Edinburgh Research Explorer

An Architecture for High Performance Computing and Data Systems using Byte-Addressable Persistent Memory

Link:

Document Version:
Peer reviewed version

Published In:
High Performance Computing

An Architecture for High Performance Computing and Data Systems using Byte-Addressable Persistent Memory

Adrian Jackson[1], Michèle Weiland[1], Mark Parsons[1], and Bernhard Homölle[2]

[1] EPCC, The University of Edinburgh, Edinburgh, United Kingdom
[2] SVA System Vertrieb Alexander GmbH, Paderborn, Germany

Abstract. Non-volatile and byte-addressable memory technology with performance close to main memory has the potential to revolutionise computing systems in the near future. Such memory technology provides the potential for extremely large memory regions (i.e. > 3TB per server), very high performance I/O, and new ways of storing and sharing data for applications and workflows. This paper proposes hardware and system software architectures that have been designed to exploit such memory for High Performance Computing and High Performance Data Analytics systems, along with descriptions of how applications could benefit from such hardware, and initial performance results on a system with Intel Optane DC Persistent Memory.

Keywords: Non-volatile memory · persistent memory · system architecture · systemware · NVRAM · B-APM

1 Introduction

There are a number of new memory technologies that are impacting, or likely to impact, computing architectures in the near future. One example of such a technology is so called high bandwidth memory, already featured on Intel's latest many-core processor, the Xeon Phi Knights Landing [1], and NVIDIA's latest GPU, Volta [2]. These contain MCDRAM [1] and HBM2 [3] respectively, memory technologies built with traditional DRAM hardware but connected with a very wide memory bus (or series of buses) directly to the processor to provide very high memory bandwidth when compared to traditional main memory (DDR channels).

This has been enabled, in part, by the hardware trend for incorporating memory controllers and memory controller hubs directly onto processors, enabling memory to be attached to the processor itself rather than through the motherboard and associated chipset. However, the underlying memory hardware is the same, or at least very similar, to the traditional volatile DRAM memory that is still used as main memory for computer architectures, and that remains attached to the motherboard rather than the processor.

Non-volatile memory, i.e. memory that retains data even after power is turned off, has been exploited by consumer electronics and computer systems for many

years. The flash memory cards used in cameras and mobile phones are an example of such hardware, used for data storage. More recently, flash memory has been used for high performance data input/output (I/O) in the form of Solid State Disk (SSD) drives, providing higher bandwidth and lower latency than traditional Hard Disk Drives (HDD).

Whilst flash memory can provide fast I/O performance for computer systems, there are some drawbacks. It has limited endurance when compare to HDD technology, restricted by the number of modifications a memory cell can undertake and thus the effective lifetime of the flash storage [29]. It is also generally more expensive than other storage technologies. However, SSD storage, and enterprise level SSD drives, are heavily used for I/O intensive functionality in large scale computer systems because of their random read and write performance capabilities.

Byte-addressable random access persistent memory (B-APM), also known as storage class memory (SCM), NVRAM or NVDIMMs, exploits a new generation of non-volatile memory hardware that is directly accessible via CPU load/store operations, has much higher durability than standard flash memory, and much higher read and write performance. B-APM, with its very high performance I/O characteristics, and vastly increased capacity (compared to volatile memory), offers a potential hardware solution to enable the construction of a compute platform that can support High-performance computing (HPC) and high-performance data analytics (HPDA) use cases, with high performance processors, very large amounts of B-APM in compute nodes, and a high performance network, providing a scalable compute, memory, and I/O system, and moving to address some of the performance imbalance systems currently have between memory and I/O performance.

In this paper, we outline the systemware and hardware required to provide such a system, and discuss prelimarily performance results from just such a system. We start by describing persistent memory, and the functionality it provides, in more detail in section 2. In section 3 we discuss how B-APM could be exploited for scientific computation or data analytics. Following this we outline our proposed hardware and systemware architectures in sections 4 and 5. We finish by presenting performance results on a prototype system containing Intel Optane DC Persistent memory, in secont 6, discussing related work in section 7, and summarise the paper in the final section.

2   Persistent Memory

B-APM takes new non-volatile memory technology and packages it in the same form factor (i.e. using the same connector and dimensions) as main memory (SDRAM DIMM form factor). This allows B-APM to be installed and used alongside DRAM based main memory, accessed through the same memory controller. As B-APM is installed in a processor's memory channels, applications running on the system can access B-APM directly in the same manner as main memory, including true random data access at byte or cache line granularity.

Such an access mechanism is very different to the traditional block based approaches used for current HDD or SSD devices, which generally requires I/O to be done using blocks of data (i.e. 4KB of data written or read in one operation), and relies on expensive kernel interrupts and context switches.

The first B-APM technology to make it to market is Intel and Micron's Optane DC Persistent^{TM}memory [5]. The performance of this B-APM is lower than main memory (with a latency $\sim$5-10x that of DDR4 memory when connected to the same memory channels), but much faster than SSDs or HDDs. It is also much larger capacity than DRAM, around 2-5x denser (i.e. 2-5x more capacity in the same form factor, with 128, 256, and 512GB currently available DIMMs).

2.1   Data Access

This new class of memory offers very large memory capacity for servers, as well as long term persistent storage within the memory space of the servers, and the ability to undertake I/O in a new way. B-APM can enable synchronous, byte level, direct access (DAX) to persistent data , moving away from the asynchronous block-based file I/O applications currently rely on. In current asynchronous I/O user applications pass data to the operating system (OS) which then use driver software to issue an I/O command, putting the I/O request into a queue on a hardware controller. The hardware controller will process that command when ready, notifying the OS that the I/O operation has finished through an interrupt to the device driver.

B-APM, on the other hand, can be accessed simply by using a load or store instruction, as with any other memory operation from an application or program. However, because B-APM can provide persistence functionality (allowing data to be accessible after power loss), some further considerations are required if persistent is to be guaranteed. Applications must also ensure stored data has been flush from the volatile CPU caches and has arrived on the non-volatile medium (using new cache flush commands and fence instructions to ensure stores are ordered ordered before subsequent instructions) before they can confirm data has been persisted (although this flush may only be required to the memory controller, rather than the non-volatile medium, if using enhanced power supply functionality such as asynchronous DRAM self-refresh [6]).

With B-APM providing much lower latencies than external storage devices, the traditional I/O block access model, using interrupts, becomes inefficient because of the overhead of context switches between user and kernel mode (which can take thousands of CPU cycles [30]). Furthermore, in the future it may become possible to implement remote persistent access to data stored in the memory using RDMA technology over a suitable interconnect. Using high performance networks has the potential to enable access to data stored in B-APM in remote nodes faster than accessing local high performance SSDs via traditional I/O interfaces and stacks inside a node.

Therefore, it is possible to use B-APM to greatly improve I/O performance within a server; increase the memory capacity of a server; or provide a remote data store with high performance access for a group of servers to share. Such

storage hardware can also be scaled up by adding more B-APM memory in a server, or adding more nodes to the remote data store, allowing the I/O performance of a system to scale as required. The use of B-APM in compute nodes also removes competition for I/O resources between jobs in a system, isolating application I/O traffic and removing the performance fluctuations associated with I/O users often experience on shared HPC systems [25]. However, if B-APM is provisioned in the servers, there must be software support for managing data within the B-APM. This includes moving data as required for the jobs running on the system, and providing the functionality to let applications run on any server and still utilise the B-APM for fast I/O and storage (i.e. applications should be able to access B-APM in remote nodes if the system is configured with B-APM only in a subset of all nodes).

As B-APM is persistent, it also has the potential to be used for resiliency, providing backup for data from active applications, or providing long term storage for databases or data stores required by a range of applications. With support from the systemware, servers can be enabled to handle power loss without experiencing data loss, efficiently and transparently recovering from power failure and resuming applications from their latest running state, and maintaining data with little overhead in terms of performance.

2.2   B-APM modes of operation

Ongoing developments in memory hierarchies, such as the high bandwidth memory in Xeon Phi manycore processors or NVIDIA GPUS, have provided new memory models for programmers and system designers/implementers. A common model that has been proposed includes the ability to configure main memory and B-APM in two different modes: Single-level and Dual-level memory [8].

Single-level memory, or SLM, has main memory (DRAM) and B-APM as two separate memory spaces, both accessible by applications, as outlined in Figure 1. This is very similar to the Flat Mode [7] configuration of the high bandwidth, on-package, MCDRAM in Intel Knights Landing processor. The DRAM is allocated and managed via standard memory API's such as *malloc* and represent the OS visible main memory size. The B-APM is be managed by programming APIs and presents the non-volatile part of the system memory. In order to take advantage of B-APM in SLM mode, systemware or applications have to be adapted to use these two distinct address spaces.

Dual-level memory, or DLM, configures DRAM as a cache in front of the B-APM, as shown in Figure 2. Only the memory space of the B-APM is available to applications, data being used is stored in DRAM, and moved to B-APM when no longer immediately required by the memory controller (as in standard CPU caches). This is very similar to the Cache Mode [7] configuration of MCDRAM on KNL processors.

This mode of operation does not require applications to be altered to exploit the capacity of B-APM, and aims to give memory access performance at main memory speeds whilst providing access to the large memory space of B-APM. However, exactly how well the main memory cache performs will depend on
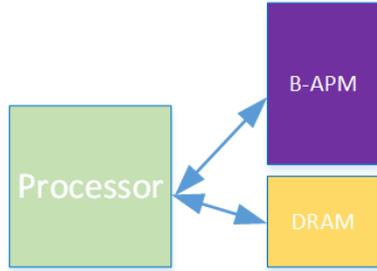
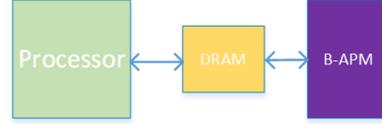**Fig. 1.** Single-level memory (SLM) configuration using main memory and B-APM

Fig. 2. Dual-level memory (DLM) configuration using main memory and B-APM

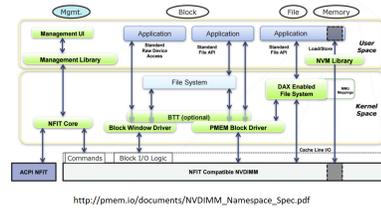http://pmem.io/documents/NVDIMM_Namespace_Spec.pdf

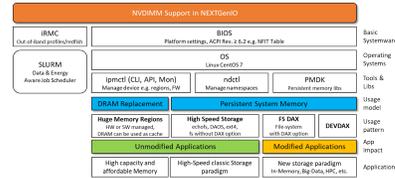**Fig. 3.** PMDK software architecture

Fig. 4. Software stack exploiting B-APM in compute nodes

the specific memory requirements and access pattern of a given application. Furthermore, persistence of the B-APM contents cannot be longer guaranteed, due to the volatile DRAM cache in front of the B-APM, so the non-volatile characteristics of B-APM are not exploited.

2.3   Non-volatile memory software ecosystem

The Storage Networking Industry Association (SNIA) have produced a software architecture for B-APM with persistent load/store access, formalised in the Linux Persistent Memory Development Kit (PMDK) [9] library. This approach re-uses the naming scheme of files as traditional persistent entities and maps the B-APM regions into the address space of a process (similar to memory mapped files in Linux). Once the mapping has been done, the file descriptor is no longer needed and can be closed. Figure 3 outlines the PMDK software architecture. Figure 4 details the software architecture we are considering for systems exploiting B-APM for HPC and HPDA work, which will be discussed in more detail in section 5.

3   Opportunities for exploiting B-APM for computational simulations and data analytics

Reading data from and writing it to persistent storage is usually not the most time consuming part of computational simulation applications. Analysis of common applications from a range of different scientific areas shows that around 5-20% of runtime for applications is involved in I/O operations [10] [11]. It is evident that B-APM can be used to improve I/O performance for applications by replacing slower SSDs or HDDs in external filesystems. However, such a use of B-APM would be only an incremental improvement in I/O performance, and would neglect some of the significant features of B-APM that can provide performance benefits for applications.

Firstly, deploying B-APM as an external filesystem would require provisioning a filesystem on top of the B-APM hardware. Standard storage devices require a filesystem to enable data to be easily written to or read from the hardware. However, B-APM does not require such functionality, and data can be manipulated directly on B-APM hardware simply through load/store instructions. Adding the filesystem and associated interface guarantees (i.e. POSIX interface [12]) adds performance overheads that will reduce I/O performance on B-APM.

Secondly, an external (to the compute nodes) B-APM based filesystem would require all I/O operations to be performed over a network connection (see Figure 5). This would limit the maximum performance of I/O to that of the network between compute nodes and the nodes the B-APM is hosted in, and expose application I/O performance to the variations associated with a shared external resource, however fast it is.
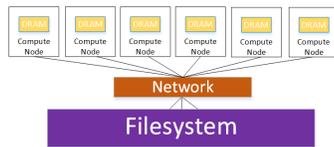
Fig. 5. Current external storage for HPC and HPDA systems
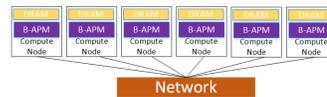
Fig. 6. Internal storage using B-APM in compute nodes for HPC and HPDA systems

Our vision for exploiting B-APM for HPC and HPDA systems is to incorporate the B-APM into the compute nodes, as outlined in Figure 6. This architecture allows applications to exploit the full performance of B-APM within the compute nodes they are using, by enabling access to B-APM through load/store operations at byte-level granularity, as opposed to block based, asynchronous I/O. Incorporating B-APM into compute nodes also has the benefit that I/O capacity and bandwidth can scale with the number of compute nodes in the system. Adding more compute nodes will increase the amount of B-APM in the system and add more aggregate bandwidth to I/O/B-APM operations.

For example, current memory bandwidth of a HPC system scales with the number of nodes used. If we assume an achievable memory bandwidth per node of 100GB/s, then it follows that a system with 10 nodes has the potential to provide 1TB/s of memory bandwidth for a distributed application, and a system with 10000 nodes can provide 1PB/s of memory bandwidth. If an application is memory bandwidth bound and can parallelise across nodes then scaling up nodes in this fashion clearly has the potential to improve performance. For B-APM in nodes, and taking Intel®Optane DC persistent memory (DCPM) as an example, if we assume 20GB/s of memory bandwidth per node (5x less than the volatile memory bandwidth), then scaling up to 10 nodes provides 200GB/s of (I/O) memory bandwidth and 10000 nodes provides 200TB/s of(I/O) memory bandwidth. For comparison, the Titan system at ORNL has a Lustre file system with 1.4TB/s of bandwidth [26] and they are aiming for 50TB/s of burst buffer [28] I/O by 2024 [27].

Furthermore, there is the potential to optimise not only the performance of a single application, but the performance of a whole scientific workflow, from data preparation, simulations, data analysis and visualisation. Optimising full workflows by sharing data between different stages or steps in the workflow has the scope to completely remove, or greatly reduce, data movement/storage costs for large parts of the workflow altogether. Leaving data in-situ on B-APM for other parts of the workflow can significantly improve the performance of analysis and visualisation steps at the same time as reducing I/O costs for the application when writing the data out.

Finally, the total runtime of an application can be seen as the sum of its compute time, plus the time spent in I/O. Greatly reduced I/O costs therefore also has the beneficial side effect of allowing applications to perform more I/O within the same total cost of the overall application run. This will enable applications to maintain I/O costs in line with current behaviour whilst being able to process significantly more data. Furthermore, for those applications for which I/O does take up a large portion of the run time, including data analytics applications, B-APM has the potential to significantly reduce runtime.

3.1   Potential caveats

However, utilising internal storage is not without drawbacks. Firstly, the benefit of external storage is that there is a single namespace and location for compute nodes to use for data storage and retrieval. This means that applications can run on any compute nodes and access the same data as it is stored external to the compute nodes. With internal storage, this guarantee is not provided, data written to B-APM is local to specific compute nodes. It is therefore necessary for applications to be able to manage and move data between compute nodes, as well as to external data storage, or for some systemware components to undertake this task, to reduce scheduling restrictions on applications sharing a system with a finite set of compute nodes.

Secondly, B-APM may be expensive to provision in all compute nodes. It may not be practical to add the same amount of B-APM to all compute nodes, mean-
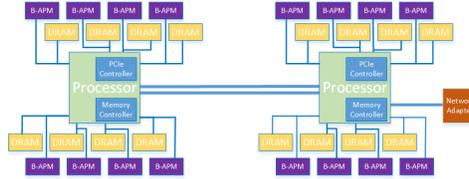
Fig. 7. Compute node hardware architecture

ing systems may be constructed with islands of nodes with B-APM, and islands of nodes without B-APM. Therefore, application or systemware functionality to enable access to remote B-APM and to exploit/manage asymmetric B-APM configurations will be required. Both these issues highlight the requirement for an integrated hardware and software (systemware) architecture to enable efficient and easy use of this new memory technology in large scale computational platforms.

4 Hardware architecture

As Intel®Optane DCPM, and other B-APM when it becomes available, is designed to fit into standard memory form factors and be utilised using the same memory controllers that main memory exploit, the hardware aspect of incorporating B-APM into a compute server or system is not onerous. Standard HPC and HPDA systems comprise a number of compute nodes, connected together with a high performance network, along with login nodes and an external filesystem. Inside a compute node there are generally 2 or more multicore processors, connected to a shared motherboard, with associated volatile main memory provided for each processor. One or more network connections are also required in each node, generally connected to the PCIe bus on the motherboard.

To construct a compute cluster that incorporates B-APM all that is required is a processor and embedded memory controller that support such memory. Customised memory controllers are required to intelligently deal with the variation in performance between B-APM and traditional main memory (i.e. DDR). For instance, as B-APM has a higher access latency than DDR memory it would impact performance if B-APM accesses were blocking, i.e. if the memory controller could not progress DDR accesses whilst an B-APM access was outstanding. However, other than modifying the memory controller to support such variable access latencies, it should be possible to support B-APM in a standard hardware platform, provided that sufficient capacity for memory is provided, as outlined in 7.

Given both DRAM and B-APM are connected through the same memory controller, and memory controllers have a number of memory channels, it is also important to consider the balance of DRAM and B-APM attached to a processor. If we assume a processor has 6 memory channels, to get full DRAM bandwidth we require at least one DRAM DIMM per memory channel. Likewise, if we want full

B-APM bandwidth we need a B-APM DIMM per memory channel. Assuming that a memory channel can support are two DIMM slots, this leads us to a configuration with 6 DRAM DIMMs and 6 B-APM DIMMs per processor, and double that with two processors per node. This configuration is also desirable to enable the DLM configuration, as DLM requires DRAM available to act as a cache for B-APM, meaning at least a DRAM DIMM is required per memory controller.

Pairing DRAM and B-APM DIMMs on memory channels is not required for all systems, and it should be possible to have some memory channels with no B-APM installed, or some memory channels with no DRAM DIMMs installed. However, if DLM mode is required on a system, it is sensible to expect that at least one DRAM DIMM must be installed per memory controller in addition to B-APM. Future system design may consider providing more than two DIMM slots per memory channel to facilitate systems with different memory configurations (i.e. more B-APM than DRAM DIMMs, or memory controllers enabling full B-APM population of memory channels, or only on-chip high bandwith memory with B-APM occupying all the DIMM slots).

Integrating new memory technology in existing memory channels does mean that providing sufficient locations for both main memory and B-APM to be added is important. Depending on the size of B-APM and main memory technology available, sufficient memory slots must be provided per processor to allow a reasonable amount of both memory types to be added to a node. Therefore, we are designing our system around a standard compute node architecture with sufficient memory slot provision to support large amounts of main memory and B-APM as shown in Figure 7.

Another aspect, which we are not focusing on in the hardware architecture, is data security. As B-APM enables data to be retained inside compute nodes, ensuring the security of that data, and ensuring that it cannot be accessed by users or applications that are not authorised to access the data is important. The reason that we are not focusing on this in the hardware architecture is because this requirement can be addressed in software, but it may also be sensible to integrate encryption directly in the memory hardware, memory controller, or processor managing the B-APM.

5   Systemware architecture

Systemware implements the software functionality necessary to enable users to easily and efficiently utilise the system. We have designed a systemware architecture that provides a number of different types of functionality, related to different methods for exploiting B-APM for large scale computational simulation or data analytics.

From the hardware features B-APM provides, our analysis of current HPC and HPDA applications and functionality they utilise, and our investigation of future functionality that may benefit such applications, we have identified

a number of different kinds of functionality that the systemware architecture should support:

1. Enable users to be able to request systemware components to load/store data in B-APM prior to a job starting, or after a job has completed. This can be thought of as similar to current burst buffer technology. This will allow users to be able to exploit B-APM without changing their applications.
2. Enable users to directly exploit B-APM by modifying their applications to implement direct memory access and management. This offers users the ability to access the best performance B-APM can provide, but requires application developers to undertake the task of programming for B-APM themselves, and ensure they are using it in an efficient manner.
3. Provide a filesystem built on the B-APM in compute nodes. This allows users to exploit B-APM for I/O operations without having to fundamentally change how I/O is implemented in their applications. However, it does not enable the benefit of moving away from file based I/O that B-APM can provide.
4. Provide an object, or key value, store that exploits the B-APM to enable users to explore different mechanisms for storing and accessing data from their applications.
5. Enable the sharing of data between applications through B-APM. For example, this may be sharing data between different components of the same computational workflow, or the sharing of a common dataset between a group of users.
6. Ensure data access is restricted to those authorised to access that data and enable deletion or encryption of data to make sure those access restrictions are maintained
7. Provide different memory modes if they are supported by the B-APM hardware.

The systemware architecture we have defined appear to have a large number of components and significant complexity, however the number of systemware components that are specific to a system that contains B-APM is relatively small. The new or modified components we have identified are required to support B-APM in a large scale, multi-user, multi-application, compute platforms are as follows; *Job Scheduler*, *Data Scheduler*, *Object Store*, and *Filesystems*. There are a number of object stores under development, of which some are focussed on efficiently exploiting B-APM hardware, such as DAOS [21] and dataClay [22]. As such we will not focus on object stores in this paper. Likewise, there are a plethora of filesystems that could be deployed on the hardware, both as local filesystems on each node (i.e. ext4) or as distributed filesystems spanning compute nodes (i.e. GekkoFS [33]). We will utilise some filesystems to test performance but not focus on the specifics of filesystems in this paper.

5.1   Job scheduler

As the innovation in our proposed system is the inclusion of B-APM within nodes, one of the key components that must support the new hardware resource

is the job scheduler. Job schedulers, or batch systems, are used to manage, schedule, and run user jobs on the shared resource that are the compute nodes. Standard job schedulers are configured with the number of nodes in a system, the number of cores per node, and possibly the amount of memory or whether there are accelerators (like GPUs) in compute nodes in a system. They then use this information, along with a scheduling algorithm and scheduling policies, to allocate user job request to a set of compute nodes. Users submit job requests specifying the compute resources required (i.e. number of nodes or number of compute cores a job will require) along with a maximum runtime for the job. This information is used by the job scheduler to accurately, efficiently, and fairly assign applications to resources.

Adding B-APM to compute nodes provides another layer of hardware resource that needs to managed by the job scheduler. As data can persist in B-APM, and one of our target use cases is the sharing of data between applications using B-APM, the job scheduler needs to be extended to both be aware of this new hardware resource, and to allow data to be retained in B-APM after an individual job has finished. This functionality is achieved through adding workflow awareness to the job scheduler, providing functionality to allow data to be retained and shared through jobs participating in the workflow, although not indefinitely [24]. The job scheduler also needs to be able to clean up the B-APM after a job has finished, ensuring no data is left behind or B-APM resources consumed, unless specifically as part of a workflow.

Job schedulers already do support assigning resources to jobs, in the form of burst buffer allocations. They also can support workflows, with users able to specify dependencies between jobs submitted or running on a system. However, currently no schedulers support workflow locality, the association of specific nodes with workflow jobs, as is required when sharing data residing in compute nodes. The allocation of burst buffer resources through scheduler functionality also does not provide support for the local nature of data in B-APM, relying on the external nature of burst buffer placements in the storage hierarchy.

Furthermore, as the memory system can have different modes of operation, a supporting job scheduler will need to be able to query the current configuration of the memory hardware, and be able to change configuration modes if required by the next job that will be using a particular set of compute nodes. There are job schedulers that do have support for querying and modify hardware configurations, such as Slurm functionality to support different KNL processor configurations. However, the configuration of B-APM is significantly more complex that KNL MCDRAM, and requires the use of multiple system tools or interfaces to ensure valid memory configurations can be achieved. This requires significant extra on-node scheduler functionality for a job scheduler.

Finally, efficiently allowing users to exploit this new hardware resource will require data aware and energy aware scheduling algorithms. These will utilise the job scheduler's awareness of B-APM functionality and compute job data requirements, and enable scheduling compute tasks to data rather than moving data to compute tasks (as is currently done with external filesystems), or moving

data between compute nodes or external filesystems as required to maximise the utilisation or efficiency of the overall system.

5.2   Data scheduler

The data scheduler is an entirely new component, designed to run on each compute node and provide data movement and shepherding functionality. This include functionality to allow users to move data to and from B-APM asynchronously (i.e. pre-loading data before a job starts, or moving data from B-APM after a job finishes), or between different nodes (i.e. in the case that a job runs on a node without B-APM and requires B-APM functionality, or a job runs and needs to access data left on B-APM in a different node by another job). To provide such support without requiring users to modify their applications we implement functionality in the data scheduler component. This component has interfaces for applications to interact with, and is also for job scheduler component on each compute node. Through these interfaces the data scheduler can be instructed to move data as required by a given application or workflow.

6   Performance Evaluation

To evaluate the performance and usability of our architectures we benchmarked on a prototype HPC system with B-APM installed in the compute nodes. We used a range of different benchmarks, from synthetic workflows, through large scale applications, and I/O benchmarks such as IOR [38].

Test system and setup: All experiments were conducted using a prototype system composed of 34 compute nodes. Each node has two Intel® Xeon® Platinum 8260M CPU running at 2.40 GHz (i.e. 48 physical cores per node), 192 GiB of DDR4 RAM (12x16GB DIMMs) and 3 TBytes of DCPM memory (12x256GB DCPM DIMMs). A single rail Intel® Omni Path network connects the compute nodes through a 100Gbps switch, as well as a 56 Gbps InfiniBand^{TM} network to communicate with a 270TB external Lustre filesystem with 6 OSTs. The compute nodes are running Linux CentOS 7.5 and we use Slurm for job scheduling. To manage and configure the DCPM we use Intel's *ipmctl* and Linux's *ndctl* [37] tools. Version 1.05 of the PMDK toolkit is installed, along with the Intel 19 compiler suite, and Intel's MPI and MKL libraries.

Synthetic workflow: We created a synthetic workflow benchmark that contains two components, a producer and a consumer of data. These components can be configured to produce and consume a number of files of different sizes, but then do no work other than reading or writing and verifying data. We ran this benchmark either targeting the Lustre filesystem or the B-APM in the compute node, and also using the job scheduler integration and data scheduler component to maintain data in B-APM between workflow component execution. Table 1 outlines the performance achieved when producing and consuming 200GB (10 x 20GB files) of data for each configuration. Each benchmark workflow ran 5 times and we report the mean time to complete the benchmark. Performance varied by

Table 1. Synthetic workflow benchmark using Lustre or B-APM in a compute node

Component	Target	Runtime (seconds)
Producer	Lustre	197
Consumer	Lustre	112
Producer	B-APM	133
Consumer	B-APM	60

Table 2. OpenFOAM workflow benchmark using Lustre or B-APM with data staging

Workflow phase	Lustre	B-APM
decomposition	1352	1323
data-staging	–	51
solver	747	95

<15% across runs when using Lustre and <2% when using B-APM. When using B-APM we ran a job that reads and writes 200GB of data between workflow components on the same node to ensure caching does not affect performance. Benchmarks were compiled using the Intel 19 compiler with the -O3 flag.

Benchmarking using Lustre was configured with the producer and consumer processes on two separate compute nodes to ensure that I/O caching locally did not affect measured runtimes. For the benchmark using B-APM we ran using the same node for producer and consumer, communicating data through the B-APM in the node. We can see from the Table 1 using B-APM storage gives ≈45% faster overall runtime (172 vs 309 seconds) for the workflow compared to using Lustre.

Application workflow: OpenFOAM [35] is a C++ library that provides computational fluid dynamics functionality that can easily be extended and modify by users. It is parallelised with MPI and is heavily used in academia and industry for large scale computational simulations. It often requires multiple stages to complete a simulation, from preparing meshes and decomposing them for the required number of parallel processes, to running the solver and processing results. It also, often, undertakes large amounts of I/O, reading in input data and producing data for analysis. It is common that the different stages require differing amounts of compute resources, with some stages only able to utilise one node, and others (such as the solver) requiring a large number of nodes to complete in a reasonable amount of time. OpenFOAM generally creates a directory per process that will be used for the solver calculations, necessitating significant amounts of I/O operations for a large simulation. It is also often useful to save data about the state of the simulation every timestep or every few timesteps. Given these features, OpenFOAM is a good target for both workflow functionality and improved I/O performance through node-local I/O hardware

To evaluate the performance of our architectures using OpenFOAM we ran a low-Reynolds number laminar-turbulent transition modeling simulation of the flow over the surface of an aircraft [34], using a mesh with ≈43 million mesh points. We decomposed the mesh over 20 nodes enabling 960 MPI processes to be used for the solver step (picoFOAM). The decomposition step is serial, takes 1105 seconds, and requires 30GB of memory.

We ran the solver for 20 timesteps, and compared running the full workflow (decomposing the mesh and then running the solver) entirely using the Lus-

Table 3. 10 node IOR performance using B-APM and GekkoFS

Benchmark	Bandwidth (GB/s)
FPP Write	24
SF Write	3
FPP Read	27
SF Read	7

tre filesystem or using node-local B-APM with data staging between the mesh decomposition step and the solver. The solver produces 160GB of output data when run in this configuration, with a directory per process. Running the solver using Lustre required 747 seconds, whereas running the solver using node-local B-APM storage required 95 seconds, more than seven times faster (see Table 2). Using node-local storage needs a redistribution of data from the storage on the single compute node used for decomposing the mesh to the 20 nodes needed for the solver. This data copy took 51 seconds, so even if not overlapped with other running tasks this approach would provide improved performance compared to directly using Lustre, more so when run for a full simulation, which would require many thousands of timesteps meaning the initial cost of copying the data would be negligible.

IOR: Finally, we ran the IOR benchmark on the prototype system using the GekkoFS distributed filesystem. We ran the filesystem on 10 compute nodes, and ran 10 IOR clients per compute node, give a total of 100 IOR processes. We tested both IOR with a single file per process (FPP), and IOR with a shared file (SF) for all processes. For the FFP benchmark each process is writing or reading 8.2GB per file. For the SF benchmark each process is writing 222000 blocks containing 47008 bytes each.

Table 3 presents the performance achieved using 10 nodes using the GekkoFS distributed filesystem exploiting B-APM. We can see that using a single file per process, read and write bandwidth as around 24-27GB/s. The bandwidth achieved using a shared file for all processes is low, at 3GB/s for write and 7GB/s for read (the B-APM is slower for writing than it is for reading). However, these tests are run with a prototype version GekkoFS using only TCP/IP for communication between the nodes, and only the B-APM on a single socket per node meaning communication performance and NUMA effects have reduced the achievable performance.

7   Related work

There are existing technological solutions that are offering similar functionality to B-APM and that can also be exploited for high performance I/O. One example is NVMe devices: SSDs that are attached to the PCIe bus and support the NVM Express interface. Indeed, Intel already has a line of an NVMe device on the market that use B-APM technology, called Intel Optane. Other vendors have a

large range of NVMe devices on the market, most of them based on different variations of Flash technology.

NVMe devices have the potential to provide byte-level storage access, using the *PMDK* libraries. A file can be opened and presented as a memory space for an application, and then can be used directly as memory by that application, removing the overhead of file access (i.e. data access through file reads and writes) when performing I/O and enabling the development of applications that exploit B-APM functionality. However, given that NVMe devices

are connected via the PCIe bus, and have a disk controller on the device through which access is managed, NVMe devices do not provide the same level of performance that B-APM offers. Indeed, as these devices still use block-based data access, fine grained memory operation still require whole blocks of data, rather than individual bytes, to be stored to the device when persistence is required rather than individual bytes, incurring all the overheads associated with such operations.

There are a wide range of parallel and high performance filesystems designed to enable high performance I/O from large scale compute clusters [13] [14] [15]. However, these provide POSIX compliant block based I/O interfaces, which do not offer byte level data access, requiring conversion of data from program data structures to a flat file format. Furthermore, whilst it is advantageous that such filesystems are external resources, and therefore can be accessed from any compute node in a cluster, this means that filesystem performance does not necessarily scale with compute nodes. Such filesystems are specified and provisioned separately from the compute resource in a HPC or HPDA system. Work has been done to optimise I/O performance of such high performance filesystems [16] [17] [18] [19], but they do not address B-APM or new mechanisms for storing or accessing data without the overhead of a POSIX-compliant (or weakly-compliant) filesystem.

The same is true of burst-buffer filesystems, now commonly used to provide improved I/O performance for large scale parallel filesystems. A range of research and implementation [28] [31] [32] work has been undertaken to exploit high performance disk systems, based on SSDs, that can provide optimised I/O operations but with reduced capacity compared to typical full scale parallel filesystems. However, as has already been discussed in section 3, such filesystems are still external to compute nodes, and limited in capacity, bandwidth, and access latency compared to B-APM in compute nodes.

Another technology that is being widely investigated for improving performance and changing I/O for applications is some form of object, or key-value, store [20] [21] [22]. These provide alternatives to file-based data storage, enabling data to be stored in similar formats or structures as those used in the application itself. Object stores can start to approach byte level access granularity, however, they require applications to be significantly re-engineered to exploit such functionality.

We are proposing hardware and systemware architectures in this work that will integrate B-APM into large scale compute clusters, providing significant I/O

performance benefits and introducing new I/O and data storage/manipulation features to applications. Our key goal is to create systems that can both exploit the performance of the hardware and support applications whilst they port to these new I/O or data storage paradigms.

Indeed, we recognise that there is a very large body of existing applications and data analysis workflows that cannot immediately be ported to new storage hardware (for time and resource constraint reasons). Therefore, our aims in this work are to provide a system that enables applications to obtain best performance if porting work is undertaken to exploit B-APM hardware features, but still allow applications to exploit B-APM and significantly improve performance without major software changes.

8   Summary

This paper outlines a hardware and systemware architecture designed to enable the exploitation of B-APM hardware directly by applications, or indirectly by applications using systemware functionality that can exploit B-APM for applications. This dual nature of the system provides support for existing application to exploit this emerging memory new hardware whilst enabling developers to modify applications to best exploit the hardware over time.

The system outlined provides a range of different functionality. Not all functionality will be utilised by all applications, but providing a wide range of functionality, from filesystems to object stores to data schedulers will enable the widest possible use of such systems. We are aiming for hardware and systemware that enables HPC and HPDA applications to co-exist on the same platform.

Whilst the hardware is novel and interesting in its own right, we predict that the biggest benefit in such technology will be realised through changes in application structure and data storage approaches facilitated by the byte-addressable persistent memory that will become routinely available in computing systems.

In time it could possible to completely remove the external filesystem from HPC and HPDA systems, removing hardware complexity and the energy/cost associated with such functionality. There is also the potential for volatile memory to disappear from the memory stack everywhere except on the processor itself, removing further energy costs from compute nodes. However, further work is required to evaluate the impact of the costs of the active systemware environment we have outlined in this paper, and the memory usage patterns of applications.

Moving data asynchronously to support applications can potentially bring big performance benefits but the impact such functionality has on applications running on those compute node needs to be investigated. This is especially important as with distributed filesystems or object stores hosted on node distributed B-APM such in-node asynchronous data movements will be ubiquitous, even with intelligent scheduling algorithms.

Acknowledgements

References

1. Avinash Sodani. *Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor* In Hot Chips 27 Symposium (HCS), 2015 IEEE. IEEE, 124.
2. NVIDIA Volta. `https://www.nvidia.com/en-us/data-center/volta-gpu-architecture`
3. Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, Keith Kim, *HBM (High Bandwidth Memory) DRAM Technology and Architecture*, Memory Workshop (IMW) 2017 IEEE International, pp. 1-4, 2017
4. Andy Turner, Simon McIntosh-Smith, *A survey of application memory usage on a national supercomputer: an analysis of memory requirements on ARCHER* `http://www.archer.ac.uk/documentation/white-papers/memory-use/ARCHER_mem_use.pdf`
5. T. Hady, Frank & Foong, A & Veal, Bryan & Williams, Dan. (2017). *Platform Storage Performance With 3D XPoint Technology.* Proceedings of the IEEE. PP. 1-12. 10.1109/JPROC.2017.2731776.
6. *NVDIMM Messaging and FAQ, SNIA website*, accessed November 2017 `https://www.snia.org/sites/default/files/NVDIMM\%20Messaging\%20and\%20FAQ\%20Jan\%2020143.pdf`
7. *Report on MCDRAM technology from Colfax Research*: `https://colfaxresearch.com/knl-mcdram/`
8. *Intel Patent on multi-level memory configuration for nonvolatile memory technology*; `https://www.google.com/patents/US20150178204`
9. pmem.io: `http://pmem.io/`
10. Layton J. (2010) *IO Pattern Characterization of HPC Applications.* In: Mewhort D.J.K., Cann N.M., Slater G.W., Naughton T.J. (eds) High Performance Computing Systems and Applications. Lecture Notes in Computer Science, vol 5976. Springer, Berlin, Heidelberg
11. Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. 2015. *A Multi-platform Study of I/O Behavior on Petascale Supercomputers*. In Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15). ACM, New York, NY, USA, 33-44. DOI=http://dx.doi.org/10.1145/2749246.2749269
12. *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004) - IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R))*
13. Schwan, Philip. *Lustre: Building a file system for 1000-node clusters.* In Proceedings of the 2003 Linux Symposium, vol. 2003. 2003

---

[3] www.nextgenio.eu

14. Frank Schmuck and Roger Haskin. 2002. *GPFS: A Shared-Disk File System for Large Computing Clusters*. In Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02). USENIX Association, Berkeley, CA, USA, Article 19.

15. *Introduction   to   BeeGFS*:   `http://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf`

16. S. Jian, L. Zhan-huai and Z. Xiao, *The performance optimization of Lustre file system* 2012 7th International Conference on Computer Science & Education (ICCSE), Melbourne, VIC, 2012, pp. 214-217. doi: 10.1109/ICCSE.2012.6295060

17. Wonil Choi, Myoungsoo Jung, Mahmut Kandemir, Chita Das, *A Scale-Out Enterprise Storage Architecture*, IEEE International Conference on Computer Design (ICCD), 2017, 10.1109/ICCD.2017.96

18. Kuan-Wu Lin, Surendra Byna, Jerry Chou, and Kesheng Wu. 2013. *Optimizing fastquery performance on lustre file system*. In Proceedings of the 25th International Conference on Scientific and Statistical Database Management (SSDBM), Alex Szalay, Tamas Budavari, Magdalena Balazinska, Alexandra Meliou, and Ahmet Sacan (Eds.). ACM, New York, NY, USA, , Article 29 , 12 pages. DOI=http://dx.doi.org/10.1145/2484838.2484853

19. Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. In Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST '11). IEEE Computer Society, Washington, DC, USA, 1-14. DOI=http://dx.doi.org/10.1109/MSST.2011.5937212

20. Jungwon Kim, Seyong Lee, Jeffrey S. Vetter: *PapyrusKV: a high-performance parallel key-value store for distributed NVM architectures*. SC 2017: 57:1-57:14

21. J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent and E. Barton, *DAOS and Friends: A Proposal for an Exascale Storage System*, SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, 2016, pp. 585-596. doi: 10.1109/SC.2016.49

22. Jonathan Mart, Anna Queralt, Daniel Gasull, Alex Barcel, Juan Jos Costa, Toni Cortes, *Dataclay: A distributed data store for effective inter-player data sharing*, In Journal of Systems and Software, Volume 131, 2017, Pages 129-145, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2017.05.080

23. Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M Badia, Jordi Torres, Toni Cortes, Jess Labarta, *PyCOMPSs: Parallel computational workflows in Python*, The International Journal of High Performance Computing Applications, Vol 31, Issue 1, pp. 66   82 First Published August 19, 201, https://doi.org/10.1177/1094342015594678

24. E. Farsarakis, I. Panourgias, A. Jackson, J.F.R. Herrera, M. Weiland, M. Parsons; *Resource Requirement Specification for Novel Data-aware and Workflow-enabled HPC Job Schedulers*, PDSW-DISCS17 `http://www.pdsw.org/pdsw-discs17/wips/farsarakis-wip-pdsw-discs17.pdf`, 2017

25. Weiland, M, Jackson, A, Johnson, N & Parsons, M 2018, *Exploiting the Performance Benefits of Storage Class Memory for HPC and HPDA Workflows* Supercomputing Frontiers and Innovations, vol 5, no. 1, pp. 79-94. DOI: 10.14529/jsfi180105

26. ORNL Titan specification `http://phys.org/pdf285408062.pdf`

27. Valentine Anantharaj, Fernanda Foertter, Wayne Joubert ,Jack Wells *Approaching Exascale: Application Requirements for OLCF Leadership Computing* `https://www.olcf.ornl.gov/wp-content/uploads/2013/01/OLCF_Requirements_TM_2013_Final1.pdf` July 2013

28. Daley, C., Ghoshal, D., Lockwood, G., Dosanjh, S., Ramakrishnan, L., Wright, N.: Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers. Future Generation Computer Systems (2017), 10.1016/j.future.2017.12.022
29. N. R. Mielke, R. E. Frickey, I. Kalastirsky, M. Quan, D. Ustinov and V. J. Vasudevan, *Reliability of Solid-State Drives Based on NAND Flash Memory* in Proceedings of the IEEE, vol. 105, no. 9, pp. 1725-1750, Sept. 2017. doi: 10.1109/JPROC.2017.2725738
30. Chuanpeng Li, Chen Ding, and Kai Shen. 2007. *Quantifying the cost of context switch.* In Proceedings of the 2007 workshop on Experimental computer science (ExpCS '07). ACM, New York, NY, USA, Article 2 . DOI: https://doi.org/10.1145/1281700.1281702
31. N. Liu et al., *On the role of burst buffers in leadership-class storage systems*, 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, 2012, pp. 1-11. doi: 10.1109/MSST.2012.6232369
32. Torben Kling Petersen, John Bent, *Hybrid flash arrays for HPC storage systems: An alternative to burst buffers*, High Performance Extreme Computing Conference (HPEC) 2017 IEEE, pp. 1-7, 2017.
33. Marc-André Vef and Nafiseh Moti and Tim Süß and Tomasso Tocci and Ramon Nou and Alberto Miranda and Toni Cortes and André Brinkmann, *GekkoFS - A temporary distributed file system for HPC applications*, Proceedings of the 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast, UK, September 10-13, 2018,
34. Andrejašič, Matej and Veble, Gregor and Bat, Nejc, *Cloud-based Simulation of Aerodynamics of Light Aircraft*, `https://hpc-forge.cineca.it/files/CoursesDev/public/2015/Workshop_HPC_Methods_for_Engineering/cloud_based_aircraft.pdf`
35. Hrvoje Jasak, *OpenFOAM: Open source CFD in research and industry*, International Journal of Naval Architecture and Ocean Engineering, vol=1, no=2, pages=89-94, 2009, issn=2092-6782
36. IPMCTL, `https://github.com/intel/ipmctl`
37. NDCTL - Utility library for managing the libnvdimm (non-volatile memory device) sub-system in the Linux kernel, `https://github.com/pmem/ndctl`
38. IOR `https://github.com/LLNL/ior`