M&MMs: Navigating Complex Memory Spaces with hwloc

Edgar A León, Brice Goglin, Andrès Rubio Proaño

M&MMs: Navigating Complex Memory Spaces with `hwloc`

Edgar A. León
Lawrence Livermore National Laboratory, California, USA
Email: leon@llnl.gov

Brice Goglin & Andres Rubio Proaño
Inria Bordeaux – Sud-Ouest, LaBRI, Univ. Bordeaux, France
Email: Brice.Goglin@inria.fr & Andres.Rubio@inria.fr

***Abstract*—The complexity of the memory system has increased dramatically in the last decade. As a result, high-performance computers include multi-level, heterogeneous, and non-uniform memories, each with significantly different properties. For example, a memory system nowadays may include three types of memory: low-latency memory (DDR), high-bandwidth memory (HBM), and high-capacity memory (NVM)–not to mention multiple NUMA domains. Because of their significantly different characteristics and number, scientific application developers face a tremendous challenge: Leverage the memory system effectively to improve performance and productivity.**

In this work, we present M&MMs, an interface to help manage the memory system complexity. It is comprised of a set of memory attributes and an API to express and manage the diverse memory characteristics using high-level metrics that are easy to understand. Our goal is to establish a building block to enable next-generation runtime systems, computing libraries, and scientific applications to leverage the best performance attributes of each memory, e.g., leverage the bandwidth of the fastest memory with the capacity of the largest memory. We believe M&MMs is a natural extension of `hwloc`–that focuses on the memory system–since `hwloc` exposes the locality of the hardware resources and it is the de facto standard for hardware topology discovery.

***Keywords*-Heterogeneous memory, multi-level memory, NVM, HBM, DDR, NVDIMM, NUMA, hwloc.**

I. Introduction

The complexity of computer architectures has grown significantly over the last decade to enable the expected increase in compute and memory capabilities of new systems. Vendors have resorted to a number of technologies to reduce power consumption per operation, keep capital costs reasonable, and improve system's capability. A few of these include multi-level memory, many-core, heterogeneous architectures (conventional processors coupled with accelerators), and heterogeneous memories.

Two types of architectures exemplify these technologies. First, the Intel Knights Landing (KNL) architecture [14] featuring a two-level memory system: A high-bandwith memory (HBM) and a high-capacity memory (DDR). Because of their significantly different capacity and bandwidth characteristics, the choice of memory can be key to the performance of applications. Furthermore, with as many as eight NUMA nodes or domains, this architecture represents a challenging programming environment for application developers. While there are a number of hardware (e.g., cache mode) and software methods to help ameliorate this complexity, extracting the performance of HBM with the capacity of DDR is still an art [17], [13], [5], [10], [9]. While the KNL architecture is not new, we expect multi-level memory to continue to play an important role in future systems.

Second, the CORAL [6] heterogeneous architecture, present in the top two supercomputers in the world according to the June 2019 Top500 list[1]. This architecture is comprised of central processing units (CPUs) coupled with graphics processing units (GPUs) and non-volatile memory. There are three types of memory: GPU-local high-bandwidth memory, CPU high-capacity memory, and, notably, non-volatile memory. For real scientific applications with large data sets, using the three-tier memory system is hard, particularly when the application's working set does not fit in GPU memory.

We expect upcoming platforms to continue this trend and include heterogeneous memories, each with different characteristics. Non-volatile dual in-line memory modules (NVDIMMs), for example, can be employed as fast local storage or large slow volatile memory. Compared to DDR, latency is expected to be higher, bandwidth lower, but capacity much higher. Thus, an arbitrary system may have high-bandwidth memory, high-capacity memory, non-volatile memory, and low-latency memory. All of these memories have different characteristics that, *in aggregate*, satisfy the demands of high-end applications.

Software environments that enable using the best properties of these memory technologies will be in high-demand and, at the same time, challenging to design and develop. To this end, we need a well-defined interface to query and manage the performance characteristics of the memory diversity of a machine.

In this position paper, we propose an interface to help manage the complexity of memories on emerging systems. We believe that having a consistent set of attributes and a consistent representation of these different types of memories is timely and can be useful for application, system, and library developers. We call this interface *Mix and Match Memories*

[1] https://www.top500.org/.

(M&MMs)–reflecting the growing memory diversity–and propose it as an extension to `hwloc`, the de facto standard for exposing the locality of hardware resources on a given system.

II. HWLOC MEMORY LIMITATIONS

`hwloc` is the de facto standard interface for exposing the locality of hardware resources in HPC platforms [4]. It builds a hierarchy of objects based on inclusion and physical location on a server. For instance, all cores of a processor package are exposed as children of that package. Since `hwloc` 2.0, memory objects are attached to the CPU hierarchy to show which cores are local to a given NUMA node [8]. At the same time, this model explicitly shows which memory nodes are local to each core.

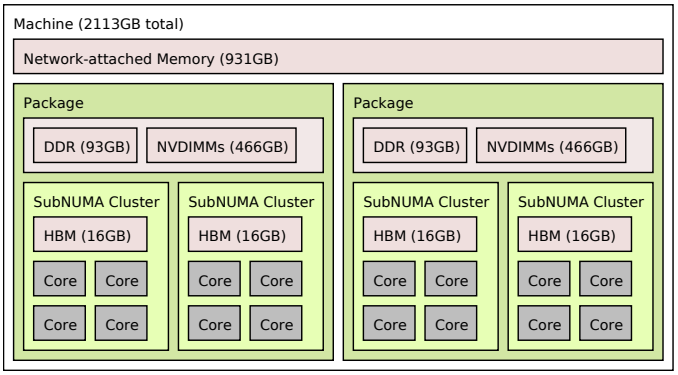

Fig. 1. Example of hwloc's output with multiple kinds of memory. Each CPU package has local DDR and NVDIMM NUMA nodes, while HBM is also attached to each SubNUMA Cluster. In addition, network-attached memory is exposed as local to the entire machine.

On Figure 1, four memories are available for allocation from each core, hence the need to provide a way to choose between them. At the package level, `hwloc` exposes DDR memory before NVDIMMs because DDR is usually the default allocation node. Unfortunately, this choice does not match the needs of all applications. HBM is attached at a smaller level of the hierarchy on this server and one may expect it to provide *higher performance*. However, there is currently no obvious way to expose such performance information and for what dimension, e.g., latency vs. bandwidth. `hwloc` could list HBM before DDR to guide applications to use HBM by default, but it is not possible on this platform because HBM and DDR are not attached to the same level of the CPU hierarchy.

III. M&MMS MEMORY ATTRIBUTES

The M&MMs framework assigns attributes to each memory device on a machine. Given the heterogeneity of memories in emerging systems, these attributes help classify them in terms of intuitive metrics to help select the *right* device for a given use case. For a given attribute, we associate a non-negative value to each memory. Positive values indicate the memory's rank, while a zero value indicates the absence of such attribute.

We use these attributes to create *orderings of memories*. The best, second best, ..., and worst device can then be listed and if the best memory is *available*, users may choose it to satisfy their memory needs. For example, for a latency-sensitive application, a user may request the orderings based on the latency attribute and allocate data on the top ranked device resulting from such ordering.

We describe below the proposed memory attributes.

A. Bandwidth

Computer architectures such as KNL include two types of memory: high-capacity and high-bandwidth. There is a large bandwidth differential between these two (four to five times), which makes bandwidth-intensive applications extremely sensitive to the choice of memory and hardware mode (e.g., cache and flat) [14]. For these applications, allocating data in the right memory is key to performance.

We expect HBM to play a key role in future systems. Currently, it is a key component of leadership systems such as Summit and Sierra from the U.S. Department of Energy [16] and Fugaku (formerly Post-K) from RIKEN in Japan [7].

On a system with HBM, DDR, and non-volatile memory (NVM), for example, the resulting ordering based on the bandwidth attribute may return the following ordering:

$$[HBM, DDR, NVM]$$
$$where \quad [HBM] = 1 \ [DDR] = 2 \ [NVM] = 3$$

Instead of using absolute memory bandwidth values, a non-negative key is used to rank the memories. In this case, HBM has the highest bandwidth because it is ranked one (lowest rank is the best–top of the list).

Note there may be more than one ordering for attributes such as bandwidth and latency. These two can have significantly different values for read access and write access. If the read ordering is different than the write ordering for the memories in the system, we may have one for read access and one for write access.

B. Latency

Although seemingly related, bandwidth and latency may not be correlated in practice. On platforms with *fast* and *slow* memory, one may expect the fast memory to always provide better latency *and* bandwidth than the slow memory. In practice, this may not be the case. For instance, HBM latency on KNL is higher than DDR's when the platform is loaded. Moreover, application requirements may be dependent specifically on bandwidth (streaming kernels) or latency (pointer chasing-like applications). Hence, providing explicit independent attributes for bandwidth and latency allow us to better describe their needs.

Following the example above for a system with HBM, DDR, and NVM, a latency ordering may be as follows:

$$[HBM, DDR, NVM]$$
$$where \quad [HBM] = 1 \ [DDR] = 1 \ [NVM] = 2$$

Note that the HBM and DDR ranks are the same as a result of comparable access latencies of these two memories. Memory access latency, however, depends on a number of

factors, including load, that may make the differences between DDR and HBM more significant. In future work, we may consider taking these factors into consideration when ranking memories.

C. Capacity

The ordering of memories is from largest to smallest, i.e., the largest memory is ranked one while the smallest memory is ranked N, for N memory devices. For example,

$$[NVM, DDR, HBM]$$
$$where \quad [NVM] = 1 \ [DDR] = 2 \ [HBM] = 3$$

Similarly to latency, memories with similar capacity would be ranked the same. In general, we can associate the same rank to two memories if the associated absolute value (bytes) is within a certain threshold. For example, if two multi-gigabyte memories differ by a few megabytes, they may be ranked the same.

In addition to hardware capacity, we may consider *available capacity* that would take into consideration dynamic behavior. The resulting ordering would be dependent on when the attribute is queried.

D. Persistance

The persistance (non-volatility) attribute can be either zero or one. A zero value corresponds to volatile memories (e.g., DDR). Non-volatile memories with load/store interfaces have a non-volatile attribute of one.

One type of persistent memory is NVDIMM. When NVDIMMs are used for persistent storage, they are not exposed by the OS as a NUMA node. On Linux, they are used through specific *DAX* devices. In this case, this memory is only listed as another storage tier.

When NVDIMM memory is configured as *normal memory*, it is exposed as an additional NUMA node (Linux 5.1) and the persistance attribute applies.

E. Locality

We also envision a locality attribute that would describe whether a memory device is attached specifically to a subset of cores or shared by many of them. On Figure 1, HBM has strong locality to four cores because it is only attached to a SubNUMA Cluster. DDR and NVDIMM are local to twice as many cores (entire CPU package), and the network-attached memory is shared by the entire machine.

This locality attribute may be useful when dealing with data sharing between cores: Two tasks sharing data might perform better if data is stored in a memory device local to both of them and if the memory interconnect is under contention. Hence, this could be a filter to avoid non-local memory devices for the involved cores.

We are still developing the specifics of this attribute, but present a use case to help clarify our goal. An MPI application running on a machine with a topology similar to Figure 1 can create MPI sub-communicators based on the locality attribute. For $distance = 1$[2], MPI tasks running on the same SubNUMA Cluster are assigned to the same sub-communicator and to the same HBM local memory. For $distance = 2$, MPI tasks running on the same package are assigned to the same sub-communicator and to the same DDR memory. Note that similar groupings can be created today using `hwloc`'s compute devices (e.g., Package). In contrast, our locality attribute targets the memory devices in the machine. This difference may be significant for two reasons: (1) Architectures may not have a 1:1 correspondance between compute *packages* and memory, and (2) our approach would provide a memory handle to the local memory based on the given distance–without having to specify what NUMA domain that may be, if any.

IV. M&MMs API

We propose the M&MMs application programming interface as an extension to `hwloc`'s interface. It includes functions to generate orderings of memories according to a given attribute, a set of memory attributes, and a set of memory devices as shown in Table I and Figure 2.

TABLE I
M&MMs MEMORY ATTRIBUTES AND AN EXAMPLE OF MEMORY DEVICES.

Attributes	Devices
latency	hbm
bandwidth	ddr
capacity	nvm
persistance	
locality	

An ordering of memories can be constructed by ranking each memory device according to a given attribute or metric. For example,

$$mmm\_get\_rank(bandwidth, hbm)$$

ranks HBM according to the bandwidth metric relative to other memory devices. In this example, $rank = 1$ since HBM provides the highest bandwidth. For attributes such as persistance, only binary values are available, e.g., `mmm_get_rank(persistance,ddr)` returns zero (absence of persistance). In addition, the ordering of memories is obtained using `mmm_get_ordering`. For example,

$$mmm\_get\_ordering(bandwidth)$$

would result in the ordered list $[hbm, ddr, nvm]$. Note that memory orderings include handles or pointers to each of the respective memories.

When multiple devices *perform* similarly on a given metric, their rank may be the same. In this case, one can employ an optional secondary attribute to disambiguate the ordering. In the case we discussed where DDR and HBM have similar latencies, one can use the bandwidth parameter to get a full ordering, e.g.,

[2]Using the term distance instead of M&MMs's rank to distinguish from MPI's *ranks*.

`mmm_get_ordering(latency, bandwidth)` would result in

$$rank(hbm) = 1,\ rank(ddr) = 2,\ rank(nvm) = 3$$

Note the difference with `mmm_get_ordering(latency)` where

$$rank(hbm) = 1,\ rank(ddr) = 1,\ rank(nvm) = 2$$

We also have a function to determine the memory device with the best attribute for a given metric, e.g.,

$$mmm\_get\_device(capacity)$$

This function would return a handle to the memory device with the highest capacity that can be used for subsequent memory allocations.

In addition to the value or rank given for a particular attribute relative to the memory devices in the system, it is often useful to know the actual (absolute) values. There is a function for this purpose. It provides both a quantity and a unit, e.g.,

$$mmm\_get\_value(bandwidth, hbm)$$

may provide, say, 500 and GB/s.

An important consideration arises when a compute node includes more than one device of the same type. In this case, the M&MMs API uses *local memory* (relative to the location where the caller executes). For a given core, local memory is the list of memory nodes attached to that core, or the containing CPU package, or the containing chiplet/SubNUMA cluster (SNC), or the entire machine. This is the definition of local memory in `hwloc` since version 2.0.

We conclude this section with a summary of important functions comprising the M&MMs API (Figure 2) and the rank matrix of a hypothetical system with three types of memory (Table II).

TABLE II
RANKS ASSIGNED TO MEMORY DEVICES ON AN EXAMPLE SYSTEM WITH THREE TYPES OF MEMORY. NOTE THAT NVM FEATURES A LOAD/STORE INTERFACE.

	hbm	ddr	nvm
latency	1	1	2
bandwidth	1	2	3
capacity	3	2	1
persistance	0	0	1

V. ASSIGNING VALUES TO MEMORY ATTRIBUTES

In the previous section, we discussed ranking memories based on a given attribute and obtaining actual performance values. In this section, we describe how we obtain these values that enable our ranking system. There are two main sources: hardware information and benchmarking, both of which we discuss below.

Provide the device's rank according to attribute.
`mmm_get_rank(attribute, device)`

Provide a memory ordering according to attribute.
`mmm_get_ordering(attribute, [attribute])`

Provide the top device for a given attribute.
`mmm_get_device(attribute)`

Provide the value of an attribute for a given device and the unit of such value.
`mmm_get_value(attribute, device)`

Provide the number of devices in the memory ordering for a given attribute.
`mmm_get_num_devices(attribute)`

Fig. 2. Important functions of the M&MMs API.

A. Hardware information

We use the *Heterogeneous Memory Attributes Table* (HMAT), which is expected to be available in upcoming platforms for better describing complex memory hierarchies. This table was introduced in revision 6.2 of the ACPI specification.[3] It describes multiple memory devices that are local to the same cores as depicted in Figure 1. In addition, the HMAT table describes *Memory-side Caches*. For example, Intel *Cascade Lake* processors configured as 2-level-memory use DDR as a cache for NVDIMMs [2] (see also Section VII-C).

The table may expose the theoretical latency and bandwidth between all memory initiators (sets of cores) and all memory targets (NUMA nodes). For instance, on a platform with both HBM and DDR, cores access their local HBM at 500 GB/s with a 100 ns latency, the same cores access their local DDR at 100 GB/s and 110 ns, while the remote CPU accesses HBM at 300 GB/s and 150 ns. Latencies and bandwidths may optionally be specified independently for read and write accesses.

Linux exposes those attributes in the `sysfs` virtual filesystem since kernel version 5.2. However, performance attributes are only exposed between cores and their *local* memory targets (no attributes for the remote HBM in the above example). Fortunately, for our proposal, it provides enough information to sort local memory targets based on latency, bandwidth, and capacity.

ACPI tables are already used by x86 and ARM platforms. We expect similar performance information to be available on other platforms in the future, for instance in the device-tree of IBM POWER machines.

Unlike ACPI tables and other architecture-specific information, the goal of the M&MMs interface is to expose the

[3]https://uefi.org/specifications.

memory attributes and functions at the `hwloc` level for portability across vendors and computer architectures.

B. Benchmarking

Even though we expect most vendors to provide an ACPI HMAT or similar table with performance attributes in their upcoming platforms, we may need to work with hardware without such tables or with buggy attributes. A fallback would be use the legacy SLIT table (*System Locality Information Table*) that exposes theoretical latencies between NUMA nodes. Unfortunately, it rarely exposes values that are precise-enough to sort local memory targets.

We believe that benchmarking is a good way to work around such incomplete or buggy firmware. For instance, DDR performance on Intel *Cascade Lake* was measured empirically to about 80 GB/s and 285 ns latency while NVDIMM performance was 10 GB/s and 860 ns latency [15]. Although these numbers depend on the number of threads accessing memory, the access pattern, etc., they are sufficient for the purposes of our memory orderings.

The values calculated through benchmarking would be calculated once and stored in an XML file within `hwloc`.

VI. Use cases

The M&MMs API can be used by system and utility libraries such as smart memory allocators and smart placement of compute workers, e.g., processes and threads. In addition, scientific applications may use these abstractions to manage data movement within the memory system.

A. Smart memory allocators

Memory allocators such as memkind [5] may leverage M&MMs by extending their APIs to include an optional parameter with high-level memory attributes such as bandwidth and latency:

$$mem\_alloc(\ \ldots,\ [bandwidth \mid latency \mid persistance]\ )$$

This function would allocate memory in the appropriate memory space according to the given attribute.

If there is not enough space available on the highest ranked memory, the allocator may use the memory orderings provided by M&MMs to determine which memory to use next. As mentioned before, the orderings are relative to where the process or thread executes, e.g., on hardware thread X of CPU package Y.

`hwloc` memory allocation and binding primitives such as `hwloc_set_area_membind()` and `hwloc_alloc_membind_policy()` may also be extended to support these new attributes as additional memory binding flags.

B. Hint-based placement of compute workers

Next-generation resource managers may implement more powerful and high-level affinity policies to map an application's processes, threads, GPU kernels, etc. to the underlying machine. Current policies are compute-driven allowing a user to map processes and threads to compute abstractions such as hardware threads, cores, and packages. The mapping can be done, for example, by spreading tasks over the resources or keeping them close together.

Emerging affinity policies may use higher-level attributes or *hints* to bind a process to the hardware. For example, a user may specify that a particular thread or task is memory-bandwidth bound and, as such, it should placed to enable maximum bandwidth within the executing scope of such process. The M&MMs API can be employed to enable such placement optimizations.

VII. M&MMs limitations and other important considerations

A. Non-local memory

The M&MMs API focuses on *local* memory only relative to the executing thread. This raises the question of whether a failure to allocate in a (full) local HBM should lead to an allocation on the local DDR or a remote HBM near another CPU. We believe that local allocations are better in the majority of cases because they avoid congestion on the memory interconnect. However, imbalances in memory needs within a parallel job can cause some memory devices to be full while others are not. CPUs local to full memories should have the opportunity to allocate remotely in such imbalanced cases. This raises several concerns:

- Although exposing performance attributes of remote memories is possible in the M&MMs API, the impact of congestion on the actual performance makes these numbers much harder to specify. A remote HBM might have higher bandwidth than a local DDR only when there is no congestion.
- The number of possible memory targets could grow significantly as node complexity increases. Upcoming Intel and AMD processors will have up to 4 proximity domains, hence up to 24 possible targets with 2 processors equipped with HBM, DDR, and NVDIMMs. It is not clear to us whether ordering that many devices is convenient and useful to applications in common cases. Exposing local devices by default, and offering remote fallbacks if local devices are full might be a better solution.
- As explained in Section V-A, Linux kernel developers chose to only expose local memory attributes in Linux 5.2 because the entire matrices of attributes between all pairs of proximity domains could waste too much kernel memory. Hence, the M&MMs API is mapped onto these currently exposed attributes.

B. Other kinds of memory

The M&MMs API is designed to support all kinds of memory devices. However, we mostly talked about HBM, DDR, and NVDIMMs in this paper. More kinds of devices are already available. For instance, NVIDIA's V100 GPUs expose their internal HBM as additional NUMA nodes on POWER9 processors (see Figure 3), allowing applications to

bind and allocate memory on the GPU directly from the host. `hwloc` currently hides these NUMA nodes by default because NVIDIA recommends only allocating memory on the GPUs using the CUDA API.[4] The M&MMs API could be extended to consider these memory targets if a specific flag is given.

Peripherals exposing memory is a more general problem. Upcoming NVMe (non-volatile memory express) drives will be able to expose *Persistent Memory Regions* that will behave like NVDIMMs while being stored on disk (see Figure 3). Both NVMe and GPU memory could be exposed in the M&MMs API, but it is not clear whether the hardware will be able to expose performance attributes since the ACPI tables do not cover such optional PCI devices. Benchmarks, however, may still be used to retrieve performance numbers as explained in Section V-B.

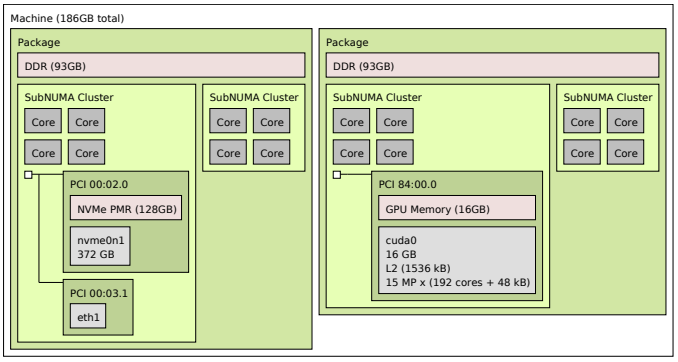

Fig. 3. Example of hwloc's output with NUMA nodes (pink-colored) inside PCI devices. The NVMe disk exposes a Persitent Memory Region whose locality is the first SubNUMA Cluster of the first CPU package. The Memory of an NVIDIA GPU is also exposed as a NUMA node whose locality is the first cluster of the second package. If a network-attached memory was available, it would be attached through the `eth1` network interface and may inherit its locality.

Finally, network-attached memory [1] could be exposed in the M&MMs API. However, these devices cannot be managed with standard NUMA APIs yet. And their locality is hard to define: If the network-attached memory is connected to a specific network interface, should that memory be considered local to the CPUs near that NIC only, as envisioned in Figure 3? Or should it be considered as far from all CPUs (as in Figure 1) because network performance is much lower than the intranode memory interconnect? This is a question we plan to address when those devices become more widely available.

C. Memory-side caches

Finally, the M&MMs API may be limited in describing memory targets that are hidden behind a cache. *Memory-side Caches* have been standardized in the ACPI specifications (see Section V-A) as caches that handle all requests from all cores to a specific range of memory.[5] Those caches are now becoming widespread thanks to the 2-Level-Memory mode on the latest Intel Xeon processors where the DDR can be configured as a Memory-side Cache in front of NVDIMMs. The KNL *Cache* mode, actually, also made MCDRAM a *Memory-side Cache* in front of DDR.

Those caches are direct-mapped and their performance depends significantly on the access pattern of the application [12]. Hence, their impact on the performance of some memory targets may be difficult to describe as basic performance attributes.

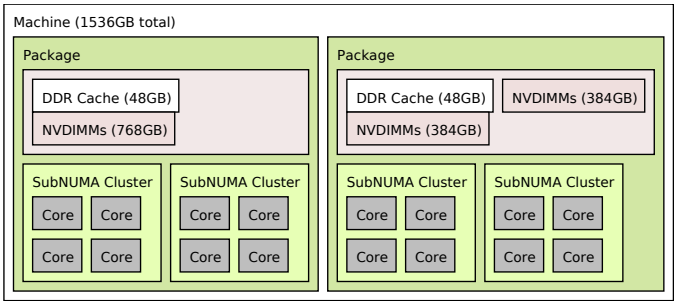

Fig. 4. Example of hwloc's output on an Intel Cascade Lake Xeon platform configured as 2-Level-Memory. The first CPU NVDIMMs are all used as normal memory (*Memory Mode*) behind a DDR cache. The second CPU has only half of its NVDIMMs configured that way. The other half is configured as *App Direct* (usually for storage), but actually exposed as a separate NUMA node (using the Linux DAX *kmem* driver). The DDR Cache does not apply to that node.

Figure 4 shows a corner case where part of a memory device is cached while the other is not. Unfortunately, the ACPI HMAT table is expected to report *uncached* performance attributes for both parts. Hence, we may need to add specific information about those caches to improve the M&MMs API.

D. Dynamic attributes

The majority of the attributes considered here are calculated once: their values do not change with system utilization. A useful extension would consider these attributes as dynamic entities. This is the case of available capacity and, potentially, latency and bandwidth. There are two avenues we could consider to retrieve attributes dynamically: periodic retrieval and instant retrieval. With the former, an attribute is refreshed at periodic intervals driven by an input period, while the latter queries attributes instantly.

There are benefits and drawbacks to both approaches. The overhead of instant retrieval can be significant and impact application performance. Furthermore, a short-duration transient event can have a significant effect on an attribute's value rendering it not representative of the next time interval. Similarly, the challenge with periodic retrieval relies on determining the *right period*, which is likely to be application dependent.

We will consider dynamic attributes in future work. Currently, the easiest attribute to consider is available capacity.

VIII. Related work

The need to manage heterogeneous memory in HPC was emphasized with the Intel KNL Xeon Phi (*flat mode*) where

[4]NUMA-interleaved allocations on such machines span the buffers on both CPU and GPU memory, causing unexpected performance issues.

[5]Contrary to traditional CPU caches that handle requests only from some cores but to all memory targets.

both DDR and MCDRAM can be used to allocate memory. Several APIs to explicitly allocate in one kind of memory or another have been proposed, including the memkind library [5]. The easy-to-use `autohbw` wrapper can use the memory allocation size as a threshold for deciding where to allocate data. Unfortunately, this API is specific to a memory system with MCDRAM and DDR. M&MMs encompasses heterogeneous memories more generally and provides high-level attributes to manage these devices.

A few *automatic* strategies have been proposed to decide where to allocate data based on application patterns. Servat et al. [13] and Narayan et al. [11] (MOCA) use a post-mortem analysis of memory allocations or access patterns. Information such as hardware counters helps determine which memory technology better suits each application dataset. The M&MMs API can be used as a way to generalize these approaches to any combination of memory technologies. It also brings a portable way to retrieve quantitative information about devices, such as bandwidths and latencies, instead of hardwiring them for a specific platform.

SICM [18] proposed to extend the Linux kernel memory allocation policies by making the preferred allocation order configurable. This approach confirms that ordering between memory devices is important, but the implementation is very different from M&MMs. First, SICM requires Linux kernel changes that do not follow the current trend of Linux kernel developers, which is to expose performance attributes to applications that help drive allocation behavior. Second, the ordering is configured machine-wide in the kernel. All allocations of all jobs running simultaneously on a node will use the same ordering. In our approach, we expose attributes to applications and libraries and let them choose the relevant ordering for each use case.

Umpire [3] is a resource management library that allows the discovery and management of different memories on a system. It is intended for high-performance applications and provides a number of memory operations, dynamic memory pools, and introspection capabilities. We envision M&MMs to enable libraries like Umpire to use memory attributes and memory orderings in their APIs.

Finally, we emphasize the abstractions presented in this paper can be leveraged by higher-level systems and libraries providing automatic frameworks that enable transparent and efficient memory allocations.

IX. Summary

In this work, we present *Mix and Match Memories*, an interface to help manage the complexity of the memory system of emerging and future architectures. Because of the different types of memories and their different characteristics, it is a great challenge to use them effectively and efficiently. Ideally, one would like to leverage the low-latency of DDR, the high-capacity of NVM, and the high-bandwidth of HBM as a single memory system that applications can utilize. We are taking a step in that direction by providing building blocks to characterize the heterogeneous memories present on a single system.

Our approach focuses on specifying a number of memory attributes and an API to query and classify the memories devices. These attributes represent high-level characteristics that are relatively easy to reason about such as bandwidth, locality, and persistance. We expect system and utility libraries to leverage these attributes and build higher level abstractions to enable programmer productivity and performance.

Acknowlegments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-786883.

References

[1] W. Allcock, B. Bernardoni, C. Bertoni, N. Getty, J. Insley, M. E. Papka, S. Rizzi, and B. Toonen. Ram as a network managed resource. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 99–106, May 2018.

[2] Mohamed Arafa, Bahaa Fahim, Sailesh Kottapalli, Akhilesh Kumar, Lily P Looi, Sreenivas Mandava, Andy Rudoff, Ian M Steiner, Bob Valentine, Geetha Vedaraman, et al. Cascade Lake: Next generation Intel Xeon scalable processor. *IEEE Micro*, 39(2):29–36, 2019.

[3] David A. Beckingsale and Richard D. Hornung. Umpire: Status Report and Future Development Plan. Technical Report LLNL-TR-754118, Office of Scientific and Technical Information, 6 2018.

[4] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*, pages 180–186, Pisa, Italia, February 2010. IEEE Computer Society Press.

[5] Christopher Cantalupo, Vishwanath Venkatesan, Jeff R. Hammond, and Simon Hammond. User Extensible Heap Manager for Heterogeneous Memory Platforms and Mixed Memory Policies, 2015. http://memkind.github.io/memkind/memkind_arch_20150318.pdf.

[6] CORAL: Collaboration of Oak Ridge, Argonne and Livermore National Laboratories. Draft CORAL build statement of work. RFP No. B604142, LLNL-PROP-636244, Office of Science and the National Nuclear Security Administration's Advanced Simulation and Computing (ASC) Program, U.S. Department of Energy, December 2013.

[7] Fujitsu. Supercomputer Fugaku. ISC High Performance, Booth Presentation, June 2019.

[8] Brice Goglin. Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications. In *International Symposium on Memory Systems*, MEMSYS'16, Washington, DC, 2016. ACM.

[9] Edgar A. León and Matthieu Hautreux. Achieving transparency mapping parallel applications: A memory hierarchy affair. In *International Symposium on Memory Systems*, MEMSYS'18, Washington, DC, October 2018. ACM.

[10] Ang Li, Weifeng Liu, Mads R. B. Kristensen, Brian Vinter, Hao Wang, Kaixi Hou, Andres Marquez, and Shuaiwen Leon Song. Exploring and analyzing the real impact of modern on-package memory on HPC scientific kernels. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'17, Denver, CO, 2017. ACM.

[11] Aditya Narayan, Tiansheng Zhang, Shaizeen Aga, Satish Narayanasamy, and Ayse K. Coskun. MOCA: Memory Object Classification and Allocation in Heterogeneous Memory Systems. In *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium*, Vancouver, BC, Canada, May 2018. IEEE.

[12] NERSC. KNL Cache Mode Performance, 2017.

[13] Harald Servat, Antonio Pena, German Llort, Estanislao Mercadal, Hans-Christian Hoppe, and Jesus Labarta. Automating the Application Data Placement in Hybrid Memory Systems. In *Proceedings of the IEEE International Conference on Cluster Computing*, Hawaii, USA, September 2017.

[14] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights landing: Second-generation intel xeon phi product. *IEEE Micro*, 36(2):34–46, March 2016.

[15] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent Memory I/O Primitives. *arXiv e-prints*, abs/1904.01614, April 2019.

[16] Sudharshan S. Vazhkudai, Bronis R. de Supinski, Arthur S. Bland, Al Geist, James Sexton, Jim Kahle, Christopher J. Zimmer, Scott Atchley, Sarp Oral, Don E. Maxwell, Veronica G. Vergara Larrea, Adam Bertsch, Robin Goldstone, Wayne Joubert, Chris Chambreau, David Appelhans, Robert Blackmore, Ben Casses, George Chochia, Gene Davison, Matthew A. Ezell, Tom Gooding, Elsa Gonsiorowski, Leopold Grinberg, Bill Hanson, Bill Hartner, Ian Karlin, Matthew L. Leininger, Dustin Leverman, Chris Marroquin, Adam Moody, Martin Ohmacht, Ramesh Pankajakshan, Fernando Pizzano, James H. Rogers, Bryan Rosenburg, Drew Schmidt, Mallikarjun Shankar, Feiyi Wang, Py Watson, Bob Walkup, Lance D. Weems, and Junqi Yin. The design, deployment, and evaluation of the CORAL pre-exascale systems. In *International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC'18, pages 52:1–52:12, Dallas, Texas, 2018. IEEE.

[17] Gwendolyn Voskuilen, Arun F. Rodrigues, and Simon D. Hammond. Analyzing Allocation Behavior for Multi-level Memory. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 204–207, New York, NY, USA, 2016. ACM.

[18] Sean Williams, Latchesar Ionkov, and Michael Lang. NUMA Distance for Heterogeneous Memory. In *Proceedings of the Workshop on Memory Centric Programming for HPC*, MCHPC'17, pages 30–34, New York, NY, USA, 2017. ACM.