

Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications

Brice Goglin

▶ **To cite this version:**

Brice Goglin. Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications. 1st ACM International Symposium on Memory Systems (MEMSYS16), Oct 2016, Washington, DC, United States. 10.1145/2989081.2989115 . hal-01330194v2

HAL Id: hal-01330194

https://inria.hal.science/hal-01330194v2

Submitted on 17 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications

Brice Goglin
Inria Bordeaux – Sud-Ouest
LaBRI – University of Bordeaux, Talence, France
Brice.Goglin@inria.fr

***Abstract*—High-performance computing requires a deep knowledge of the hardware platform to fully exploit its computing power. The performance of data transfer between cores and memory is becoming critical. Therefore locality is a major area of optimization on the road to exascale. Indeed, tasks and data have to be carefully distributed on the computing and memory resources.**

We discuss the current way to expose processor and memory locality information in the Linux kernel and in user-space libraries such as the hwloc software project. The current *de facto* standard structural modeling of the platform as the tree is not perfect, but it offers a good compromise between precision and convenience for HPC runtimes.

We present an in-depth study of the software view of the upcoming Intel Knights Landing processor. Its memory locality cannot be properly exposed to user-space applications without a significant rework of the current software stack. We propose an extension of the current hierarchical platform model in hwloc. It correctly exposes new heterogeneous architectures with high-bandwidth or non-volatile memories to applications, while still being convenient for affinity-aware HPC runtimes.

***Keywords*-Heterogeneous memory; locality; affinity; structural modeling; user-space runtimes; high-performance computing; Linux.**

I. Introduction

Parallel platforms are increasingly complex. Processors now have many cores, multiple levels of caches as well as a NUMA interconnect. Exploiting the computing power of these machines requires deep knowledge of the actual organization of the hardware resources. Indeed tasks and data buffers have to be carefully distributed on computing and memory resources so as to avoid contention, remote NUMA access, etc. Making the most of the platform requires high performance computing runtimes to match the application requirements with the hardware topology. Memory hierarchy is a key component in this topology awareness. Precise knowledge of the locality of NUMA nodes and caches with respect to CPU cores is required for proper placement of task and data buffers.

Operating systems such as Linux already expose some locality information to user-space applications [1], and HPC runtimes in particular. Modeling the platform as a hierarchical tree is a convenient way to implement locality-aware task and data placement. Even if a tree does not perfectly match the actual hardware topology, it is a good compromise between precision and performance; algorithms such as recursive top-down partitioning can be easily implemented for distributing tasks on the platform according to the hierarchical model of the hardware. Basic queries such as finding which cores and NUMA nodes are close to each other are also straightforward in such a structural model. This is as easy as looking up specific resources in parent or children nodes in the tree.

However, new memory architecture trends are going to deeply modify the actual organization of the memory hierarchy. Indeed, high bandwidth and/or non-volatile memories as well as memory-side caches are expected to significantly change the traditional platform model. As a case study, we explain how the Linux kernel and user-space libraries expose the topology of the upcoming *Intel Knights Landing* architecture. We reveal several flaws in the current HPC software stack, which lacks expressiveness and a generic memory model. We then propose a new structural model of the parallel computing platforms. This model comes as an extension to the hwloc library, the *de facto* standard tool for exposing hardware topology to HPC applications. Our model still satisfies the needs of many existing locality-aware HPC runtimes while supporting new heterogeneous memory hierarchies.

II. Exposing Memory Locality as a Hierarchical Tree of Resources

We explain in this section why memory locality is critical to high-performance computing and we discuss performance and structural platform modeling. We then explain why modeling as a hierarchical tree of hardware resources is a good trade-off between precision and convenience.

A. Memory Locality matters to HPC Applications

The importance of NUMA awareness has been known in high-performance computing for decades [2]. Threads should run as close as possible to the data they use, and data should be allocated on NUMA nodes close to these threads. Many research works focused on improving application performance by placing threads and data using information about the application behavior and about the architecture [3], [4]. Such locality issues may be dealt with by looking at hardware performance counters to auto-detect non-local NUMA memory accesses [5], or by having the application provide the runtime with hints about its usage of memory buffers [6].

The democratization of multicore processes in the last twelve years added cache sharing to the reasons why locality matters to performance. Indeed modern processors contain multiple levels of caches, some being private to a single core, others being shared by all or some cores, as depicted on Figure 1. This causes synchronization and data sharing performance to vary with task placement even more since data may or may not be available in a local cache thanks to another core using it. Information about the affinities between threads and data buffers had therefore been used for better placement based on the impact of caches on performance [7].

There is actually a need to find a trade-off between cache and NUMA affinities [8]. Indeed keeping multiple tasks below a shared cache favors communication between them, as long as the exchange data sets fits in the cache. However, the opposite placement – spreading tasks on cores of different NUMA nodes – also has the advantage of increasing the total available memory bandwidth by using more NUMA nodes. There is a generic problem of distributing the workload across machines while addressing affinity constraints. Memory-intensive applications may prefer spreading, while communication- or synchronization- intensive applications may prefer compact placement below a shared cache.

These ideas apply to shared-memory programming because of shared buffers between threads, but also

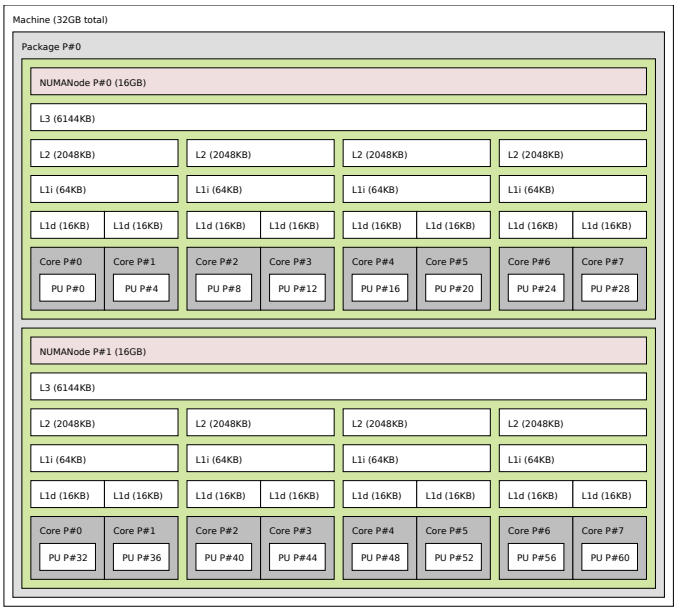


Fig. 1. AMD platform containing Opteron 6272 processors, simplified to a single processor, and reported by hwloc's `lstopo` tool. This processor package is made of two parts containing one NUMA node and one L3 cache each. L2 and L1i caches are then shared by *Compute Units* pairs of cores, while the L1d is private to each single-thread core.

to processes as soon as they communicate with each other within the machine. Indeed MPI communication performance varies with the physical distance. Intra-socket communication is usually faster than inter-socket causing performance to increase when processes are placed close to their favorite peers [9].

Placing tasks and data according to the application affinities may be performed dynamically by monitoring performance counters and detecting cache contention or remote NUMA accesses [10]. Another approach consists in having the application help the runtime system by providing hints about its affinities [6]. MPI process launchers may also use the application communication pattern to identify which processes to co-locate [11]. These ideas rely on information about the application software. They also requires deep knowledge about the hardware so that the application needs may be mapped on to the machine resources.

B. Performance or Structural Modeling

The complexity of modern computing platforms makes them increasingly harder to use, causing the gap between peak performance and application performance to widen. Understanding the platform behavior under different kinds of load is critical to performance optimization. Performance counters are a convenient way

to retrieve information about bottlenecks for instance in the memory hierarchy [12] and apply feedback to better schedule the next runs [7]. However these strategies remain difficult given the number of parameters that are involved (memory/cache replacement policy, prefetching, bandwidth at each hierarchy level, etc.), many of them being poorly documented.

The platform may also be modeled by measuring the performance of data transfers between pairs of resources, either within a single host or between hosts. Placement algorithms may then use this knowledge to apply weights to all pairs of cores when scheduling tasks [13]. This approach may even lead to experimentally rebuilding the entire platform topology for better task placement [14].

These ideas however lack a precise description of the structural model of the machine. Experimental measurement cannot ensure the reliable detection of the hierarchy of computing and memory resources such as packages, cores, shared caches and NUMA nodes. For instance, it may be difficult to distinguish cores that are physically close and cores that are slightly farther away but still inside the same processor. Also, the hierarchical organization of hardware resources does not impact performance in a uniform way. Different workloads suffer differently from remote NUMA accesses or cache contention. For instance a larger memory footprint increases the cache miss rate for a given cache size, causing the locality of the target NUMA node to become more important since it is actually accessed much more often.

Performance models only give hints about the impact of the platform on performance. If the application memory access patterns is known, one may guess the cache miss rate based on the memory footprint, reuse distance, cache size, etc. However, the actual impact of these parameters under a parallel load depends on many factors, including data set sizes, access patterns, sharing, etc. Building a performance model that covers all possible cases would lead to a combinatorial explosion. Models have to be simplified to remain usable.

On the other hand, the structural modeling of the platform easily gives precise information such as a cache or a memory link being shared by some of the cores. Even if their actual impact on performance cannot be precisely defined, using such information for task placement will help most workloads, or at least it will not slow them down.

OpenMP thread scheduling [6] or MPI process placement [11] are examples of scheduling opportunities that can benefit from deep platform topology knowledge. Indeed hardware resources are physically connected by links that may be modeled by a graph that applications may consult to better place their tasks and data.

C. Modeling the Hierarchy of Resources as a Tree

A straightforward way to model the structural organization of computing resources in a machine is to consider a hierarchical model: a computer contains processor packages, that contain cores, that optionally contain several hardware threads (with optional intermediate level such as AMD *Compute Unit*, see Figure 1).

Then, from a locality point of view, memory resources such as caches and NUMA nodes can be considered as embedded into such computing resources. Indeed NUMA nodes are usually attached to sets of cores[1], while caches are usually placed between some cores and the main memory,

Therefore we can sensibly extend a hierarchy of computing resources to a hierarchy of memory resources as depicted in Figure 2. This tree of resources is organized by physical inclusion and locality. The more the hardware threads share common ancestors (same NUMA node, shared caches, or even same core), the better locality between them is.

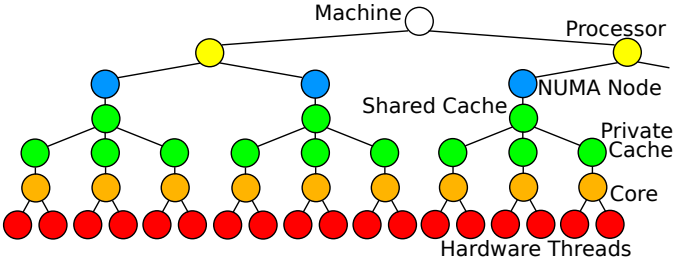


Fig. 2. Hierarchical modeling of a dual-processor host. Each processor contains 2 NUMA nodes with 3 dual-threaded cores each, and shared and private caches.

This approach has several advantages:

- First the tree representation of the topology is very convenient because it exposes the natural inclusion-based organization of the platform resources. Indeed binding memory near a core or finding a shared cache between cores only requires to walk up the tree until we find the relevant ancestor object and/or walk down to iterate over children. We can easily iterate over cores and threads close to a given object in the tree, a very common operation in locality-aware runtimes.

[1] Old architectures such as Itanium even had each NUMA node attached to several processors.

Another solution would be to represent the structure as a generic graph whose nodes are resources and edges are physical connections. However, the concepts of inclusion, container parent and contained children would not be obvious, making application queries less convenient.

- Secondly, this logical organization based on locality solves many portability issues by hiding platform specific parameters. Indeed vendor-specific configurations, BIOS or software upgrades may change the numbering of resources even if the processors are identical[2]. Therefore relying on hardware numbering of resources to perform explicit task placement makes programs non-portable, even to a similar platform with the same processors but a different BIOS. On the other hand, relying on resource locations in the logical tree does not suffer from such problems.
- Third, using a tree-structure is a good trade-off between performance and precision: while using a graph to represent the architecture would be more precise in some cases, tree algorithms are often much more efficient in both memory and time. Indeed, process placement techniques often rely on recursive graph partitioning techniques (a communication graph must be mapped onto an architecture graph) which are much more efficient when generic graphs are replaced with trees. For instance, the Scotch partitioning software supports a *Tleaf* architecture definition for enabling specifically optimized algorithms on hierarchical platforms [15].
- Finally, several operating systems[3] expose resource localities as bitmasks of smaller included objects (usually hardware threads). Hence building the hierarchical structure is straightforward.

D. Structural Modeling with hwloc

We explained in the previous section why structural and hierarchical modeling of platforms is a good compromise between precision and convenience. Such a modeling is already widespread in high-performance computing. We now present the actual implementation of this idea.

Locality-awareness became critical to most HPC applications in the last ten years. HPC runtimes (including OpenMP, MPI, task-graph programming, etc.) have to find out the number of CPU cores, accelerators, and NUMA nodes as well as their physical organization so as to properly distribute tasks and data buffers. Some software projects still use their own custom implementation for performing this topology discovery and binding tasks and memory. However it raises portability issues when it comes to supporting different platforms and operating systems. Indeed the ecosystem is very different between a BlueGene supercomputer [16] and a Linux cluster even if they run similar applications. Therefore there is a strong tendency towards delegating this work to specialized libraries.

Topology discovery and task binding was implemented in the former PLPA[4] and libtopology[5] libraries, as well as in the LIKWID performance analysis suite [12]. It is now at the core of the hwloc project [17]. hwloc is already used by many parallel libraries, HPC runtimes and resource managers. It became the *de facto* standard tool for discovering platform topologies and binding tasks and memory buffers in HPC[6].

hwloc builds a generic tree, from the root *Machine* object to the leaf *Processing Units* (hardware threads, logical processors) as depicted on Figure 1. It is based on the natural inclusive order of computing resources and the locality of memory resources. It can also build hierarchical *Groups* of NUMA nodes to better represent the large cc-NUMA machines (such as SGI Altix UV) based on the latency matrix reported by the hardware.[7]

The hwloc programming interface allows walking the tree edges to find neighbor resources. Walking up to ancestors or down to child objects lets applications find which NUMA nodes or processor cores are local. Iterating over objects of the same type (for instance when binding one process per core) is as easy as a breadth-first traversal of the tree. Given its widespread use in high-performance computing and its existing convenient API to retrieve locality information, hwloc looks like the natural candidate for studying how to expose the topology of new heterogeneous memory architectures.

III. Case Study with Intel Knights Landing Processor

We introduce in this section the software issues raised by the heterogeneous memory architecture of the upcoming Intel *Knights Landing* (KNL) processor.

[2] For instance, hardware threads #0 and #4 (called PU for *Processing Units*) are close in Figure 1 while #1 is located in another processor (not displayed).

[3] At least, Linux, AIX and Windows.

[4] http://www.open-mpi.org/projects/plpa

[5] http://libtopology.ozlabs.org

[6] http://www.open-mpi.org/projects/hwloc

[7] Some examples of distance-based grouping are presented at http://www.open-mpi.org/projects/hwloc/lstopo/

A. Memory Architecture of the KNL

From the memory point of view, the main innovation in the *Knights Landing* processor is the integration of a high-bandwidth memory in the package [18]. This MCDRAM (*Multi-Channel DRAM*) is faster than the usual off-package memory (DDR4). Cores in the processor therefore have direct access to two distinct local memories. This is a major change in HPC architecture. Indeed, previous architectures featured a single kind of CPU-accessible byte-addressable memory (e.g. DDR). Non-volatile memory used in storage systems such as disks cannot be directly accessed by the processors (a DMA to host memory is required first).

Additionally, the MCDRAM may be configured in 3 different modes:

- In *Flat* mode, it is exposed as a second NUMA node, distinct from the DDR4 NUMA node. Each processor core has two local NUMA nodes.
- In *Cache* mode, the MCDRAM serves as an additional cache level between the processor and the DDR (similar to a L3 cache). Each processor core has a single local NUMA node but there is an additional level of cache.
- Finally the *Hybrid* mode statically splits the MCDRAM into one *Flat* part and one *Cache* part.

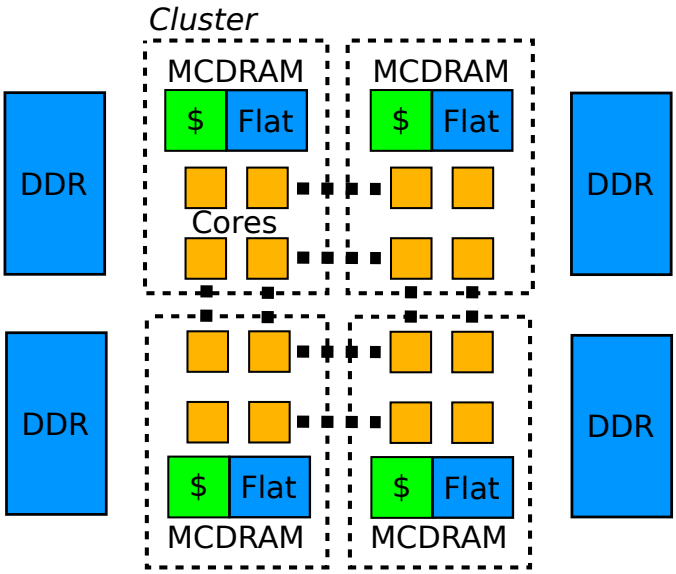


Fig. 3. Overview of the Intel *Knights Landing* architecture when configured in *Sub-NUMA Clustering* and *Hybrid* mode. Each *Cluster* (quarter of the processor) contains some cores, a part of the MCDRAM as local memory (*Flat*), a part of the MCDRAM as cache (*$*), and it is directly connected to a part of the DDR memory.

Moreover, the processor may be split into 4 pieces if the *Sub-NUMA Clustering* mode is enabled. Each cluster then contains a quarter of the cores. These cores have direct access to a quarter of the MCDRAM and a quarter of the DDR. As depicted on Figure 3, this creates an architecture with 8 distinct NUMA memory nodes – 2 local to each cluster – when MCDRAM is configured in *Flat* or *Hybrid* mode.

B. KNL Memory as exposed by the Linux kernel

We explained in the previous section that the MCDRAM is exposed as a separate NUMA node by the hardware (unless the *Cache* mode is enabled). Therefore the operating system should report that two NUMA nodes are local to each core. This is unfortunately not possible with the current Linux kernel which assumes that each CPU core is close only one NUMA node[8]. Indeed the ACPI SRAT specification (*System Resource Affinity Table*) was designed with the idea that each APIC ID (processor core identifier) is associated with a single *Proximity Domain* (NUMA node) [19].

The way KNL is currently exposed by the Linux kernel is to consider that processor cores are local to the DDR while the MCDRAM does not have any local core. Hence memory buffers are allocated to the DDR by default based on the usual *First Touch* policy (memory is physically allocated on the NUMA node close to the executing core). The advantage of this approach is that the smaller and faster MCDRAM memory cannot be used by mistake or wasted by the operating system for non-performance critical buffers. The MCDRAM may only be used when explicitly requested by applications.

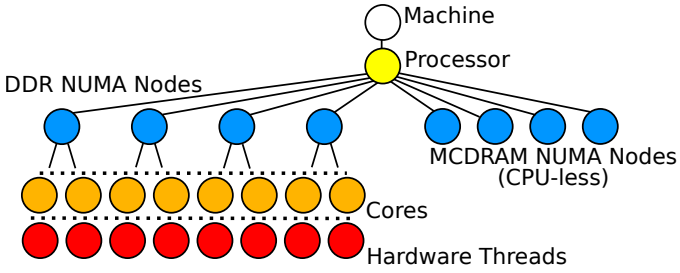


Fig. 4. Hierarchical modeling of Intel *Knights Landing* architecture when configured in *Sub-NUMA Clustering* and *Hybrid* mode as reported by the Linux kernel NUMA node attributes. Caches are not presented.

The drawback of this approach is that the application does not know which cores are actually close to the MCDRAM, as depicted on Figure 4. However, in the

[8] For instance the cpu_to_node() kernel internal function and the corresponding structures contain a single integer for describing the local NUMA node.

Sub-NUMA Clustering mode, there is a need to find out which one of the four MCDRAM NUMA nodes is close to which quarter of the cores. Also any multi-socket system with such heterogeneous memory in the future would raise the same issue even without clustering.

TABLE I
LATENCY MATRIX BETWEEN THE 8 NUMA NODES OF A KNL IN *Flat* AND *Sub-NUMA Clustering* MODES AS REPORTED BY THE ACPI SLIT TABLE (NORMALIZED RELATIVE LATENCIES). THE LOCAL MCDRAM QUARTER IS AT DISTANCE 31 FROM ITS LOCAL DDR, WHILE OTHERS ARE AT DISTANCE 41.

	DDR nodes				MCDRAM nodes			
	0	1	2	3	0	1	2	3
DDR0	10	21	21	21	31	41	41	41
DDR1	21	10	21	21	41	31	41	41
DDR2	21	21	10	21	41	41	31	41
DDR3	21	21	21	10	41	41	41	31
MCDRAM0	31	41	41	41	10	41	41	41
MCDRAM1	41	31	41	41	41	10	41	41
MCDRAM2	41	41	31	41	41	41	10	41
MCDRAM3	41	41	41	31	41	41	41	10

This information may be found in the ACPI SLIT table (*System Locality Information Table*) which provides relative theoretical latencies between NUMA nodes [19]. Table I shows that finding the MCDRAM close to a given DDR node is indeed easy, which means finding the MCDRAM local to some cores is feasible. However this table does not really respect the actual ACPI SLIT specification. Indeed the local MCDRAM latency is not actually 3.1 times higher than the local DDR latency (31 against 10 in the table).[9] The table is coherent with the idea of showing the MCDRAM as CPU-less to avoid memory allocation by mistake, but it is not coherent with the actual performance of the memory subsystem. Applications that already use the ACPI SLIT table[10] for precise NUMA topology management will be confused on KNL. They need some KNL-specific changes to support this new architecture.

This unobvious way to expose KNL memory affinity was designed to be compatible with the existing ACPI specification and Linux kernel implementation. And it guarantees that memory allocations would not go to the MCDRAM by default. Newer hardware specifications are being developed to better expose these memory architectures but they are not publicly available yet. In the meantime, hardware vendors may expose custom information through the SMBIOS tables [20] but this information is not available to unprivileged user-space. The Linux kernel still needs to be modified to parse these tables and expose new memory attributes to user-space, for instance as new virtual files under /sys/bus/node/devices/node*/.

Another issue with the way the KNL architecture is exposed in software is the MCDRAM in *Hybrid* mode. This cache only applies to DDR memory accesses. Memory accesses going to the MCDRAM NUMA node do not go through the MCDRAM part that acts as a cache. Unfortunately, again, the Linux kernel currently has no way to report this information. Cache attributes exposed in /sys/bus/cpu/devices/cpu*/cache/index*/ have a CPU affinity attribute (shared_cpu_map). However no attribute says which NUMA nodes are close to this cache, and which NUMA node accesses go through this cache.

C. User-Space Tools for Exposing KNL Memory Locality

We explained in the previous section that the hardware and current operating systems cannot precisely expose the locality of the memory nodes of the KNL architecture in a portable way. We now discuss the existing ways to manage this locality in high performance computing applications.

Memory locality information is available from the Linux kernel through numerous sysfs virtual files. However reading these files requires a lot of work that should rather be factorized in a library. libnuma[11] is the official way to manage NUMA-ness on Linux. However this library does not try to bring any locality information besides what is in sysfs virtual files. It reports the hardware information in a C programming interface but it cannot improve the information with KNL-specifics such as finding the local MCDRAM node. Besides libnuma is Linux-specific, and it does not cover anything but hardware threads and NUMA nodes. Processor, core, and cache affinities are not provided, while they are widely used in HPC runtime when placing tasks according to the hardware hierarchy [11], [6], [21].

The memkind library[12] was developed specifically to address KNL-like memory architectures. It offers an API to allocate memory from different memory pools, on different kinds of memory, backed by normal or huge pages, etc. [22] Unfortunately, memkind does not cover

[9] The MCDRAM latency is *similar* to DDR, but possibly smaller under load.

[10] Available in /sys/bus/node/devices/node*/distance on Linux.

[11] http://oss.sgi.com/projects/libnuma/

[12] https://github.com/memkind/memkind

the entire spectrum of memory management requirements on usual HPC architectures. For instance, it cannot allocate memory on non-local NUMA or interleave pages between multiple nodes. Again, these features are often used in HPC [23], [24].

Most HPC runtimes actually use hwloc to discover the topology of parallel platforms and bind tasks and memory. Such higher-level tools expose the hierarchy and locality of hardware resources in a portable and abstracted manner. Hence they are good candidates for reworking and better exposing the unobvious locality information reported by the operating system.

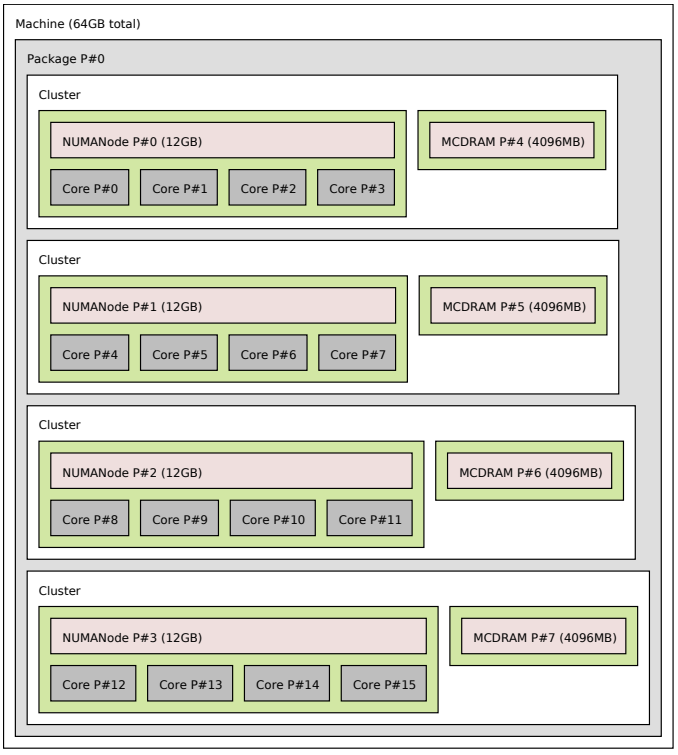


Fig. 5. hwloc 1.11.3's view of the KNL memory architecture when adding Cluster objects to group local MCDRAM and DDR NUMA node objects. Only 4 cores per cluster are presented. Caches and hardware threads are also hidden.

As explained in Section II-C, hwloc exposes the hardware resource hierarchy as a tree. It suffers from the issues exposed on Figure 4. As a first work-around for the missing MCDRAM locality, we added KNL-specific quirks to the hwloc library using the ACPI SLIT latencies.[13] Figure 5 shows that an additional intermediate *Cluster* level is added to the hierarchy so that MCDRAM and DDR of the same quarter of the processor remain together in the tree.

[13] This work is already available in hwloc releases since v1.11.3.

An application using hwloc and willing to find the local MCDRAM now just has to walk up the tree until finding the DDR and them use its neighbor leaf. Object attributes are also used to tell applications about the MCDRAM performance and size. Additionally, we modified hwloc to hide the KNL latency matrix so that applications do not get confused by the MCDRAM latency being reported higher than the DDR.

We envision the following software stack for managing affinities on KNL.

- First, kernel and early user-space allocations may only be managed by the operating system using the default memory allocation policy, which uses the normal DDR memory.
- Affinity-aware tasks and data placement in HPC runtimes and parallel libraries still rely on the *de facto* standard hwloc library for getting a global view of the hierarchy of computing and memory resources.
- Fine-grain memory allocations may then be implemented with memkind.

There are some thoughts about porting memkind over hwloc so that memkind focuses on specialized mallocs for different kinds of memory and pages, while hwloc takes care on locating the corresponding hardware resources.

IV. Exposing Heterogeneous Memory Locality

We now take a step back from the specific case of KNL and look at the general case of upcoming heterogeneous memory architectures. The aim is to envision a software structural platform model for exposing locality in a portable and convenient way. It should be generic enough to cover upcoming memory architecture changes. It must also answer the currently existing needs of many locality-aware parallel libraries in terms of locating hardware resources and placing tasks and memory buffers. This includes placement algorithms walking the tree in recursive top-down manner as well as simple queries such as finding the local memory nodes, the local cores, or specifics such as shared cache sizes.

A. Upcoming Memory Architecture Changes

Beside the KNL architecture, there are several hints about the upcoming memory changes. First, most processor vendors have announced their own *High Bandwidth Memory* implementation (HBM). These technologies increase memory access performance. However they cannot scale to hundreds of gigabytes yet for cost and

dimension reasons. Hence it is expected that normal off-package memory will still be used as a slower but larger volatile memory pool. Additionally, non-volatile memory is announced as the next storage revolution [25], [26]. Byte-addressable NVDIMMs will be directly attached to processors just like current volatile memory DIMMs [27]. **Servers may thus have two or three kinds of memory local to each processor socket in the future (normal, high-bandwidth, and non-volatile).** Applications will need ways to identify the kind and locality of each of them.

The next expected memory architecture change regards cache hierarchies. Memory-side caches have been used in the past as eDRAM connected to external memory controllers (for instance on POWER8 platforms [28]). They are basically invisible to the operating system. However applications may still want to be aware of them because memory footprint or algorithms can be tuned according to cache sizes [29]. When KNL is configured in *Hybrid* mode, the MCDRAM cache part can be considered a memory-side cache. Moreover, it only caches accesses to the DDR, not those to the MCDRAM flat memory. **Memory-side caches are specific to some physical memory regions, while caches are usually CPU-side (restricted to some cores but applied to all memory accesses).** There are indeed some works on 2-level memory hierarchies where *near*-memory can optionally be used as a *far*-cache [30].

Then comes the question of the memory interconnect between all these processors and memories. Aside of the specific market of very large NUMA architectures such as SGI Altix UV, it seems that servers now rarely have more than two sockets, especially in high-performance computing. However the NUMA interconnect now extends inside processors due to the increasing number of cores. Indeed most modern processors[14] are actually organized as two internal NUMA nodes. For instance, Intel Xeon are now made of a two rings interconnecting two sets of cores and two memory controllers [31]. The *Sub-NUMA Clustering* mode of KNL follows the same trend. The NUMA interconnect therefore usually looks like a hierarchy made of an inter-socket network with small intra-processor interconnects. However, this point does really change the requirements on the software. The Linux kernel may already expose the ACPI SLIT latency matrix. And software such as hwloc already use it to build a hierarchical organization by grouping close NUMA nodes. This tree representation cannot be a perfect match for non-hierarchical NUMA interconnects but the application may still query the latency matrix to get the exact topology information if needed.

[14] At least Intel Xeon E5, IBM POWER8, AMD Opteron, and Fujitsu Sparc XIfx.

B. Extending the Resource Tree for Memory-Side Hierarchies

We explained in the previous section that the software stack has to adapt to different kinds on memory connected to each processor, and caches applying to only some of these memories. Contrary to the Linux kernel, user-space tools such as hwloc already have the ability to precisely describe such details. Indeed each resource locality is described by a set of close processing units and/or a set of close memories.

We still focus on a structural tree representation because it has many advantages for HPC users as explained in Section II-C even if it is not perfect. Figure 4 shows that we were able to slightly improve hwloc to better expose the MCDRAM locality on KNL. However this workaround is not actually satisfying for existing applications. First, placement algorithms usually walk down the tree to spread application tasks according to intermediate node arities (the set of tasks is split according to the number of processors, then subsets are split again according to the number of cores per processor, etc.). They are confused on KNL because they should split tasks between the DDR and MCDRAM, while the corresponding cores are actually the same. Secondly, the model proposed on Figure 4 lies by showing no cores that are local to the MCDRAM.

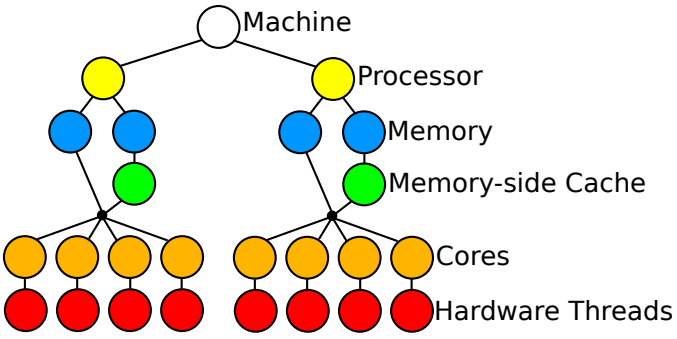


Fig. 6. Non-hierarchical modeling of a dual-processor architecture with two kinds of memories connected to each processor. Memory-side caches apply to only one of them.

Cores should indeed appear close to both the MCDRAM and DDR. Also the MCDRAM cache should appear as a memory-side cache that applies to the DDR only. The model should therefore rather look

like Figure 6. Unfortunately this representation is not a hierarchical tree anymore. It breaks many existing use cases where applications walk up the tree to find parent containing cores (processors) or close to cores (shared caches or NUMA nodes).

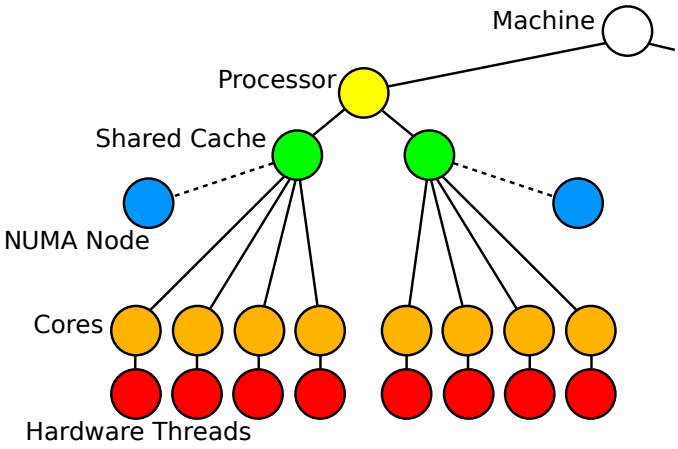


Fig. 7. Hierarchical modeling of a dual-processor architecture with two NUMA nodes in each processor. NUMA nodes are moved to the memory-side (dashed edges) outside of the main compute hierarchy, and attached to the object that share the same local resources (a shared cache here).

We rather propose to move the memory-side hierarchy back out of the main compute hierarchy. Inserting CPU caches and NUMA nodes inside the hierarchy of computing units (see Figure 1) is actually useful for showing some sharing of resources (caches, local access to certain part of the memory, etc.). However it does not mean that these shared resources must appear inside the tree itself, as parents of these processor cores. What is important to locality-aware HPC applications is:

- the structural hierarchy of cores (for recursive top-down task placement);
- the relative locality of cores and memory (for relative placement of tasks and memory with respect to each other).

Hence the hwloc tree is now based on the compute-side hierarchy including processor packages, cores, hardware threads, and CPU-side caches. Memory-side hierarchies become new leaves that are attached to the compute-side node according to their locality. For instance, a single processor with a single NUMA node would show that NUMA node object as a single memory-side leaf below the processor package object. If the processor contains two NUMA nodes, each node is attached to the corresponding level of sharing in the compute hierarchy (usually a shared cache as in Figure 7). Architectures with different kinds of memory and

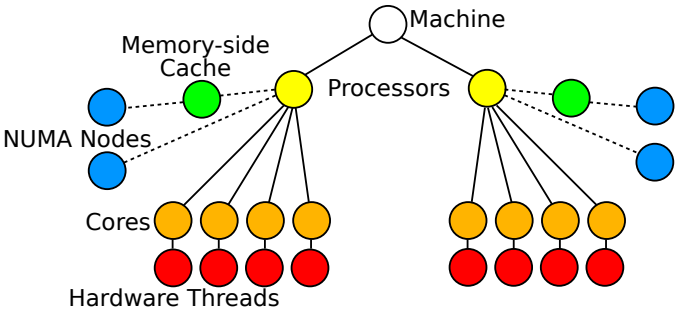


Fig. 8. Hierarchical modeling of a dual-processor architecture with two kinds of memories connected to each processor, and memory-side caches applying to only one of them. Each NUMA node is connected to its local processor, either directly or through a memory-side cache.

some memory-side caches (for instance KNL) now use multiple memory-side objects as shown on Figure 8. If needed, an intermediate CPU-side hierarchy level could also be added to properly attach the memory objects if their locality only corresponds to a specific part of a processor.

This model extends the hierarchical structural modeling to cope with heterogeneous memory architectures without breaking many locality-aware HPC runtimes. Finding the NUMA nodes that are close to a given core is now just a matter of walking up the tree until a parent has some memory-side children, and walking down these memory-side edges. Finding the cores that are close to a given memory, either DDR or HBM, just consist in walking up the memory edges up to a compute-side parent, and walking down to cores.

C. Non-Volatile Memory

Heterogeneous memories may include both technologies with different performance, and with different characteristics such as volatility. Upcoming byte-addressable non-volatile memories (NVDIMMs) may be exposed to applications as normal memory nodes as described in the previous section. However operating systems such as Linux are rather going to expose them through storage interfaces [32]. Indeed, after reboot, applications need a way to find out where and how their data is stored in this persistent storage. File-systems have been designed exactly for this purpose. Hence, applications will not be able to directly allocate memory for the non-volatile pool. They will rather have to create files and map them in virtual memory so that the non-volatile memory becomes directly accessible to the CPU and byte-addressable.

NVDIMMs are exposed by Linux as *pmem* block devices just like any other storage device, even if the

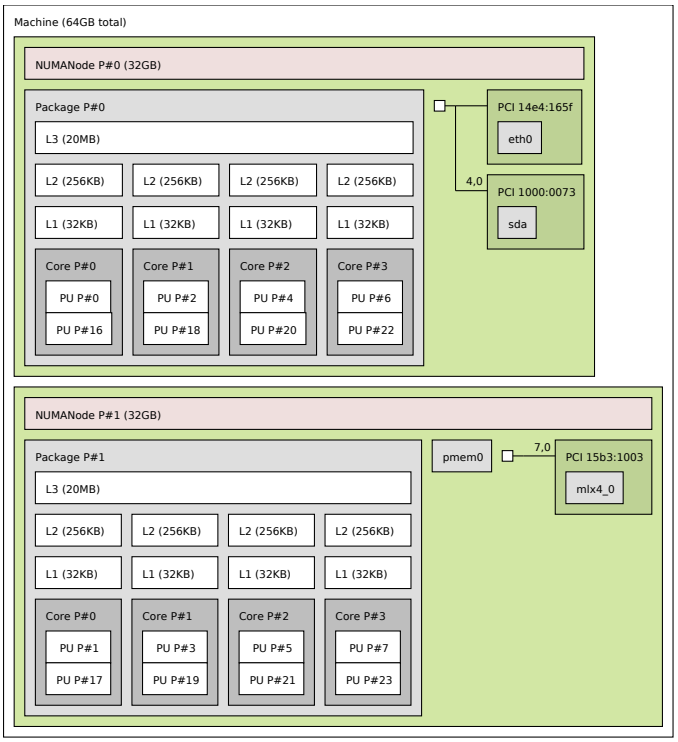


Fig. 9. Intel dual-socket quad-core hyper-threaded platform with I/O devices as shown by hwloc's `lstopo` tool. The non-volatile byte-addressable NVDIMM block device *pmem0* is attached to its local processor package just like PCI devices (*eth0* network interface, *sda* disk and *mlx4_0* InfiniBand HCA).

granularity may be a single byte. The Linux kernel already exposes the locality of these devices[15]. And user-space tools such as libnuma and hwloc can already use it to help applications bind tasks and memory buffers close to such devices.

hwloc represents I/O devices using I/O-specific edges that connect PCI and non-PCI I/O hierarchies to the compute-side hierarchy based on locality (see Figure 9). In fact, this is similar to our memory-side proposal in Section IV-B. The compute hierarchy is at the core of the structural platform model with memory and I/O devices attached according to their physical locality.

V. Conclusion and Future Works

The increasing complexity of parallel platforms and the gap between theoretical and application performance raised the need to better understand the hardware. Many HPC runtimes use locality information to place tasks and data according to their affinities. The structural modeling of the platform is often used as a way to find out which cores and NUMA nodes are close to each-other. We explained why a hierarchical modeling as a tree is a good compromise between precision and convenience for HPC applications. Even if a tree is not a prefect match for certain NUMA interconnects, it provides very nice algorithmic properties without lacking significant performance properties.

We then used the upcoming Intel Knights Landing memory architecture to reveal multiple issues in the current software stack. The Linux kernel affinity attribute files are not expressive enough, and they were not designed for processors with multiple local NUMA nodes. Moreover, user-space libraries such as libnuma, memkind and hwloc all have their own limits. We described a workaround used by current hwloc releases since 1.11.3 to better represent the KNL memory model.[16] We then proposed a new way to extend the hwloc hierarchical tree to upcoming heterogeneous memory architectures and memory-side caches. Our solution maintains convenience for finding local resources and applying recursive top-down placement algorithms while correctly exposing the topology of these new architectures.

We are now working at integrating our proposal in the future hwloc 2.0 release. Future works include further rethinking the model to better support non-hierarchical architectures or even dynamically-reconfigurable hardware.

Acknowledgments

We would like to thank Intel for providing us with hints for designing our new hwloc model.

References

[1] C. Lameter, "NUMA (Non-Uniform Memory Access): An Overview," *ACM Queue*, vol. 11, p. 40, 2013, http://queue.acm.org/detail.cfm?id=2513149.

[2] T. Brecht, "On the Importance of Parallel Application Placement in NUMA Multiprocessors," in *Proceedings of the Fourth Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, San Diego, CA, Sep. 1993.

[3] N. Robertson and A. Rendell, "OpenMP and NUMA Architectures I: Investigating Memory Placement on the SGI Origin 3000," in *3rd International Conference on Computational Science*, ser. Lect. Notes in Comp. Science, S. Verlag, Ed., vol. 2660, 2003, pp. 648–656.

[4] R. Yang, J. Antony, P. P. Janes, and A. P. Rendell, "Memory and Thread Placement Effects as a Function of Cache Usage: A Study of the Gaussian Chemistry Code on the SunFire X4600 M2," in *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*, 2008, pp. 31–36.

[15] The closest NUMA node to block device *name* is reported in /sys/class/block/⟨name⟩/device/numa_node.

[16] Available from from https://www.open-mpi.org/software/hwloc/v1.11/.

[5] M. M. Tikir and J. K. Hollingsworth, "Using Hardware Counters to Automatically Improve Memory Performance," in *Proceedings of the ACM/IEEE SC2004 Conference on High Performance Networking and Computing*. Pittsburgh, PA: IEEE Computer Society, Nov. 2004, p. 46.

[6] F. Broquedis, N. Furmento, B. Goglin, P.-A. Wacrenier, and R. Namyst, "ForestGOMP: an efficient OpenMP environment for NUMA architectures," *International Journal on Parallel Programming, Special Issue on OpenMP; Guest Editors: Matthias S. Mller and Eduard Ayguad*, vol. 38, no. 5, pp. 418–439, 2010. [Online]. Available: http://hal.inria.fr/inria-00496295

[7] F. Song, S. Moore, and J. Dongarra, "Analytical Modeling and Optimization for Affinity Based Thread Scheduling on Multicore Systems," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing*, New Orleans, LA, Aug. 2009.

[8] Z. Majo and T. R. Gross, "Memory Management in NUMA Multicore Systems: Trapped between Cache Contention and Interconnect Overhead," in *Proceedings of the International Symposium on Memory Management (ISMM'11)*. San Jose, CA: ACM, Jun. 2011, pp. 11–20.

[9] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter, "Characterization of Scientific Workloads on Systems with Multi-Core Processors," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, San Jose, CA, 2006.

[10] M. M. Tikir and J. K. Hollingsworth, "Hardware Monitors for Dynamic Page Migration," *Journal of Parallel and Distributed Computing*, vol. 68, pp. 1186–1200, 2008.

[11] E. Jeannot, G. Mercier, and F. Tessier, "Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 993–1002, 4 2014.

[12] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, Sept 2010, pp. 207–216.

[13] E. Rodrigues, F. Madruga, P. Navaux, and J. Panetta, "Multicore aware process mapping and its impact on communication overhead of parallel applications," in *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, July 2009, pp. 811–817.

[14] J. González-Domínguez, G. L. Taboada, B. B. Fraguela, M. J. Martín, and J. Touriño, "Automatic Mapping of Parallel Applications on Multicore Architectures using the Servet Benchmark Suite," *Computers and Electrical Engineering*, vol. 38, pp. 258–269, 2012.

[15] F. Pellegrini, "Scotch and libScotch 5.1 User's Guide," Dec. 2010, https://gforge.inria.fr/docman/view.php/248/7104/scotch_user5.1.pdf.

[16] M. Giampapa, T. Gooding, T. Inglett, and R. W. Wisniewski, "Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2010.

[17] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. 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)*. Pisa, Italia: IEEE Computer Society Press, Feb. 2010, pp. 180–186. [Online]. Available: http://hal.inria.fr/inria-00429889

[18] Intel Corporation, "What public disclosures has Intel made about Knights Landing?" https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing.

[19] "Advanced Configuration and Power Interface Specification, Revision 5.0a," Nov. 2013, http://www.acpi.info/spec50a.htm.

[20] Distributed Management Task Force, Inc., "System Management BIOS (SMBIOS) Reference Specification," http://www.dmtf.org/standards/smbios.

[21] J. Hursey and J. M. Squyres, "Advancing Application Process Affinity Experimentation: Open MPI's LAMA-Based Affinity Interface," in *Recent Advances in the Message Passing Interface. The 20th European MPI User's Group Meeting (EuroMPI 2013)*. Madrid, Spain: ACM, Sep. 2013, pp. 163–168.

[22] C. Cantalupo, V. Venkatesan, J. R. Hammond, and S. Hammond, "User Extensible Heap Manager for Heterogeneous Memory Platforms and Mixed Memory Policies," 2015, http://memkind.github.io/memkind/memkind_arch_20150318.pdf.

[23] M. Castro, L. G. Fernandes, C. Pousa, J.-F. Mehaut, and M. S. de Aguiar, "NUMA-ICTM: A Parallel Version of ICTM Exploiting Memory Placement Strategies for NUMA Machines," in *PDSEC-09: The 10th International Workshop on Parallel and Distributed Scientific and Engineering Computing, held in conjunction with IPDPS 2009*. Rome, Italy: IEEE Computer Society Press, May 2009.

[24] A. Srinivasa and M. Sosonkina, "Nonuniform Memory Affinity Strategy in Multithreaded Sparse Matrix Computations," in *Proceedings of the 2012 Symposium on High Performance Computing*, ser. HPC '12. San Diego, CA, USA: Society for Computer Simulation International, 2012, pp. 9:1–9:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2338816.2338825

[25] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.56

[26] D. Li, J. S. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu, "Identifying Opportunities for Byte-Addressable Non-Volatile Memory in Extreme-Scale Scientific Applications," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 945–956.

[27] Storage Networking Industry Association, "NVDIMM Special Interest Group," http://www.snia.org/forums/sssi/NVDIMM.

[28] W. J. Starke, J. Stuecheli, D. M. Daly, J. S. Dodson, F. Auernhammer, P. M. Sagmeister, G. L. Guthrie, C. F. Marino, M. Siegel, and B. Blaner, "The cache and memory subsystems of the IBM POWER8 processor," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 3:1–3:13, Jan 2015.

[29] J.-S. Park, M. Penner, and V. K. Prasanna, "Optimizing Graph Algorithms for Improved Cache Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 769–782, Sep. 2004. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2004.44

[30] R. Ramanujan, G. Hinton, and D. Zimmerman, "Dynamic partial power down of memory-side cache in a 2-level memory hierarchy," Oct. 9 2014, intel Corporation. US Patent App. 13/994,726. [Online]. Available: https://www.google.com/patents/US20140304475

[31] B. Bowhill, B. Stackhouse, N. Nassif, Z. Yang, A. Raghavan, C. Morganti, C. Houghton, D. Krueger, O. Franza, J. Desai, J. Crop, D. Bradley, C. Bostak, S. Bhimji, and M. Becker, "The Xeon processor E5-2600 v3: A 22nm 18-core product family," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015.

[32] A. Rudoff, "pmem.io - Persistent Memory Programming - Crawl, Walk, Run..." http://pmem.io/2014/08/27/crawl-walk-run.html.