-side Cache* in front of the *Memory Mode* part of NVDIMMs exposed as normal volatile memory.

Unfortunately this mode does not bring the exact same performance as a pure DDR [8]. One reason is that each DDR cache is direct-mapped, which is known to perform inconsistently over time [12].[1]

In this 2LM mode, the *App Direct* part of NVDIMMs is exposed as storage just like in 1LM mode. We detail this storage mode in the next section.

2.3 App Direct and 1-Level-Memory for Storage

Latest Xeon processors may also be configured in *1-Level-Memory* mode (1LM) which puts back DDR as the main volatile memory as show on Figure 3. The *Memory Mode* part of NVDIMMs is not usable anymore. The *App Direct* part is exposed as a *Persistent Memory Regions* (called *region* in the reminder of this paper) that may be used as a disk (e.g. `/dev/pmem1`). However this disk is directly byte-accessible by the processor. Contrary of usual disks, there is no need to copy disk blocks in memory (in the kernel page-cache) before actually accessing those bytes. This mode is called DAX (*Direct Access*) in Linux and Windows. It enables the mapping of the actual backend data directly in application virtual memory and the use of load and stores. This avoids the need for intermediate copy and page-cache allocations.

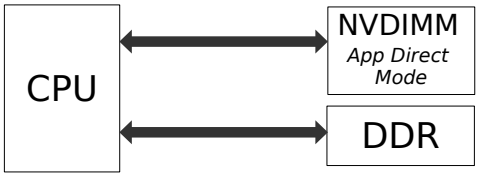

Figure 3: 1-Level-Memory mode (1LM) uses DDR as the main memory while NVDIMMs are exposed as a persistent memory region that is usually used as storage.

[1] Linux kernel version 5.2 will mitigate this issue by shuffling the list of free pages. `https://lkml.org/lkml/2019/2/1/15`

1-Level-Memory has several interesting use cases for HPC applications, ranging from local disks as burst buffers [7], to recovering memory contents after a fault thanks to persistence. When a modern filesystem is used to store data in the App Direct part of NVDIMMs, the region is actually configured in *FSDAX* mode in Linux (*File System DAX*). Applications may use these files as usual. However, optimal performance requires application to be modified for DAX: they should stop using explicit file access (read/write requires a copy) and rather map files in virtual memory instead (to directly access data).

Accessing the App Direct part of NVDIMMs (either in 1LM or 2LM) never goes through an intermediate DDR cache, hence performance is lower [8, 15, 1]. However 1LM latency is better than 2LM in case of cache-miss because there is no need to lookup the data in the DDR cache [11].

2.4 Device DAX and kmem additional NUMA nodes

Although FSDAX is expected to be used in the vast majority of cases because it exposes persistent storage as a normal filesystem, App Direct regions may also be useful without a filesystem. This mode is called *Device DAX* in Linux. It exposes a mmap'able linear space where applications may manually store their datasets without the structure and help of a file system. It was designed to expose large regions of non-volatile memories to specific applications such as virtual machines, but we are going to show in this paper that it is actually much more useful than this.

Device DAX requires significant rework of applications because they have to manually separate independent data without the help of independent files. However we explained in Section 2.3 that DAX requires applications to be rewritten to benefit from improved performance (map files instead of read/write). Hence we believe additional application changes for supporting Device DAX are not a significant hurdle.

Partitioning Device DAX between different jobs indeed requires synchronization between jobs. We will explain in Section 4.2 how resource managers may solve this issue using namespaces. Partitioning between different tasks of a job is where application developers will have to update their code to use different parts of a Device DAX for different datasets.

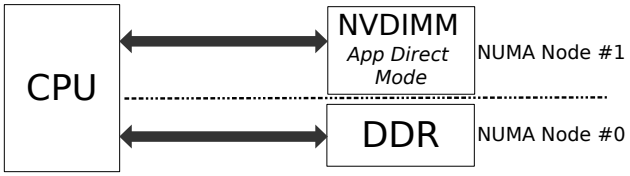

Figure 4: When the App Direct part of NVDIMMs is managed by the kmem device DAX driver in Linux, it appears as an additional NUMA node.

Device DAX brings an important feature since Linux 5.1: the *kmem* DAX driver can expose NVDIMM pages as an additional NUMA node where applications can allocate memory as usual [6], as depicted in Figure 4. This may be considered similar to Intel Xeon Phi *Flat* mode where both fast and slow memories are exposed as separate NUMA nodes.[2] It means that applications now have to manually allocate in one of the nodes depending on the required performance for each

[2]NVDIMMs 1LM and 2LM modes are similar to KNL *Flat* and *Cache* modes. However, NVDIMMs do not enable a KNL-like *Hybrid* mode: KNL could partition the fast memory (MCDRAM) between cache and normal memory. NVDIMMs rather allow partitioning the slow memory between cached (by the fast memory, DDR) and uncached.

dataset. This requires more work from application developers but provides more flexibility than 2LM and possibly higher performance [3]. Indeed developers have to choose between pure DDR (faster than NVDIMMs with DDR cache) and pure NVDIMM (slower) [8, 15].

2.5 Locality of NVDIMMs

NVDIMMs being attached to processors through memory slots, their access performance suffers from locality like normal DDR memory, *i.e.* accessing a NVDIMM is faster from the CPU where it is attached. Optane DCPMM performance being lower than DDR, one may expect these NUMA effects to be negligible. Unfortunately, there are actual higher [10] which means applications must take locality into account when choosing their target NVDIMMs.

This is obviously true for 1LM because NVDIMMs are accessed by the processor like DDR. But it is also true for 2LM because the DDR-cache acts as a *Memory-side Cache*: accesses to NVDIMMs of another CPU are cached in the DDR cache of that CPU, they are not cached locally.

3 Co-Scheduling Jobs with Memory and Storage Needs

Modern HPC nodes feature lots of cores and memory. They are therefore good candidates for co-scheduling several small jobs. Unfortunately node sharing raises multiple issues in terms of performance [13]. Hence we now explain how to partition nodes equipped with NVDIMMs.

3.1 Hardware Partitioning in 2LM

We explained in the previous section that partitioning is possible in most configurations. However we assumed the hardware configuration matches the job requirements. We now look at the case where some jobs want a 2LM configuration (Memory Mode for large amounts of volatile memory) and some others want 1LM (App Direct for persistent storage). The only way to have both Memory Mode and App Direct available at the same time in a machine is to configure the processors in 2LM (see Sections 2.2 and 2.3, and Figure 5).

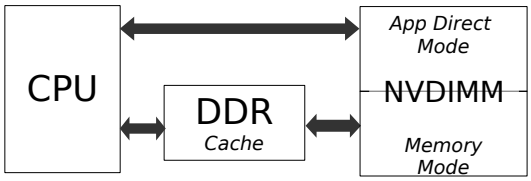

Figure 5: 2-Level-Memory enables exposing both Memory Mode as DDR-cached main memory and App Direct as storage.

However this configuration has major drawbacks: First, the administrators would have to choose a good ratio for NVDIMM partitioning between Memory Mode and App Direct. This ratio depends on the needs of all jobs that will be scheduled simultaneously on a node, and setting up the ratio requires a reboot of the node.[3]

[3] Additionally a 32G granularity seems to constrain possible ratios in hardware.

Secondly, locality issues arise as shown in Figure 6: If a socket is allocated to a 1LM job, its local NVDIMMs should be entirely set in App Direct. However it means there is no local memory anymore: both local DDR and NVDIMM cannot be used as volatile memory (DDR is entirely used as a cache; NVDIMMs are entirely used as App Direct). Cores of this socket would therefore use remote memory, which incurs bad performance as explained in Section 2.5.

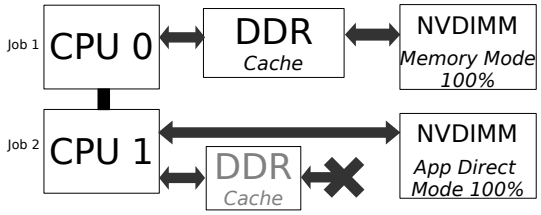

Figure 6: Allocating one socket to a job that wants 100% Memory Mode and the other socket to a job that wants 100% App Direct causes the latter to have no local memory anymore, and its DDR cache is useless.

In the end, we believe using 2LM to share a node with such different jobs is not a good idea and we do not expect significant improvements in future hardware platforms. Administrators would rather create one set of 1LM nodes and a separate set of 2LM nodes[4] and possibly reconfigure, reboot and move some nodes from one set to another depending on users' needs. However we will now show how 1LM may actually offer a more flexible solution.

3.2 Flexible Co-Scheduling with 1LM and kmem NUMA nodes

We explained in the previous section that 2LM requirements incur too many drawbacks. Therefore we propose not to use 2LM anymore. As explained in Section 2.4, Device DAX may be exposed as additional NUMA nodes. This provides lots of volatile memory that 2LM applications require but requires application developers to explicitly manage allocation between fast and slow memory. We believe that this additional work for developers is a good trade-off because of the flexibility it provides to users and administrators.

Hence we propose the following strategy:

1) NVDIMMs are configured 100% in App Direct and processors are in 1LM mode.

2) NVDIMM regions are configured as Device DAX by default in the Linux configuration (may be used for persistent storage as a single file).

3a) An application that cannot work without multiple files may request the reconfiguration of a region as FSDAX.[5]

3b) An application that needs lots of volatile memory may request the reconfiguration of a region as an additional NUMA node through the kmem driver.[6]

[4] This is similar to what happened in many KNL clusters: some nodes were in Cache mode, others in Flat mode.
[5] Using the `ndctl` command-line tool, which does not require a reboot.
[6] Using the `daxctl` command-line tool, which does not require a reboot.

This solution does not bring locality issues because each CPU still has its local DDR explicitly available, while its local NVDIMMs may be exposed in the mode that matches the local job needs. Besides, this approach is on par with current Linux kernel development towards exposing both DDR and PMEM as explicit NUMA nodes and having ways to migrate hot pages between fast and slow memory[7].

Table 1 summarizes the advantage of our proposal compared to 2LM memory presented in the previous Section.

Table 1: Advantages and Drawbacks of 2LM and 1LM modes for co-scheduling jobs.

CPU Config	**2-Level-Memory**	**1-Level-Memory**
NVDIMM Config	Memory Mode ratio depends on jobs Reboot required for updating	100% App Direct
Fast/Slow Memory Management	Automatic (DDR Cache)	Manual & Flexible (NUMA)
Storage Management	Limited to App Direct ratio	OK
Locality	May miss local memory	OK

4 Fine-Grain Partitioning between HPC Jobs

We showed in the previous section that 1LM is a good trade-off enabling flexibility with respect to application needs and memory management. We now explain how to actually partition and expose different kinds of memory between jobs at fine grain.

HPC resource managers may already use Linux *Cgroups* for partitioning CPUs between jobs [4], as well as NUMA nodes (individual nodes or amounts of memory may be dedicated to each group). This work may already be applied to partition NVDIMM-based NUMA nodes, either in 2LM or kmem nodes in 1LM.

However, when a single Device DAX is used, there is no way to partition it between multiple jobs. This is an issue that we will now address.

4.1 NVDIMM Hardware Partitioning

As explained in 2.1, each NVDIMM (its App Direct part) may be exposed as an individual region or it may be interleaved with others (see Figure 7). Each region is exposed as a different FSDAX, Device DAX or NUMA node in Linux, which may be allocated to different jobs by the administrator. However, with only 6 channels per CPU and 1 single DCPMM per channel (128, 256 or 512GB each), there are very few possibilities for partitioning. Moreover, modifying regions requires a long reconfiguration process (minutes) and a reboot. Hence we do not think this is a good way to partition NVDIMMs between jobs.

4.2 Multi-DAX and Namespace-based Software Partitioning

We believe that partitioning should rather be applied in software on top of persistent memory regions. Indeed, each region may be split into different *Namespaces* that are configured by the

[7] `https://lwn.net/Articles/787418/`

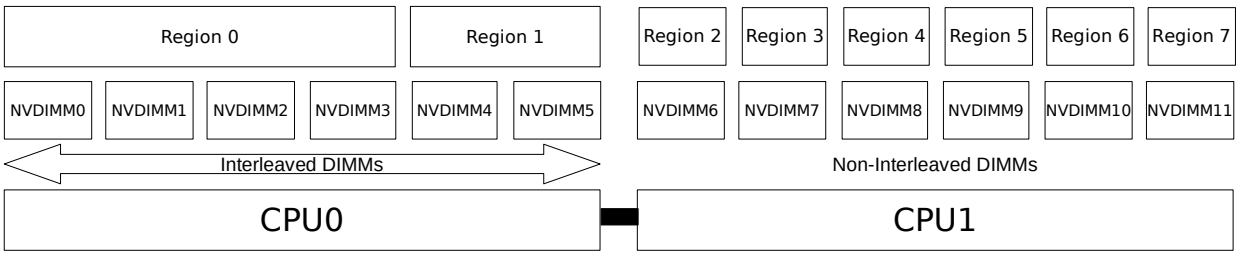

Figure 7: Partitioning NVDIMMs using Regions and Interleaving. On the first processor two interleaved regions use respectively 4 and 2 NVDIMMs. On the second processor, all NVDIMMs are exposed as individual non-interleaved regions.

administrator without requiring a reboot [14].[8] Hence we believe that the hardware configuration should consist in one interleaved region per locality domain (CPU or SubNUMA Cluster, for good NUMA locality). The resource manager would then use namespaces for partitioning those static regions dynamically on job allocation. We observed a 1-gigabyte minimal granularity for this partitioning on our platform, and we believe this is sufficient for current HPC jobs on platforms with tens of hundreds of GB of memory.

Hence we propose to extend our strategy from Section 3.2:

1) NVDIMMs are configured 100% in App Direct and processors are in 1LM mode. **NVDIMM regions are interleaved at CPU level (or SubNUMA Cluster).**

2) Jobs request one or several region namespaces from the resource manager. Namespaces are configured as Device DAX by default in the Linux configuration (may be used for persistent storage as a single file).

3) Jobs specify how each namespace should be configured, as shown in Figure 8.

3a) An application that cannot work without multiple files may request the reconfiguration **of a namespace** as FSDAX.

3b) An application that needs lots of volatile memory may request the reconfiguration **of a namespace** as an additional NUMA node through the kmem driver.

If multiple namespaces from the same physical region are exposed as NUMA node, they are actually exposed as a single NUMA node[9] Fortunately, Linux Cgroups may be used to partition the memory of that shared NUMA node between jobs.

One may wonder whether using namespaces to partition a single region into multiple DAX incurs a performance penalty. Figure 9 shows that the overhead is negligible. Indeed processes only map DAX pages in their virtual address spaces and access them as regular memory. The actual overhead of using multiple namespaces is their creation during job prologue (a couple of minutes).

[8] Using the `ndctl` command-line tool again.

[9] Each persistent memory region corresponds to a unique NUMA node in ACPI tables.

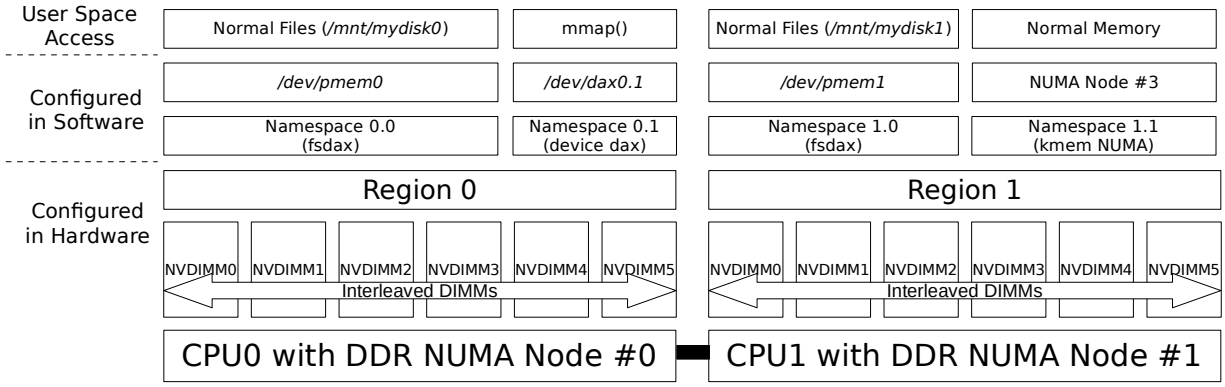

Figure 8: Using namespaces to partition regions between jobs requiring FSDAX, Device DAX or NUMA nodes. Each processor is configured with a single interleaved region. Software splits them between namespaces that may be configured according to jobs requirements.

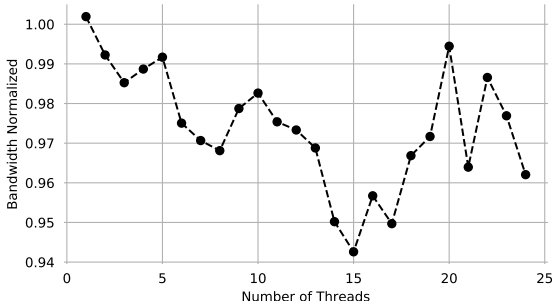

Figure 9: Performance of the STREAM Triad benchmark when using one Device DAX per thread (in a single region), normalized to using the same Device DAX for all threads.

4.3 DAX Locality

Finally, we look at how locality information is exposed in our proposed strategy. Indeed, even if the resource manager tries its best to allocate local namespaces to jobs, there is no guarantee that it will always be possible, and we explained in Section 2.5 that locality matters to performance of NVDIMMs. Hence, there is a need for the resource manager and the application to gather locality information about the different software handles that correspond to NVDIMMs.

When NVDIMMs are exposed as additional NUMA nodes, we implemented in hwloc a way to find out the corresponding local CPUs and DDR by looking at NUMA distances and memory target-initiator information in Linux.[10] Figure 10 depicts an example of such configuration.

For other cases (FSDAX, Device DAX and raw namespace), the information exposed by Linux is currently incomplete: only one local DDR node is reported even if there are multiple of them. For instance, in Figure 10, they would be reported as close to NUMA node #0 only (SubNUMA Cluster) instead of both #0 and #1 (entire Package). We are currently working with kernel devel-

[10]This code is currently in hwloc git master and will be published in the upcoming 2.1 release.

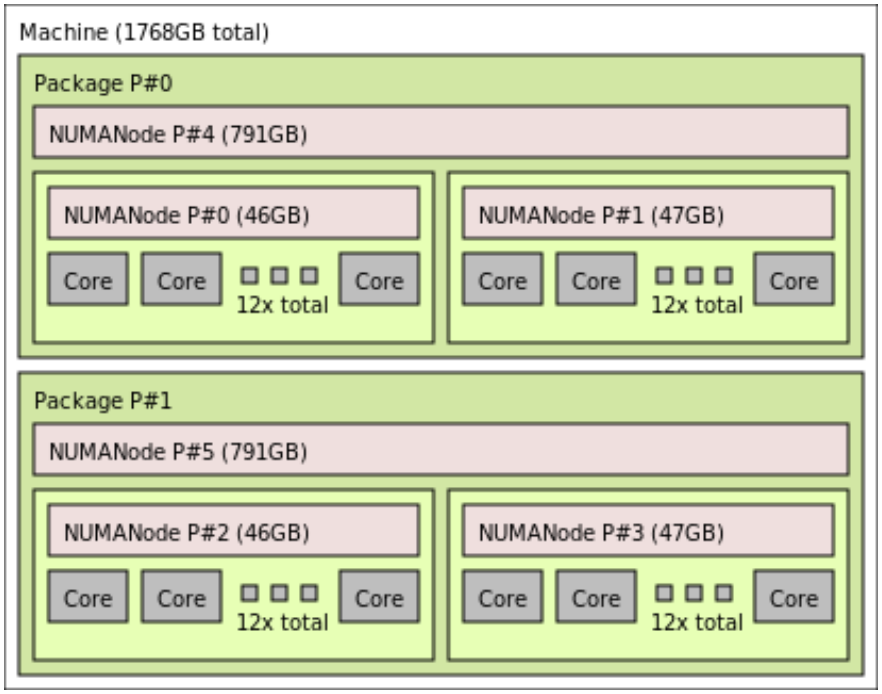

Figure 10: hwloc's lstopo representation of a platform with NVDIMMs exposed as additional NUMA nodes using the kmem DAX driver. Each processor has one local DDR NUMA node per SubNUMA Cluster (e.g. #0 and #1) and a single NVDIMM NUMA node (e.g. #4). Hence each core has two local memories.

opers to expose the correct information[11].

5 Related Works

HPC nodes are growing, causing co-scheduling to become necessary as soon as applications do not scale well to many cores. Indeed it is better to fill powered-on nodes rather than powering up yet another partially-used node. However previous work has shown than node sharing raises several performance issues, especially in the memory subsystem [13]. Many resources may be partitioned in software to avoid processes disturbing each others.

Resource managers such as SLURM are usually in charge of allocating cores and memory to jobs. They now use techniques such as Linux Cgroups for partitioning these resources between jobs [4] or containers [17]. Cache partitioning also appeared in recent processors as a way to also avoid co-existing cache pollution between applications [9]. However it is currently not supported by DDR caches in 2LM. Fortunately, we explained that we do not believe that 2LM is a sensible choice for HPC nodes.

When NVDIMMs are used as persistent storage, the resource manager is in charge of allocating this local storage to jobs. Like any local disk in computing nodes, these FSDAX may be provisioned by the manager, for instance as explicit or automatic burst buffers [7, 16]. This local

[11] `https://lists.01.org/pipermail/linux-nvdimm/2019-April/020822.html`

storage may also be used as a high-performance temporary storage between different jobs [7] for instance for in-situ analysis.

All these techniques are compatible with our proposal for partitioning non-volatile memory since we apply the partitioning when launching jobs (in the job prologue) and not in hardware.

6 Conclusion and Future Work

Non-volatile memory DIMMs are a promising technology that blurs the separation between volatile memory and persistent storage. We studied the different ways to use Intel Optane DCPMM and showed that supporting different use cases for different application needs requires careful hardware configuration. We explained why we think 2-Level-Memory is not a convenient solution for locality-aware partitioning of NVDIMMs between jobs. We showed why 1-Level-Memory looks like a better approach with more flexibility for memory allocation, easier configuration for the administrator and resource managers, and better locality.

Future work includes exposing better locality information from the Linux kernel to the resource managers and applications, as well as exposing in hwloc some information about the different kinds of NUMA nodes to ease application allocation policies. We are also looking at implementing our proposed ideas inside a resource manager such as SLURM.

References

[1] Intel 64 and IA-32 Architectures Optimization Reference Manual, Section 11.2 - Device Characteristics of Intel Optane DC Persistent Memory Module (Apr 2019)

[2] Arafa, M., Fahim, B., Kottapalli, S., Kumar, A., Looi, L.P., Mandava, S., Rudoff, A., Steiner, I.M., Valentine, B., Vedaraman, G., et al.: Cascade Lake: Next generation Intel Xeon scalable processor. IEEE Micro **39**(2), 29–36 (2019)

[3] Barnes, T., Cook, B., Deslippe, J., Doerfler, D., Friesen, B., He, Y., Kurth, T., Koskela, T., Lobet, M., Malas, T., Oliker, L., Ovsyannikov, A., Sarje, A., Vay, J., Vincenti, H., Williams, S., Carrier, P., Wichmann, N., Wagner, M., Kent, P., Kerr, C., Dennis, J.: Evaluating and Optimizing the NERSC Workload on Knights Landing. In: 2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). pp. 43–53 (Nov 2016), https://doi.org/10.1109/PMBS.2016.010

[4] Georgiou, Y., Hautreux, M.: SLURM Resources isolation through cgroups (2011), SLURM USER Group

[5] Götze, P., van Renen, A., Lersch, L., Leis, V., Oukid, I.: Data management on non-volatile memory: A perspective. Datenbank-Spektrum **18**(3), 171–182 (Nov 2018), https://doi.org/10.1007/s13222-018-0301-1

[6] Hansen, D.: Allow persistent memory to be used like normal RAM (Jan 2019), https://lwn.net/Articles/777212/, Linux Weekly News

[7] Henseler, D., Landsteiner, B., Petesch, D., Wright, C., Wright, N.J.: Architecture and Design of Cray DataWarp. In: Cray User Group Conference. London, UK (May 2016)

[8] Izraelevitz, J., Yang, J., Zhang, L., Kim, J., Liu, X., Memaripour, A., Soh, Y.J., Wang, Z., Xu, Y., Dulloor, S.R., Zhao, J., Swanson, S.: Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. CoRR (2019), http://arxiv.org/abs/1903.05714

[9] Lin, J., Lu, Q., Ding, X., Zhang, Z., Zhang, X., Sadayappan, P.: Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In: IEEE 14th International Symposium on High Performance Computer Architecture (HPCA). pp. 367–378. IEEE (2008)

[10] Liu, J., Chen, S.: Initial Experience with 3D XPoint Main Memory, Joint Workshop of HardBD and Active, held in Conjunction with ICDE. Macau, China, 2019

[11] Looi, L.: Intel Optane DC Persistent Memory Performance Overview, https://www.youtube.com/watch?v=UTVt_AZmWjM, Tech Field Day Exclusive At Intel Data-Centric Innovation Day. Apr. 2019

[12] NERSC: KNL Cache Mode Performance, https://www.nersc.gov/research-and-development/knl-cache-mode-performance-coe/

[13] Simakov, N.A., DeLeon, R.L., White, J.P., Furlani, T.R., Innus, M., Gallo, S.M., Jones, M.D., Patra, A., Plessinger, B.D., Sperhac, J., Yearke, T., Rathsam, R., Palmer, J.T.: A quantitative analysis of node sharing on hpc clusters using xdmod application kernels. In: Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale. pp. 32:1–32:8. ACM, New York, NY, USA (2016), https://doi.org/10.1145/2949550.2949553

[14] Upadhyayula, U., Scargall, S.: Introduction to Persistent Memory Configuration and Analysis Tools, Storage Developer Conference. Santa Clara, CA. Sep. 2018

[15] van Renen, A., Vogel, L., Leis, V., Neumann, T., Kemper, A.: Persistent Memory I/O Primitives. arXiv e-prints (Apr 2019), http://arxiv.org/abs/1904.01614

[16] Wang, T., Mohror, K., Moody, A., Sato, K., Yu, W.: An Ephemeral Burst-Buffer File System for Scientific Applications. In: SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 807–818 (Nov 2016), https://doi.org/10.1109/SC.2016.68

[17] Zounmevo, J.A., Perarnau, S., Iskra, K., Yoshii, K., Gioiosa, R., Essen, B.C.V., Gokhale, M.B., Leon, E.A.: A container-based approach to OS specialization for exascale computing. In: International Workshop on Container Technologies and Container Clouds (WoC) (2015)