Rebasing I/O for Scientific Computing: Leveraging Storage Class Memory in an IBM BlueGene/Q Supercomputer

Felix Schürmann[1], Fabien Delalondre[1], Pramod S. Kumbhar[1], John Biddiscombe[2], Miguel Gila[2], Davide Tacchella[2], Alessandro Curioni[3], Bernard Metzler[3], Peter Morjan[4], Joachim Fenkes[4], Michele M. Franceschini[5], Robert S. Germain[5], Lars Schneidenbach[5], T.J. Christopher Ward[6], and Blake G. Fitch[5]

[1] Blue Brain Project, Brain Mind Institute, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{felix.schuermann,fabien.delalondre,pramod.kumbhar}@epfl.ch
[2] CSCS, Swiss National Supercomputing Centre, Lugano, Switzerland
{biddisco,miguel.gila,tack}@cscs.ch
[3] IBM Research GmbH, 8803 Rueschlikon, Switzerland
{cur,bmt}@zurich.ibm.com
[4] IBM Deutschland Research & Development GmbH, 71032 Böblingen, Germany
{peter.morjan,fenkes}@de.ibm.com
[5] IBM T.J. Watson Research Center Yorktown Heights, NY 10598, USA
{franceschini,rgermain,schneidenbach,bgf}@us.ibm.com
[6] IBM Software Group, Hursley Park, Hursley SO212JN, U.K.
tjcw@uk.ibm.com

Abstract. Storage class memory is receiving increasing attention for use in HPC systems for the acceleration of intensive IO operations. We report a particular instance using SLC FLASH memory integrated with an IBM BlueGene/Q supercomputer at scale (Blue Gene Active Storage, BGAS). We describe two principle modes of operation of the non-volatile memory: 1) block device; 2) direct storage access (DSA). The block device layer, built on the DSA layer, provides compatibility with IO layers common to existing HPC IO systems (POSIX, MPIO, HDF5) and is expected to provide high performance in bandwidth critical use cases. The novel DSA strategy enables a low-overhead, byte addressable, asynchronous, kernel by-pass access method for very high user space IOPs in multithreaded application environments. Here, we expose DSA through HDF5 using a custom file driver. Benchmark results for the different modes are presented and scale-out to full system size showcases the capabilities of this technology.

Keywords: data-intensive supercomputing, IO, storage class memory, IOPS, verbs.

J.M. Kunkel, T. Ludwig, and H.W. Meuer (Eds.): ISC 2014, LNCS 8488, pp. 331–347, 2014.

1 Introduction

Despite the ever present ranking of supercomputers by their computational performance, there is a growing awareness of the need to include other system characteristics relevant for many scientific applications such as overall memory footprint, memory bandwidth and latency as well as access to storage[1–3]. This process is fueled by several developments: recent years of technology development yielded a scaling of logic circuits that was not matched by equivalent scaling of either capacity or bandwidth of the primary memory technology, DRAM. For scientific applications that do not share the property of high arithmetic intensity of standard benchmarks [4] this results in a diminishing fraction of utilized peak performance of those systems. Secondly, novel memory technologies are emerging, which exhibit properties distinctly different from classical DRAM or hard disk drives and present alternative design points in terms of density and energy efficiency as well as latency and bandwidth [5], hinting at opportunities to overhaul classical memory and IO system hierarchies. The term Storage-Class-Memory (SCM) has been coined to refer to these technologies which are typically non-volatile. Lastly, novel applications from the field of data analytics or data-intensive modeling are entering the HPC realm and their memory and IO requirements are driving interest in more cost-effective design points than conventional DRAM-only supercomputers fulfilling those requirements can offer.

Detailed bio-physical modeling of brain tissue, such as that pioneered by the Blue Brain Project [6–8], is an example of a complex scientific application hoping to exploit the properties of storage-class-memory in supercomputers in various scenarios and is the driver for this present study. One of the features of detailed brain modeling is the intrinsic heterogeneity of the brain: each neuron and each synapse is distinct, which in a computational representation leads to parameters and state variables unique to every modeled entity [9]. Combined with the large numbers of those entities, e.g. an estimated 200 million neurons and 10^{12} synapses for an entire rat brain [10], the resulting memory footprint is large and at the same time the algorithmic intensity low. With the human brain being an additional three orders of magnitude more complex, cellular models of the human brain will occupy a daunting estimated 100PB of memory that will need to be revisited by the solver at every time step.

Apart from the absolute scale and bandwidth challenges of detailed brain tissue models, advocating a paradigm of data-centric computing, there are numerous other, more mundane, aspects of today's computational workflows that make the exploration of SCM interesting and progressively tractable: on the one hand, multiple steps are involved for building the computational representation of the brain tissue model [11], solving the time evolution of the equations by the simulator [12], and validating the structural and functional aspects [8]. On the other, analysis of those brain tissue models often requires transformation of the data representation. A classical example is that a visualization may require a portion of a model selected by view-frustum and spatial coordinates while the simulator utilizes an object-based decomposition [13]. Acceleration of large-scale, block-oriented access

as well as facilitation of concurrent small random reads limited by the number of IOPS will be beneficial for these workflows.

2 Related Work

Substantial work has been published on characterizing IO patterns of scientific applications and benchmarking classical disk-based file systems, see e.g. [14–19]. The complexity of the storage hierarchies and the number of parameters that can affect performance led to the development of high-level IO libraries such as ADIOS [20] or SIONlib [21] but tuning for performance is an ongoing challenge[22]. For the last several years the usage of flash in HPC applications has been proposed and investigated, mostly in form of using commodity SSDs and PCIe cards with flash[23–26, 1, 3].

A more systems-level view has been proposed in the Active Storage work [27, 28] that now includes development of the hardware/software stack from storage containers to middleware and also envisions the inclusion of storage-class-memory technologies beyond flash. Other system implementations comprising clusters of power-efficient nodes with flash storage have been explored at CMU [29, 30]. Another related activity is the RAMCloud work that is exploring the aggregation of large amounts of DRAM on a network to support data intensive applications that require very low latency access [31]. Considerations of where to integrate SCM continue [32]. The distinguishing feature of the Active Storage work is the combination of activities that include: 1) focusing on making storage-class-memories available as an aggregated global resource in a scale-out system 2) exploring the potential for parallel computational offload into the Active Storage layer (using general purpose CPUs and embedded processing elements) 3) research efforts at many levels in the hardware/software stack, particularly the exploration of novel methods for enabling access to storage-class-memories.

3 Research HPC System

A Blue Gene Active Storage (BGAS) system is defined by extending the standard IBM BlueGene/Q (BGQ) architecture [33] by the integration of storage-class-memory and an external network interface into each node. BGAS systems can be used in a stand-alone mode or as the I/O subsystem of a standard BlueGene/Q compute fabric. An instance of this, Blue Brain IV, has been built and installed by the Swiss National Computing Center (CSCS) in Lugano for and on behalf of the Blue Brain Project at EPFL in Lausanne.

This system is based on 4 standard racks of BGQ integrated with two types of components, which can be switched via software per partition of the BGQ: 1) The Blue Brain IV *standard configuration* connects the BGQ compute racks to standard IBM BlueGene/Q I/O Nodes via a proprietary network [34], which in turn are connected with 5 IBM GSS running GPFS [35] via QDR Infiniband. The nominal specifications of the system are: 4096 nodes of A2 compute chip [36] for a total of 0.8 Petaflops theoretical peak, 16 GB DRAM/node for a total

of 64 TB DRAM memory, standard IBM BlueGene/Q Compute Node Kernel. The attached storage is a 5 IBM GSS model 26 with 4.2 PB usable raw storage running GPFS v3.5. 2) In the *active storage configuration*, the BGQ compute racks are connected to a separate set of IONs, each augmented with a prototype PCIe card. This Hybrid Scalable Solid State Storage (HS4) card is a PCIe 2.0 x8 full height half length device that integrates 1.9 TB of SLC flash, DRAM and two 10 GbE ports. Via the 10GbE ports, these IONs are also connected to the IBM GSS system. The BG/Q IO drawer is a standard rack mount 3U, air cooled, chassis containing eight Blue Gene/Q nodes on a planar which enables a $2 \times 2 \times 2$ 3D torus among them. The IO torus is a research feature and enables Blue Gene/Q torus connectivity among large numbers of I/O nodes by extending the torus with optical cables among IO drawers. The Blue Brain IV system has a 64 node BGAS system cabled as a $4 \times 4 \times 4$ node 3D torus with a total of about 120TB SLC flash.

The BGAS software stack extends the RHEL 6 based Blue Gene/Q IO node software environment [37] providing a general purpose multi-tenancy operating system capable of hosting both storage and application functions. The BGAS extensions are integrated into a single image, which is network loaded on all BGAS nodes by the Blue Gene/Q control system during partition boot.

4 Direct Storage-Class-Memory Access (DSA)

The integration of SCM presents significant challenges to the classic I/O stack. Like flash, emerging memory technologies can have limited endurance requiring wear-leveling and other management. They may read and write with asymmetric latency and bandwidth characteristics and may require explicit erase operations. In general, new memories seem to be trending toward more complex characteristics which will be best utilized in hybrid devices where the memories serve complementary roles. Additionally, the processors accessing these hybrid memory devices are likely to have an increasingly large core count.

In order to address those challenges, as well as the specific challenge of flash memory latencies, a storage interface is required that includes enabling many deep queues of asynchronous work requests. To deliver the full capability of the SCM to applications, such an interface should be directly accessible from user space, by-passing the significant overheads of the classic block I/O stack. Finally, in order to avoid another I/O interface change when truly byte addressable SCM becomes cost-effective, we set that requirement for the current interface. To this end, a novel SCM interface based on the Open Fabrics Enterprise Distribution (OFED$^{\text{TM}}$) [38] Remote Direct Memory Access (RDMA) interface has been defined: the "Direct SCM Access" (DSA) method. DSA enables the RDMA-style memory registration of SCM regions as well as user application virtual memory regions and enables zero-copy transfers among them. The registration of storage is analogous to the registration of virtual memory for OFED RDMA.

DSA is the primary interface for the BGAS HS4 device. The DSA driver is implemented as an OFED RDMA verbs provider and plugs into the standard

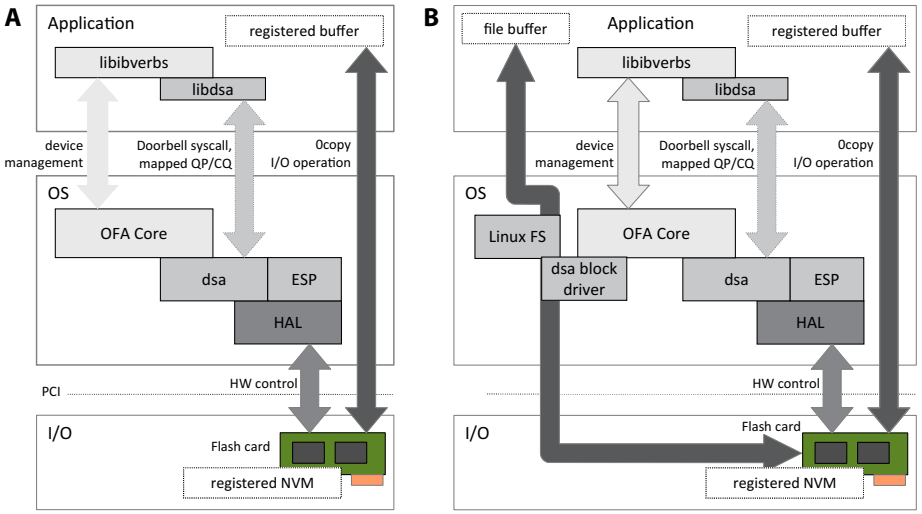

Fig. 1. A) Direct SCM Access (DSA) module implemented as a regular RDMA verbs provider that plugs into the OFED framework. B) Adding support for block based NVM access to DSA.

OFED RDMA framework. The DSA driver includes a module called an "Embedded Storage Peer" (ESP) and this is the only process which can be connected to by the DSA driver. Once a DSA user process connects to the ESP, OFED RDMA send/receive type messages are used to open and close storage memory partitions in a request/response manner. DSA uses the two sided RDMA communication model (Send and Receive) for control operations on the HS4 device, and the one-sided RDMA Reads and Writes to read and write flash content.

Figure 1A depicts a high level design overview of the components and their interactions with the OpenFabrics core as well as with the HS4 hardware. As with any OpenFabrics RDMA provider, DSA comprises two main software modules: 1) A loadable kernel module which includes the ESP, attaches to the OFA core, and further interacts with the hardware abstraction layer (HAL). DSA establishes a fast data path for flash read and write operations from user level. This fast data path is implemented as application private work and completion queues represented by memory regions shared between DSA and the user level library. 2) A dynamically linkable user library, libDSA, which links with the generic libibverbs OFA library to provide both DSA specific RDMA control path and fast path. Interacting with the DSA kernel module via the generic OFA control path, libDSA manages communication contexts and queue pair/completion queues.

Figure 1B shows the integration of HS4 memory resources into the Linux file system. The DSA block driver module bridges between legacy block-based memory access and the byte addressable nature of the DSA API. The DSA kernel module exports a kernel level RDMA verbs interface accessed by the DSA block driver via the OpenFabrics kernel RDMA verbs interface. The DSA block driver

in turn provides a generic block driver interface used by the Linux file system services. This also offers opportunities to exploit the features of HS4 card and build journaling file systems that make use of the other available SCM types.

4.1 Exposing DSA Using HDF5

In order to expose DSA level access directly to applications, we have extended a previously developed Virtual File Driver (VFD) for the HDF5 Parallel IO library. A full description of the driver with examples of usage is provided in [39, 40]. The driver maps a region of Distributed Shared Memory (DSM) hosted on nodes within a parallel job and makes them visible to the HDF5 library as a byte addressable file space hosted in RAM instead of disk. This allows for high speed access to/from a file in memory which can be shared by coupled applications allowing them to exchange information using the HDF5 API. A natural extension of this is to map the file addresses to flash memory distributed over the HS4 cards and thereby create a shared file with persistence so that not only can data be shared between coupled applications, but the persistent nature of the memory allows it to be reused between sessions.

The DSM driver exposes RAM directly to the MPI layer to enable one sided transfers to take place to/from memory on a hosting node to the application. Since the DSA presents a view of flash memory that is behind an API, it is no longer possible to perform one sided put/get operations directly to the user application and it is necessary to add an additional virtualization layer between DSM address and the storage. Data reads/writes from flash are therefore locally buffered on each node and then passed to MPI for exchange to remote nodes. This has the disadvantage of reducing performance relative to the RAM based approach (in addition to the higher latency), but does allow for arbitrary sized HDF5 files to be created as one is no longer limited by the available/addressable RAM on each DSM node and may create a virtual space as large as the combined sizes of the flash partitions available.

5 Benchmarks

dsa_bench is a micro-benchmark created to test and measure the DSA stack performance from user space. It is configurable via command line parameters and can access all partitions and types of memory on the HS4 hardware. The benchmark posts the specified number of requests until the queue limit is reached. Then it polls for a completion before posting the next request. The command line arguments also allow controlling the alignment of addresses in host memory and in HS4 storage. This is especially useful for testing memories other than flash or walking alternative code paths for flash access because flash can only be written in 8 KiB increments. IOPS are measured using settings with deep queues and fewer signalled requests with small transfer sizes (e. g. 8 KiB, with 512 outstanding requests and a signal every 32 requests). Bandwidth measurements are done with at least 64 1 MiB requests.

By default, the benchmark performs 2 phases, an initial sequential write over a given range of memory on the HS4 card and a second phase, by default comprising random reads, over the same memory range. A separate transfer data size can be specified for each phase. The second phase can be configured to use random or sequential access for pure reads and writes or a 50:50 mix of reads and writes. Data verification and different data patterns are possible too. The user controls the depth of the queues and the number of unsignalled requests between signalled requests. Unsignalled requests complete without completion queue entries and are completed with the next signalled request. Therefore, unsignalled requests are posted in batches together with a signalled request as the last in a batch.

The Linux utility **dd** is being used to measure bandwidth for sequential reads/writes from/to flash configured as block device.

IOR is a highly configurable MPI application designed to measure parallel file system read/write I/O performance using various interfaces and access patterns. The version of the code used for this work is available on GitHub [41] master branch with aa604c1d38de803aa0db3f3abb5515e0ed1857ea SHA1. IOR executables have been compiled with both POSIX and MPI-I/O interfaces using MPICH2 (V1.5) [42] and XLC (V12.0) or gcc (V4.4.7) compilers on IBM Blue Gene/Q or BGAS cluster respectively. GPFS file systems created on the flash of the HS4 cards for this study use 16 MiB block and 4 GiB page pool size (per ION) for bandwidth measurements and 128 KiB block and 1 GiB page pool size (per ION) for IOPS measurements; for bandwidth measurements, IOR BlockSize (`$IORBlocksize`) was set to 160 GiB on BGAS and 10 GiB on BGQ CNK and respectively to 32 GiB on BGAS and BGQ CNK for IOPS measurements. Accordingly, the page pool sizes are always well below 1 % of the transferred data to avoid caching effects. Execution of IOR benchmark directly on BGAS or from Blue Gene/Q compute nodes to BGAS nodes allowed measuring bandwidth and IOPS performance. IOPS using IOR are computed from the bandwidth for 4 KiB IOR data transfer size, where the file opening time is negligible. Depending on the benchmark the number of MPI processes/node (`$MPIproc`) was varied. IOR was invoked with the following parameters: `ior -b $IOR_BlockSize -r10 -p $MPIproc -t $TransferBlockSize -a POSIX/MPIIO -u -I`.

HDF5 bench is a simple program using the HDF5 API to open a file collectively across a series of nodes and writes a 1D hyperslab of data per process into the file, such that the hyperslabs cover the entire file space. The flash-DSM size is set to a multiple of the number of processes used with each process writing a (configurable) fixed size chosen in this case to be 256 MiB each, with an average timing from 15 runs being taken. To smooth out network contention, the DSM is set to use Block Cyclic memory mapping, with a block size of 4 MiB, implying that each 4 MiB chunk of data maps to a consecutive MPI rank of the DSM. This implies that each DSA put operation (IOP) is also 4 MiB in this example.

6 Early Benchmark Results and Discussions

6.1 Micro-benchmarks

The BGAS stack can be measured at several standard and novel interfaces to evaluate the cost/benefit of each layer. In Table 1a, we present a snapshot of BGAS stack micro-benchmark results for flash measured at the DSA layer and at the verbs block device layer. Table 1b contains small block size IOPS results for on-card DRAM measured at the DSA layer to provide some indication of the performance possible with memory technologies other than flash. Benchmark results on the IBM Power7+ system serve as a baseline reference for the capability of the HS4 device.

Table 1. Single card read (R) and write (W) micro-benchmarks for both Blue Gene/Q and Power 7+ platforms. All read benchmarks used a random access pattern while all write benchmarks used sequential access. We observed relatively small performance differences due to random versus sequential access. The bandwidth data were acquired using 1 MB transfer block sizes while the IOPS results are reported for a selection of small block sizes. Flash access performance is reported for single-threaded tests as well as the best performance achieved (Number of threads in parentheses or "==" if single-threaded performance was the best). Results for 32 Byte random IOPS achieved to DRAM on the HS4 card are provided to showcase the low overheads possible using the DSA software stack.

Flash Performance:			BG/Q		P7+	
			Single Thr.	Best Conf.	Single Thr.	Best Conf.
DSA client	BW (MB/s)	W(1 MB)	1100	==	2710	==
		R(1 MB)	1240	2270 (2)	3020	==
	IOPS	W(8 KB)	65k	91k (4)	270k	==
		R(8 KB)	75k	200k (3)	360k	==
		R(4 KB)	85k	180k (3)	600k	700k (2)
	Lat. (μs)	W(8 KB)	490	==	440	==
		R(4 KB)	133	==	78	==
VBD (dd)	BW (MB/s)	W(1 MB)	860	1000 (2)	1300	2200 (2)
		R(1 MB)	2100	2200 (2)	3000	==

(a) Results for accessing flash storage via DSA directly or via the verbs block device (VBD).

32 Byte DRAM Access

	BG/Q	P7+	Thr.
W	120k	710k	1
W	230k		2
W	340k		3
R	120k	740k	1
R	235k	1060k	2
R	340k	1050k	3

(b) IOPS performance results for 32 Byte accesses to DRAM storage on the HS4 card.

The DSA bandwidth results show both Blue Gene/Q and P7+ approaching the PCIe 2.0 x8 limit (after encoding overheads) of 3.2 GB/s for flash reads. We note that two processes are required on Blue Gene/Q to achieve best performance on flash reads while P7+ requires just a single thread. On P7+ we achieve high write bandwidth using a single thread however, on Blue Gene/Q, the performance results indicate that more processes will be required to achieve

full bandwidth. Similarly, DSA achieves our target of 200k IOPs using 3 processes on Blue Gene/Q while P7+ sets a baseline of 600k IOPs on one thread. We have implemented the Verbs Block Device (VBD) to enable a standard Linux Block Device interface using DSA to enable files systems such as Ext4 and GPFS to utilize storage-class-memory. The overhead associated with layering a block device is shown in Table 1a.

In Table 1b we report on DSA performance while accessing DRAM. We make all the memories on the HS4 card available via DSA and benchmarking DRAM enables us to explore the performance limits of our stack independent of flash attributes. Read and write IOPs scale well with increased threads on Blue Gene/Q while Power7+ requires fewer processors for similar performance. Micro-benchmarking DSA accesses to DRAM gives a snapshot of the current limits of the software stack on a given platform. We exceed our target of 200k IOPs on Blue Gene/Q with three processes and achieve over one million IOPs on Power7+. These results show that the BGAS stack built around DSA will exceed our performance targets on power efficient, many core architectures.

6.2 Block Device Interface

The block device interface represents the easiest path for an application to adopt the BGAS capabilities since no modifications are necessary if standard IO libraries such as POSIX and MPIO are used. In order to map out what an application may expect from BGAS in this mode, we measured the bandwidth (IOPS) as function on transfer size for a GPFS with settings as described in Section 5 for the two different scenarios. Figure 2 shows the respective GPFS performance of a single BGAS node. Single node performance is the average performance per node on an 8 ION GPFS partition where `$MPIproc` = 1. In line with the micro-benchmarks, read performance is about twice the write performance for large transfers in terms of bandwidth and about 50% larger in terms of small IOPS. The different IO interfaces (POSIX file per process/shared file; MPIO file per process/shared file) at this scale are fairly equivalent.

As also indicated by the micro-benchmarks, bandwidth performance for block device operations on the HS4 card shows some but not a very large dependency on the number of threads used of the A2 processor. If more MPI processes per node are used for the IOR benchmark, maximal read and write performance is reached at smaller transfer sizes (Figure 3; left panel) and maximal read performance is slightly improved. It should be noted that the overhead introduced by GPFS for larger reads and writes is about 20-30% when compared to the lower level dd measurements reported in Table 1a, however, more tuning for the A2 processor might diminish that further. These measurements indicate that GPFS on BGAS is well suited for applications requiring bandwidth even for transfers smaller than the GPFS block size.

When looking at IOPS on a block device, adding concurrency is very important as can be seen by a more than 2.5X increase in IOPS for small reads and writes when going from `$MPIproc` = 1 to 4 (Figure 3; right panel). However, even these numbers do not max out the HS4 capabilities compared to when

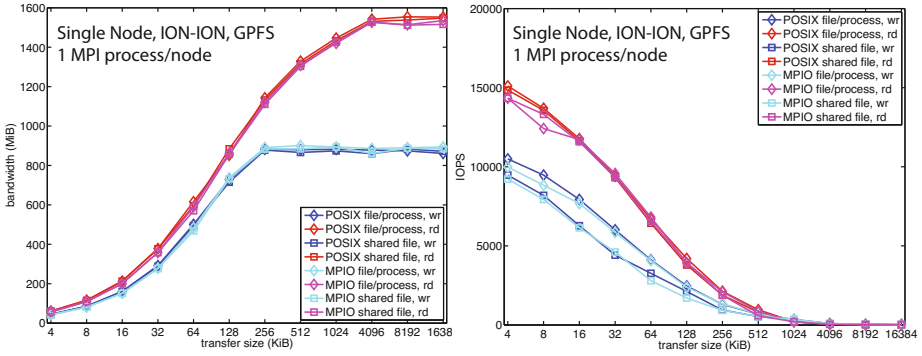

Fig. 2. Single node performance of an IOR benchmark running on BGAS ION to GPFS on flash. GPFS and IOR parameters as described in Section 5.

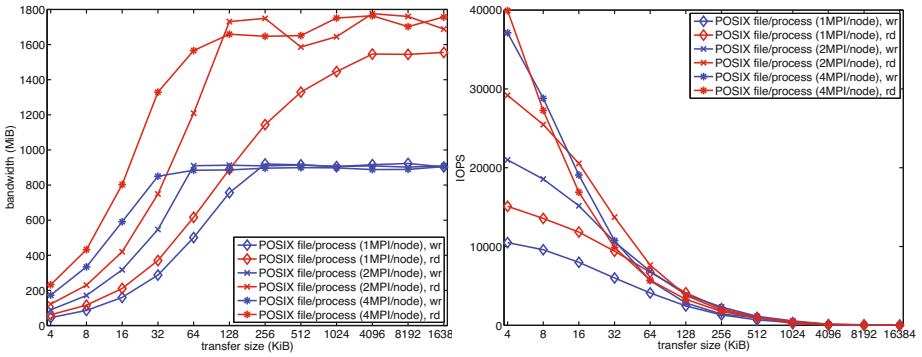

Fig. 3. Single node performance on BGAS ION as function of the number of MPI processes per node used for IOR bandwidth and IOPS measurements

using DSA directly as shown in Table 1a, which is the reason why exploring alternative access methods to block devices is desirable especially for applications bound by many small random transfers.

To demonstrate the building block approach of the BGAS technology, four different GPFS partitions were created spanning 8, 16, 32 and 64 BGAS IONs respectively. These partitions can be used stand-alone on BGAS (ION-ION) or from BGQ in optimal partitions of 512, 1024, 2048, and 4096 CNK nodes respectively (CNK-ION). Scale out performance of the BGAS technology is shown in the left panel of Figure 4. Taking the aggregate performance of the 8 node ION-ION measurement as reference, we observe essentially ideal scaling to 16, 32, and 64 IONs. This holds true for POSIX shared file access (not shown). The same absolute performance and ideal scaling is also observed when accessing BGAS from 512, 1024, 2048, and 4096 BGQ compute nodes (CNK-ION). This

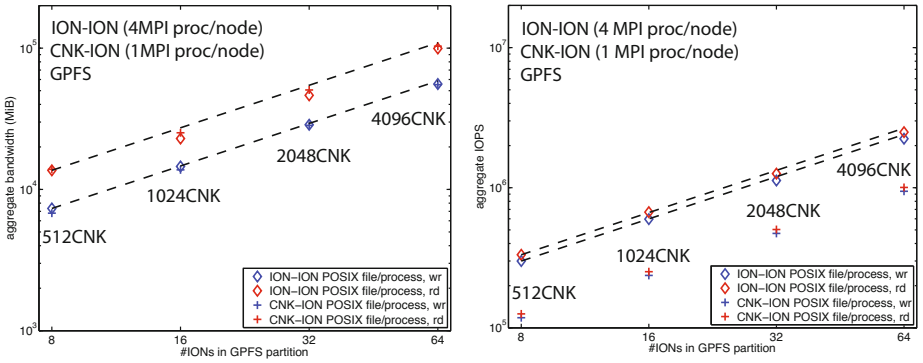

Fig. 4. Aggregate performance of BGAS as function of number of IONs partaking in the GPFS partition; Left: IOR bandwidth measurements from IONs and CNK; Right: IOR IOPS measurements from IONs and CNK

results into an aggregate bandwidth of 99,457 MiB/s read/55,748 MiB/s write for the full BGAS system.

Similarly for IOPS (see right panel of Figure 4), ideal scaling up to full system size can be observed for the stand-alone BGAS (ION-ION) scenario reaching 2,494,310 read/2,237,754 write IOPS (4 KiB) when using 64 IONs. When driving the IOPS measurement from the BGQ CNKs ideal scaling is still observed but the absolute performance only shows about 40 % of the stand-alone BGAS scenario. If one reduces the overall transferred amount of data (implying that more requests will be found in the page pool), this performance drop can be recovered fully. We therefore conclude that the GPFS implementation on BGAS is in principle capable of coping with the increased concurrency in the CNK-ION scenario but more analysis is needed to understand the ideal thread settings and pinning and optimal GPFS parameters.

Bandwidth performance was measured with the GPFS and IOR parameters described in Section 5 and `$TransferBlockSize` = 4 MiB; `$MPIproc` = 4 for the ION-ION scenario and `$TransferBlockSize` = 16 MiB; `$MPIproc` = 1 for the CNK-ION scenario. Analogously, aggregate IOPS were measured using `$TransferBlockSize` = 4 KiB and `$MPIproc` = 4 in the ION-ION scenario and `$MPIproc` = 1 in the CNK-ION scenario.

6.3 HDF5 on DSA

Measurements of bandwidth writing to a single shared file in parallel using the HDF5 driver give values of 1074, 2120, 4349, 7253 MiB/s for 1,2,4,8 nodes respectively. This shows good scaling for the node counts tested and reaches 2K IOPS for 4MB block transfers. The implementation makes use of a single DSM server per node which sends DSA put/get requests to the flash driver and MPI Send/Receive requests to the other IONs to coordinate the IO. The interface

between the two-sided MPI and one-sided DSA is the limiting factor in the performance in this implementation and can be simplified and improved by using an OFED verbs layer for both channels of communication with asynchronous transfers on both sides. Using multiple servers per node should increase the throughput by ensuring the DSA driver read/write queues are always kept full. The implementation of HDF5 directly on top of DSA represents the first step towards exposing the low level hardware access directly to applications in user-space, which in turn opens up new possibilities for the exploitation of this type of memory layer.

7 Conclusions and Outlook

We presented a particular instance of storage-class-memory, namely SLC flash, integrated at scale with an IBM BlueGene/Q supercomputer: BlueGene Active Storage. The motivation for building this system stemmed from the use-case of data-centric computing, particularly prominent in detailed brain tissue modeling and simulation as pursued by the Blue Brain Project. A prototype system has been developed and installed, and first benchmarks presented.

Particular effort has been put into exposing the properties of SCM to the application by developing a novel direct storage-class-memory access (DSA) method, based on the OFED RDMA interface. We demonstrated that it can serve as the foundation on which to build block devices to support file systems, namely GPFS, which serves as an immediate transition path for applications using legacy IO layers; we demonstrated this by achieving 99,457 MiB/s read and 55,748 MiB/s write bandwidth using 64 BGAS IONs; these results show ideal scaling to full system size. We furthermore demonstrate that the bandwidth performance is fully available to the attached BlueGene/Q compute nodes, also showing ideal scaling from 512 to 4096 nodes and identical absolute performance. We furthermore tested the systems capabilities for small (4 KiB) IOPS using GPFS, showing ideal scaling from 8 to 64 IONs and reaching 2,494,310 read and 2,237,754 write IOPS when running on 64 IONs of BGAS. Scaling holds true when driving the small IOPS from 512 to 4096 BlueGene/Q compute nodes but overall IOPS are reduced when compared to the BGAS only case.

We conclude that GPFS provides an excellent basis for bandwidth oriented access patterns and also provides a good foundation for large numbers of small IOPS with some more configuration work to do. At the same time, a user-space micro-benchmark shows that if DSA is used directly without a block device, it has the potential to boost IOPS even further. We present prototype work to expose this to applications using HDF5 and a custom driver to map it directly on top of DSA. We show first results for 8 IONs and we expect to scale this work to the entire system in the future.

Even though this prototype system is still in its early evaluation phase, the preliminary benchmark results convincingly demonstrate its potential. Further work will be undertaken to tune the software stack to get even closer to the capabilities of the hardware as indicated by the micro-benchmarks. Progressively,

more applications and workflows will be adapted to directly use the DSA and to illustrate the role of IOPS optimized storage in HPC for a real-world application use case going beyond checkpoint/restart. How to provision this versatile storage-class memory to applications will be part of the future research: one can imagine provisioning tailored storage on a per application and job basis or using BGAS's capabilities to have multiple partitions (each tailored for a particular access pattern) running concurrently.

We acknowledge that alternative solutions exist, including using DRAM-resident file systems or key-value stores (e.g. [31]), however, they cannot compete with the energy and cost efficiency of the solution presented here at the same capacity. Flash-based solutions can provide larger capacity than DRAM for the same cost while providing much more capability than a disk-based system. The relative costs of various types of memories/storage have been considered in series of papers, the most recent of which includes flash as well as DRAM and magnetic disk storage [43–45]. These papers use a criterion for deciding when there is a cost benefit to purchasing additional higher performance/higher cost storage based on frequency of access. The cross-over occurs when the cost of purchasing an additional increment of DRAM or Flash equals the cost of provisioning a disk-based system that can support the target access rate for a block of storage. Blocks whose target access rate exceed this threshold should be stored in the higher performing storage technology because the relative costs justify it. One can make an analogous argument to assess the relative energy costs of two storage technologies and some data regarding the relative power consumption of flash and magnetic disk is available [46].

Using the design point we are prototyping with the current BGAS system, integrating SCM into the Blue Gene/Q compute fabric should enable 1 PB (MLC flash) per half rack. As memory densities increase, for example with 3D stacking, increasingly large storage-class-memory fabrics with very high internal bandwidth become possible. The large internal bandwidth and the cost of moving data across a data center drives the key active storage design objective of embedding computation in the storage-class-memory array. Especially, with the ambitious roadmap toward human brain scale models, the cost efficiency of SCM has to be exploited. The current BGAS prototype, which anticipates the byte-addressability and hybrid nature of future SCM systems, offers an exciting research platform to explore data-centric computing today.

Contributions

The IBM team initiated the BGAS project, developed the respective hardware and software of the HS4 card and integration with the IONs as well as provided micro benchmarks and support. The CSCS team manages the system and led the development of the custom HDF5 driver and performed respective benchmarks. The EPFL team led the overall system integration and the present study.

Acknowledgments. We thank Carlos Aguado and Luc Corbeil for their active role of the definition and operation of the Blue Brain IV environment. We thank the members of the extended Active Storage hardware and software teams within IBM including Todd Takken, Heiko J. Schick, Ben Krill, Michael Deindl, Michael Kaufmann, Marc Dombrowa, David Satterfield, Ralph Bellofatto, Alda Ohmacht, and Uwe Fischer for their essential contributions to the Blue Gene Active Storage platform. We also thank the IBM Research CSS group for assembling the HS4 cards used for this work. Portions of the Blue Gene Active Storage development have been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B554331. The EPFL Blue Brain Project, the Blue Brain IV system as well as parts of this study are funded by the ETH board.

References

1. Strande, S.M., Cicotti, P., Sinkovits, R.S., Young, W.S., Wagner, R., Tatineni, M., Hocks, E., Snavely, A., Norman, M.: Gordon: Design, performance, and experiences deploying and supporting a data intensive supercomputer. In: Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond, XSEDE 2012, New York, NY, USA, pp. 3:1–3:8. ACM (2012)
2. NNSA and US DoE - Office of Science, FastForward R&D draft statement of work (March 2013), `https://asc.llnl.gov/fastforward/`
3. Lawrence livermore, intel, cray produce big data machine to serve as catalyst for next-generation hpc clusters. Press Release (November 2013)
4. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Communications of the ACM 52(4), 65–76 (2009)
5. Eleftheriou, E., Haas, R., Jelitto, J., Lantz, M., Pozidis, H.: Trends in storage technologies. Bulletin of the Technical Committee on Data Engineering 33(4), 4–13 (2010)
6. Markram, H.: The blue brain project. Nature Reviews. Neuroscience 7, 153–160 (2006), PMID: 16429124
7. Hay, E., Hill, S., Schürmann, F., Markram, H., Segev, I.: Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. PLoS Comput. Biol. 7, e1002107 (2011)
8. Reimann, M.W., Anastassiou, C.A., Perin, R., Hill, S.L., Markram, H., Koch, C.: A biophysically detailed model of neocortical local field potentials predicts the critical role of active membrane currents. Neuron 79, 375–390 (2013)
9. Hill, S.L., Wang, Y., Riachi, I., Schürmann, F., Markram, H.: Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits. Proceedings of the National Academy of Sciences 109, E2885–E2894 (2012), PMID: 22991468
10. Herculano-Houzel, S., Mota, B., Lent, R.: Cellular scaling rules for rodent brains. Proceedings of the National Academy of Sciences of the United States of America 103, 12138–12143 (2006)

11. Kozloski, J., Sfyrakis, K., Hill, S., Schürmann, F., Peck, C., Markram, H.: Identifying, tabulating, and analyzing contacts between branched neuron morphologies. IBM J. Res. Dev. 52, 43–55 (2008)
12. Migliore, M., Cannia, C., Lytton, W.W., Markram, H., Hines, M.L.: Parallel network simulations with NEURON. Journal of Computational Neuroscience 21, 119–129 (2006)
13. Tauheed, F., Biveinis, L., Heinis, T., Schürmann, F., Markram, H., Ailamaki, A.: Accelerating range queries for brain simulations. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 941–952 (April 2012)
14. Mesnier, M.P., Wachs, M., Sambasivan, R.R., Lopez, J., Hendricks, J., Ganger, G.R.: Trace: Parallel trace replay with approximate causal events. In: Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST 2007). MCDOUGALL (2007)
15. Shan, H., Shalf, J.: Using IOR to analyze the I/O performance for HPC platforms. In: Cray User Group Conference (CUG 2007) (2007)
16. May, J.: Pianola: A script-based I/O benchmark. In: Petascale Data Storage Workshop, PDSW 2008, 3rd edn., pp. 1–6 (November 2008)
17. Frings, W., Hennecke, M.: A system level view of petascale I/O on IBM blue Gene/P. Computer Science - Research and Development 26, 275–283 (2011)
18. Carns, P., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R., Ross, R.: Understanding and improving computational science storage access through continuous characterization. In: 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–14 (May 2011)
19. Xie, B., Chase, J., Dillow, D., Drokin, O., Klasky, S., Oral, S., Podhorszki, N.: Characterizing output bottlenecks in a supercomputer. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC 2012, Los Alamitos, CA, USA, pp. 8:1–8:11. IEEE Computer Society Press (2012)
20. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In: Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE 2008, New York, NY, USA, pp. 15–24. ACM (2008)
21. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel I/O to task-local files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2009, New York, NY, USA, pp. 17:1–17:11. ACM (2009)
22. Behzad, B., Luu, H.V.T., Huchette, J., Byna, S.: Taming parallel I/O complexity with auto-tuning. In: Gropp, W., Matsuoka, S. (eds.) SC, p. 68. ACM (2013)
23. Cohen, J., Dossa, D., Gokhale, M., Hysom, D., May, J., Pearce, R., Yoo, A.: Storage-intensive supercomputing benchmark study. Technical report, Lawrence Livermore National Laboratory (2007)
24. Park, S., Shen, K.: A performance evaluation of scientific I/O workloads on flash-based SSDs. In: IEEE International Conference on Cluster Computing and Workshops, CLUSTER 2009, pp. 1–5 (August 2009)
25. Jung, M., Kandemir, M.: Revisiting widely held SSD expectations and rethinking system-level implications. In: Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2013, New York, NY, USA, pp. 203–216. ACM (2013)

26. Zheng, D., Burns, R., Szalay, A.S.: Toward millions of file system IOPS on low-cost, commodity hardware. In: Proceedings of SC 2013: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2013, New York, NY, USA, pp. 69:1–69:12. ACM (2013)
27. Fitch, B., Rayshubskiy, A., Ward, T., Germain, R.: Toward a general parallel operating system using active storage fabrics on Blue Gene/P. In: Computing with Massive and Persistent Data (CMPD 2008) (September 2008)
28. Fitch, B.G., Rayshubskiy, A., Pitman, M.C., Ward, T.J.C., Germain, R.S.: Using the active storage fabrics model to address petascale storage challenges. In: Proceedings of the 4th Annual Workshop on Petascale Data Storage, PDSW 2009, New York, NY, USA, pp. 47–54. ACM (2009)
29. Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V.: Fawn: a fast array of wimpy nodes. In: SOSP 2009: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, New York, NY, USA, pp. 1–14. ACM (2009)
30. Vasudevan, V., Tan, L., Andersen, D., Kaminsky, M., Kozuch, M.A., Pillai, P.: Fawnsort: Energy-efficient sorting of 10gb. Winner of 2010 10GB Joulesort Daytona and Indy categories (2010),
http://sortbenchmark.org/fawnsort-joulesort-2012.pdf
31. Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Ongaro, D., Parulkar, G., Rosenblum, M., Rumble, S.M., Stratmann, E., Stutsman, R.: The case for RAMcloud. Communications of the ACM 54(7), 121–130 (2011)
32. Jung, M., Wilson III, E.H., Choi, W., Shalf, J., Aktulga, H.M., Yang, C., Saule, E., Catalyurek, U.V., Kandemir, M.: Exploring the future of out-of-core computing with compute-local non-volatile memory. In: Proceedings of SC 2013: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2013, New York, NY, USA, pp. 75:1–75:11. ACM (2013)
33. I. B. G. team, The IBM blue gene project. IBM Journal of Research and Development 57, 0:1–0:6 (2013)
34. Chen, D., Eisley, N.A., Heidelberger, P., Senger, R.M., Sugawara, Y., Kumar, S., Salapura, V., Satterfield, D., Steinmacher-Burow, B., Parker, J.: The IBM blue Gene/Q interconnection fabric. IEEE Micro 32(1), 32–43 (2012)
35. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: FAST 2002: Proceedings of the 1st USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, p. 19. USENIX Association (2002)
36. Haring, R., Ohmacht, M., Fox, T., Gschwind, M., Satterfield, D., Sugavanam, K., Coteus, P., Heidelberger, P., Blumrich, M., Wisniewski, R., Gara, A., Chiu, G., Boyle, P., Chist, N., Kim, C.: The IBM blue Gene/Q compute chip. IEEE Micro 32, 48–60 (2012)
37. Ryu, K.D., Inglett, T.A., Bellofatto, R., Blocksome, M.A., Gooding, T., Kumar, S., Mamidala, A.R., Megerian, M.G., Miller, S., Nelson, M.T., Rosenburg, B., Smith, B., Van Oosten, J., Wang, A., Wisniewski, R.W.: IBM blue Gene/Q system software stack. IBM Journal of Research and Development 57, 5:1–5:12 (2013)
38. OFED overview. Open Fabrics Alliance Website,
https://www.openfabrics.org/resources/ofed-for-linux-ofed-for-windows/ofed-overview.html
39. Soumagne, J., Biddiscombe, J., Esnard, A.: Data Redistribution using One-sided Transfers to In-memory HDF5 Files. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 198–207. Springer, Heidelberg (2011)

40. Biddiscombe, J., Soumagne, J., Oger, G., Guibert, D., Piccinali, J.-G.: Parallel Computational Steering for HPC Applications using HDF5 Files in Distributed Shared Memory. IEEE Transactions on Visualization and Computer Graphics 18, 852–864 (2012)
41. Ior: Github repository, `https://github.com/chaos/ior`
42. Mpich2: Official website, `http://www.mcs.anl.gov/research/projects/mpich2staging/goodell/`
43. Gray, J., Putzolu, F.: The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for cpu time. SIGMOD Rec. 16(3), 395–398 (1987)
44. Gray, J., Graefe, G.: The five-minute rule ten years later, and other computer storage rules of thumb. SIGMOD Rec. 26(4), 63–68 (1997)
45. Graefe, G.: The five-minute rule 20 years later: and how flash memory changes the rules. Queue 6, 40–52 (2008)
46. Gray, J., Fitzgerald, B.: Flash disk opportunity for server applications. Queue 6, 18–23 (2008)