The DEEP-ER project: I/O and resiliency extensions for the Cluster-Booster architecture

Anke Kreuzer, Norbert Eicker, Estela Suarez*
Jülich Supercomputing Centre (JSC)
Institute for Advanced Simulation (IAS)
Forschungszentrum Jülich GmbH
52425 Jülich, Germany
Email*: e.suarez@fz-juelich.de

Jorge Amaya
Katholieke Universiteit Leuven
BE-3001, Heverlee, Belgium

Raphäel Léger
INRIA Sophia-Antipolis Méditerranée
Sophia-Antipolis, France

***Abstract*—The recently completed research project *DEEP-ER* has developed a variety of hardware and software technologies to improve the I/O capabilities of next generation high-performance computers, and to enable applications recovering from the larger hardware failure rates expected on these machines.**

The heterogeneous Cluster-Booster architecture – first introduced in the predecessor DEEP project – has been extended by a multi-level memory hierarchy employing non-volatile and network-attached memory devices. Based on this hardware infrastructure, an I/O and resiliency software stack has been implemented combining and extending well established libraries and software tools, and sticking to standard user-interfaces. Real-world scientific codes have tested the projects' developments and demonstrated the improvements achieved without compromising the portability of the applications.

***Index Terms*—Exascale; Architecture; Cluster-Booster architecture; Co-design; Resiliency; I/O; Modular Supercomputing;**

I. Introduction

During the last decade the computing performance of HPC systems is growing much faster than their memory bandwidth and capacity [1]. This so-called *memory wall* is not expected to disappear in the near future. Additionally, higher failure rates are predicted for next generation machines due to their huge number of components. These are two of the main issues most directly affecting the scientific throughput that can be extracted from Exascale systems.

The *DEEP projects* [2] are a series of (by now) three EC funded projects (DEEP, DEEP-ER, and DEEP-EST) that address the Exascale computing challenges with their research. All three follow a stringent *co-design* strategy, in which full-fledged scientific applications guide the design and implementation of system hardware and software. Their requirements, identified by detailed application analysis, guide all the projects' developments. The selected codes have also been adapted to the project platforms and served as a yardstick to validate and benchmark the hardware and software achievements implemented in the course of the projects.

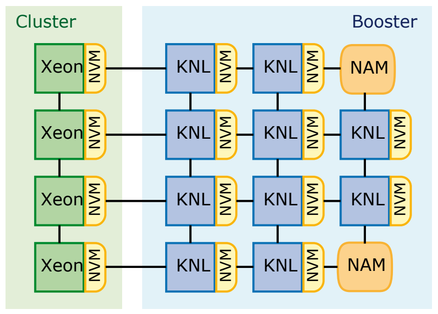
This paper describes the technology developed within the DEEP-ER project to improve the I/O and resiliency capabilities of HPC systems. In particular, the heterogeneous Cluster-Booster architecture [3] (first introduced in DEEP) was extended by a multi-level memory hierarchy. This served as a foundation of a complete I/O and resiliency software stack.

Section II of this paper presents the DEEP-ER system architecture, including the underlying Cluster-Booster concept, the specific hardware configuration of the DEEP-ER prototype, and its memory hierarchy and technologies. The software stack is explained in Section III, including the programming environment already introduced in the predecessor DEEP project, and – more detailed – the DEEP-ER I/O and resiliency software developments. The co-design applications are shortly described in Section IV. A selection of results obtained during the evaluation of the DEEP-ER concepts are presented in Section V, while Section VI puts them in context with related work. Finally, the conclusions of the paper are summarized in Section VII.

II. System Architecture

Cluster computing enables building high-performance systems benefiting from lower-cost of commodity of the shelf (COTS) components. Traditional, homogeneous clusters are built by connecting a number of general purpose processors (e.g. Intel Xeon, AMD Opteron, etc.) by a high speed network (e.g. InfiniBand or OmniPath). This approach is limited by the relatively high power consumption and cost per performance of general purpose processors. Both make a large scale homogeneous systems extremely power hungry and costly.

The cluster's overall energy and cost efficiency can be improved by adding accelerator devices (e.g. many-core processors or general purpose graphic cards, GPGPUs), which provide higher Flop/s performance per Watt. Standard heterogeneous clusters are built attaching one or more accelerators to each node. However, this *accelerated node* approach presents some caveats. An important one is the combined effect of the accelerators' dependency on the host CPU and the static arrangement of hardware resources, which limits the accessibility of the accelerators for other applications than the one running



(a) Sketch of the Cluster-Booster architecture as implemented in the DEEP-ER project (KNL: Knights Landing; NVM: non-volatile memory; NAM: network attached memory).

(b) Picture of the DEEP-ER prototype, at JSC.

Fig. 1: Cluster-Booster architecture in DEEP-ER.

on the host CPU. Furthermore, both CPU and accelerator have to compete for the limited network bandwidth in this concept.

A. *Cluster-Booster concept*

The *Cluster-Booster architecture* (Fig. 1a) integrates heterogeneous computing resources at the system level. Instead of plugging accelerators into the node and attaching them directly to the CPUs, they are moved into a stand-alone cluster of accelerators that has been named *Booster*. It is capable to run full codes with intensive internal communication, leveraging the fact that accelerators therein are autonomous and do communicate directly with each other through a high-speed network without the help of an additional CPU.

The Booster is attached to a standard HPC Cluster via a high-speed network. This connection, together with a uniform software stack running over both parts of the machine (see Section III), enables Cluster and Booster acting together as a unified system. This opens up new prospects to application developers, who have now full freedom to decide how they distribute their codes over the system. In contrast to accelerated clusters the Cluster-Booster concept poses no constraints on the combined amount of CPU and accelerator nodes that an application may select, since resources are reserved and allocated independently. Performance benefits of an application distributed over Cluster and Booster on the DEEP-ER prototype are discussed in [4].

B. *Prototype hardware configuration*

The first prototype of the Cluster-Booster concept was designed and built in the course of the *DEEP* project [2]. The later *DEEP-ER prototype* (Fig. 1b) is the second generation of the same architecture and was installed at the Jülich Supercomputing Centre (JSC) in 2016. It consists of 16 Cluster nodes and 8 Booster nodes. The system's configuration is detailed in Table I. Cluster and Booster modules are integrated into a single, air-cooled 19" rack. This rack also holds the storage system (one meta-data, two storage servers, and 57 TB of storage on spinning disks). A uniform high-speed Tourmalet A3 EXTOLL fabric runs across Cluster and

TABLE I: Hardware configuration of the DEEP-ER prototype.

Feature	**Cluster**	**Booster**
Processor	Intel Xeon E5-2680 v3	Intel Xeon Phi 7210
Microarchitecture	Haswell	Knights Landing (KNL)
Sockets per node	2	1
Cores per node	24	64
Threads per node	48	256
Frequency	2.5 GHz	1.3 GHz
Memory (RAM)	128 GB	16 GB – MCDRAM
		96 GB – DDR4
NVMe capacity	400 GB	400 GB
Interconnect	EXTOLL Tourmalet A3	EXTOLL Tourmalet A3
Max. link bandwidth	100 Gbit/s	100 Gbit/s
MPI latency	1.0 μs	1.8 μs
Node count	16	8
Peak performance	16 TFlop/s	20 TFlop/s

Booster, connecting them internally, among each other, and to the central storage [5].

1) Non-volatile Memory: The DEEP-ER prototype is enhanced by advanced memory technologies. A multi-level memory hierarchy has been built providing a total memory capacity of 8 TBytes, to enable the implementation of innovative I/O and resiliency techniques (see Sections III-C and III-D). Each node in the DEEP-ER prototype (in both Cluster and Booster) feature a non-volatile memory (NVM) device for efficient buffering of I/O operations and writing checkpoints. The chosen technology is Intel's DC P3700, a device aimed to replace SSDs, with 400 GByte capacity. It provides high speed, non-volatile local memory, attached to the node via 4 lanes of PCIe gen3. Extensive experiments and a wide range of measurements with I/O benchmarks and application mock-ups have been performed, which shows substantial performance increase over conventional best-of-breed SSDs and state-of-the art I/O servers (see Section V-A, in particular for scenarios with many I/O requests in parallel).

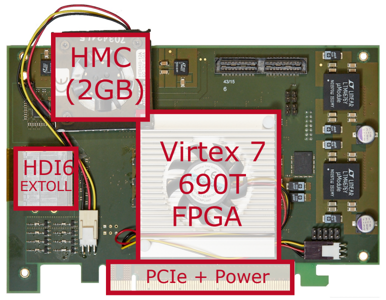


Fig. 2: Network Attached Memory (NAM) board.

2) Network Attached Memory (NAM): DEEP-ER has also introduced a new memory concept: the *network attached memory* (NAM) [6]. It exploits the RDMA capabilities of the EXTOLL fabric, which enable accessing remote memory resources without the intervention of an active component (as a CPU) on the remote side. The NAM board (Fig. 2) is built using a PCIe form-factor and acts fully autonomously. The PCIe connection is only utilized for power supply and debugging functionality. The NAM combines Hybrid Memory Cube (HMC) resources with a state-of-the-art Xilinx Virtex 7 FPGA. The DEEP-ER prototype holds two NAM devices with a capacity of 2 GBytes each. This relatively small size is due to current HMC technology limitations. The FPGA implements three functions: the HMC controller[1], the EXTOLL network interface with two full-speed Tourmalet links, and the NAM logic. Together, they create a high-speed memory device, which is directly attached to the EXTOLL fabric and therefore globally accessible by all nodes in the system.

The *libNAM* library has been implemented to give system and application software access to the NAM memory pool [6]. The library operates on top of the existing EXTOLL RMA API. The function calls provided by libNAM are very similar to EXTOLL's libRMA, such that existing applications that use the latter can be modified without much effort. Reading and writing is performed via send and receive buffers organized in a ring structure. The EXTOLL/NAM notification mechanism is used to handle the buffer space, i.e. to free up locations when data has been transmitted (`put`) or received (`get`). Fig. 3 presents bandwidth and latency measurements applying these operations on the NAM, for various message sizes. Latency and bandwidth results to read and write data from and to the NAM are very close to the best achievable values on the network alone.

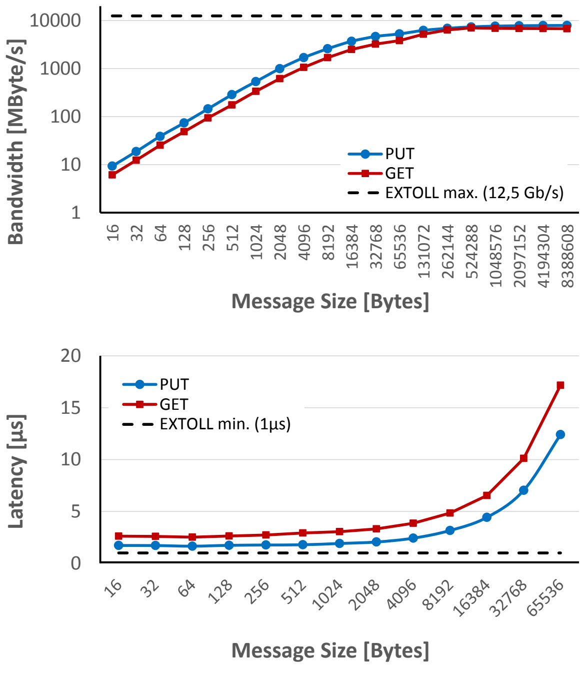


Fig. 3: RMA benchmarks (bandwidth and latency) on the NAM. Best values achievable with Extoll are also plotted.

As a first NAM use-case, an improved checkpointing/restart functionality has been implemented. It utilizes the NAM's FPGA to pull the required data from the compute nodes and to calculate parity information meant to be stored on the device locally. By this means most of the checkpointing overhead is offloaded to the NAM. At the same time this is transparent to the applications enabled to use SCR. Results obtained from applications using this functionality are presented in Section V-B, Fig. 9.

[1]The HMC controller has been released as Open Source [7].

III. Software Environment

The main guiding principle in the software development for the Cluster-Booster concept has been to stick, as much as possible, to standards and well established APIs. The specific software features required to operate Cluster and Booster as a unified system are implemented in the lower layers of the software stack and are as transparent as possible to application developers. In the end they experience a software environment very similar to the one found on any other current HPC system. By this means they do not have to deal with the underlying hardware complexity. Furthermore, their codes stay portable and can run practically out-of-the-box on this new kind of platform, as well as on any other HPC system.

A. Programming Environment

The ParaStation MPI library has been specifically optimized to efficiently run within both, Cluster and Booster, and also across them. This global MPI provides an efficient way of exchanging data between the two parts of the system [8]. It implements a heterogeneous, *global MPI* by exploiting semantic concepts long existing in the MPI-standard. In particular, the MPI-2 function `MPI_Comm_spawn` realizes the offloading mechanism, which allows spawning groups of processes from Cluster to Booster (or vice-versa).

B. OmpSs abstraction layer

Directly employing the `MPI_Comm_spawn` function call forces the programmer to coordinate and manage two or more sets of parallel MPI processes. This includes the explicit handling of the required data exchanges between Cluster and Booster. This approach may become cumbersome for large and complex applications. To reduce the porting effort, an abstraction layer employing the global MPI has been implemented already in the DEEP project. It is based on the OmpSs data-flow programming model [9]. OmpSs enables application developers offloading large, complex tasks by simply annotating those parts of their code that shall run on a different part of the system via `pragmas` [10].

C. I/O

The non-volatile memory of the DEEP-ER prototype (see Section II-B1) is used as the foundation of a scalable I/O infrastructure. The resulting software platform combines the parallel I/O library SIONlib [11] with the parallel file system BeeGFS [12]. Together, they enable the efficient and transparent use of the underlying hardware and provide the functionality and performance required by data-intensive applications and multi-level checkpointing-restart techniques.

The I/O library SIONlib acts as a concentration-layer for applications employing task-local I/O. It aims for the most efficient use of the underlying parallel file system. For this, SIONlib bundles together all data locally generated by applications, and stores it into one or very few large files, which the parallel file system manage more easily. Furthermore, in DEEP-ER SIONlib bridges between the I/O and resiliency components of the software stack (Section III-D1).

The file system utilized in DEEP-ER is BeeGFS. It provides a solid, common basis for high-performance, parallel I/O operations. Advanced functionality, such as a local cache layer in the file system, have been added to BeeGFS during the DEEP-ER project. The cache domain – based on BeeGFS on demand (BeeOND) [13] – stores data in fast node-local NVM devices and can be used in a synchronous or asynchronous mode. This speeds up the applications' I/O operations (Section V-A) and reduces the frequency of accesses to the global storage, increasing the overall scalability of the file system.

D. *Resiliency*

The DEEP-ER project has adopted an improved application-based checkpoint-restart approach, in combination with a task-based resiliency strategy.

1) Checkpoint/Restart: The Open Source Scalable Checkpoint-Restart library (SCR) offers a flexible interface for applications to write checkpoint information and to restart from those checkpoints in case of failure [14]. The user simply calls SCR and indicates the data required by the application to restart execution. This library keeps a database of checkpoints and their locations in preparation for eventual reinitializations.

Combining SCR features with project-specific software and hardware developments, four checkpoint/restart strategies can be employed in DEEP-ER, listed here ordered from most basic to more advanced:

- **Single**: The `SCR_SINGLE` feature stores checkpoints locally on each node. This enables applications to recover from transient errors.
- **Buddy**: This enhancement of the SCR's *Partner* mode combines SCR, ParaStation MPI, SIONlib, and BeeGFS. The standard `SCR_PARTNER` mode first saves checkpoint-data to the local node, re-reads it from there, sends it to a *Partner* node and writes it to its local storage. This strategy enables application to restart even after node-failures, recovering data from the companion node. Thus, the application can continue with minimum time-loss. The DEEP-ER *Buddy* enhancement utilizes SIONlib to skip the intermediate step of reading the data from the local storage before sending it to the partner node, reducing the checkpointing overhead. While SCR in this approach keeps track of the association between host nodes and buddies, SIONlib takes care that all MPI processes running on a single node jointly write their checkpoint-data into a single file on the buddy-node. Finally, BeeOND stores the data itself on the cache file system on the local NVMe, eventually transferring it asynchronously to the permanent global storage.
- **Distributed XOR**: Both `SCR_PARTNER` and DEEP-ER's *Buddy* mode save the whole checkpoint-data twice. This obviously doubles the required amount of memory per node and generates significant overhead due to the additional writing time. An alternative strategy is to generate and store parity information, instead of copying the full checkpoints. SCR can perform this by applying an XOR operation to the checkpointing data. In a second step it distributes the resulting parity data over all nodes. If one node fails, the missing checkpoint-data can be reconstructed by combining the parity data and the checkpoint data from the remaining nodes. This method saves the checkpoint data on the node-local NVMe (as in the *Single* mode) and distributes and remotely stores only the parity data, which is much smaller than the full checkpoints.
- **NAM XOR**: The network-attached memory (NAM) technology developed in DEEP-ER is an ideal vehicle to accelerate the *Distributed XOR* strategy. SCR and SIONlib call libNAM to trigger data collection and parity computations on the NAM. Furthermore, XOR-data is stored on this central location. This enables application developers to transparently checkpoint/restart to/from the NAM without modifying their codes. The high read/write performance of the NAM (Fig. 3) and its accessibility at network-speed from all nodes in the system, largely reduces the checkpointing overhead when compared to the previous mode. At the same time it provides a similar level of resiliency as the *Buddy* mode.

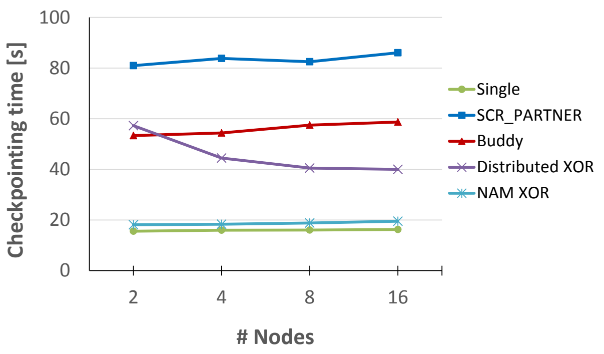


Fig. 4: N-body code testing various checkpointing strategies on the DEEP-ER Cluster (weak scaling).

Weak-scaling results obtained testing these functionalities on the DEEP-ER Cluster with the N-body code are presented in Fig. 4. The *Buddy* and NAM XOR methods developed in the DEEP-ER project are both faster than the equivalent SCR functions: `SCR_PARTNER` and *Distributed XOR*, respectively. Results of a full application comparing the *Distributed XOR* with the *NAM XOR* modes are discussed in Section V-B.

2) OmpSs resiliency: The OmpSs programming model has been also enhanced by three new resiliency features.

- **Lightweight task-based checkpoint/restart mechanism** writes the input of the OmpSs tasks into main memory before starting them. Thus, they can be restarted in case of failure. If no error occurs and the task finishes successfully, the checkpoint is evicted.

- **Persistent task-based checkpointing** saves all input dependencies of a task. When the application is restarted after a crash, OmpSs transparently identifies the execution as a recovery and fast-forwards it to the point where the failure occurred, restoring the appropriate data.
- **OmpSs resilient offload** is applied specifically to the offload mechanism developed in the DEEP projects (Section III-B). The ParaStation process management daemon has been extended by an interface to query resiliency-related status information from the MPI layer and thus also from the OmpSs runtime environment. ParaStation MPI itself is now able to detect, isolate and clean up failures of MPI-offloaded tasks, which can be then independently restarted without requiring a full application recovery. This enables to recover failed offloaded tasks without losing the work that had been performed in parallel by other OmpSs tasks.

Application benchmarks utilizing the third OmpSs resiliency approach are described in Section V-B.

IV. Co-design applications

Seven real-world HPC applications were chosen to steer and evaluate the design of the DEEP-ER hardware and software developments (Sections II and III), and to benchmark their functionality and performance. The DEEP-ER applications come from a wide range of scientific areas, representing the typically broad user portfolio of a large-scale computer centre. For the sake of brevity, details are given here only for three of the codes. Their results (Section V) cover almost all the I/O and resiliency features developed in DEEP-ER:

- **xPic** is a simulation code from KU Leuven (Katholieke Universiteit Leuven) to forecast space weather related events like e.g. damage of spacecraft electronics or GPS signal scintillation. It simulates the inter-planetary plasma using the Moment-Implicit method [15]. Like most particle-in-cell codes, xPic consists of two parts, a particle solver and a field solver: The particle solver calculates the motion of charged particles in response to the electromagnetic field and the gathering of their moments (e.g. net current and charge density); the field solver computes the electromagnetic field evolution in response to the moments.
- **GERShWIN** [16] assesses human exposure to electromagnetic fields and is provided by Inria (Institut National de Recherche en Informatique et en Automatique). This application uses a Discontinuous Galerkin - Time Domain solver of the 3D Maxwell-Debye equation system [17] to simulate the propagation of electromagnetic waves through human tissues. This field of research studies for instance the effect of wireless communication devices on head tissues or the implantation of antennas in the human body for the purpose of monitoring health-related devices.
- **FWI** [18] is a seismic imaging code developed by the Barcelona Supercomputing Center (BSC). Seismic imaging uses sound waves to acquire the physical properties of the subsoil from a set of seismic measurements. The application starts from a guess (initial model) of the variables (e.g., sound transmission velocity), the stimulus introduced, and the recorded signals. For the inversion several phases of iterative computations (frequency cycles) are done until the real value of the set of variables being inverted is reached (with an acceptable error threshold).

Further applications used in DEEP-ER are:

- **SKA data analysis pipeline** by ASTRON (Netherlands Institute for Radio Astronomy).
- **TurboRvB** from CINECA (Consorzio Interuniversitario del Nord-Est per il Calcolo Automatico).
- **SeisSol** from LRZ (Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften).
- **CHROMA**: by the University of Regensburg.

The role of the applications in DEEP-ER is two-fold: on the one hand, their requirements have provided co-design input to fix the characteristics of hardware and software components; on the other hand, the codes have evaluated the project developments by running different uses cases on the DEEP-ER prototype. Examples of the co-design influence are the determination of the amount of memory to be available per node, the required MPI functionality when offloading code from one side to the other of the system, or the way in which the NAM should be addressed. A selection of the application results achieved by the first three listed codes (xPic, GERShWIN, and FWI) is presented in the next section.

V. Results

The hardware and software concepts developed in DEEP-ER have been evaluated using the co-design applications. Unless stated otherwise, all measurements have been obtained on the DEEP-ER prototype (Section II-B). The codes have been used in various simulation scenarios in order to test different system features (e.g. input parameters leading to more data communication were used to stress I/O features).

A. I/O application results

The features described in Section III-C have lead to several I/O improvements in the DEEP-ER applications. Some of them are shown here since the I/O capabilities have a direct impact on checkpointing performance. The setup of all I/O experiments in this section is described in Table II.

Fig. 5 shows the reduction of data writing time for the GERShWIN application when using SIONlib to collectively carry out task-local I/O operations into a reduced number of files. Different use cases where tested, varying the Lagrange order of the calculations (order three (P3) requires more data and provides higher precision than order one (P1)). Significant performance improvements are achieved when using SIONlib: up to 7.4$\times$ faster for P1, and up to 3.7$\times$ for P3.

Even with the help of SIONlib, using the file system to execute I/O operations to the global storage from a large number of compute nodes may still lead to a bottleneck at the storage: once the maximum storage bandwidth is reached, the bandwidth per node decreases when additional nodes participate in I/O. In DEEP-ER this I/O scalability issue is targeted

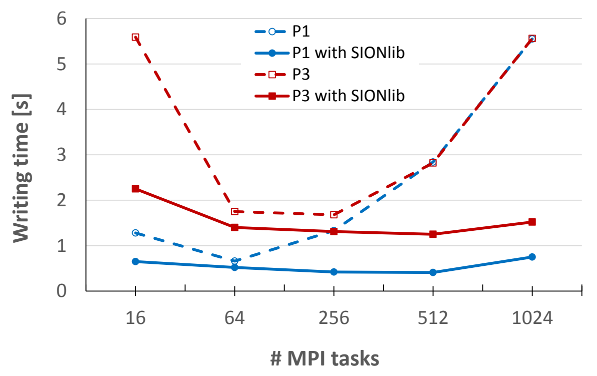


Fig. 5: I/O improvement through SIONlib measured with GERShWIN.

by the BeeGFS caching-layer, which transparently employs the node-local NVMe devices (Section II-B1) as scalable local storage. This has the effect of a constant storage bandwidth per node, significantly increasing the I/O performance and scalability of applications.

Due to the small size of the DEEP-ER prototype, such scaling effects had to be measured on an alternative platform. The QPACE3 system was selected, which is a Booster-like platform with 672 KNL nodes [19], large enough to perform scalability measurements. Fig. 6 shows weak-scaling studies with xPic on QPACE3.

A software configuration similar to the DEEP-ER platform could be installed on QPACE3. Since this system lacks node-local NVMe devices, these had to be emulated by RAM-disks residing in the local memories of each node. The absolute performance numbers are not comparable to NVMe (RAM on KNL is 75× faster than NVMe) but the advantage of using local storage devices with respect to the global storage system is clearly demonstrated, as well as the evident gain in performance when increasing the number of nodes. In fact, the application scales almost perfectly when using local storage. This makes the application 7× faster compared to writing directly to QPACE3's global file system which is also BeeGFS.

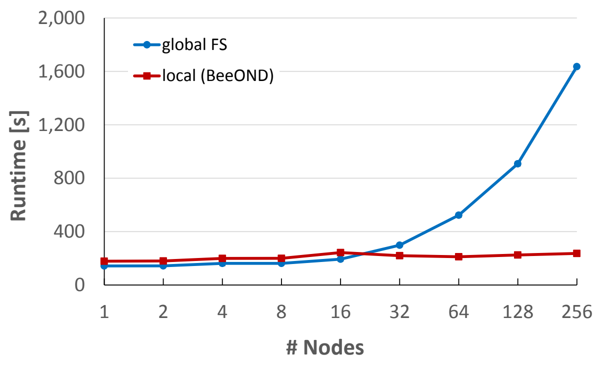


Fig. 6: xPic on QPACE3: writing on global file system vs. using node-local storage with BeeOND.

As demonstrated on QPACE3, the concept of writing to node-local storage does not necessarily require NVMe devices to be attached to the compute nodes. The same strategy can be employed with other node-local storage technologies, leading to different absolute performance numbers. Fig. 7 presents xPic measurements on the DEEP-ER Cluster. Here both NVMe and hard disks (HDD) are attached to each node. Writing to the NVMe storage is up to 4.5× faster than writing to the node-local HDD. The absolute performance gain depends on the number of nodes performing I/O and, as explained above, the actual benefit from the NVMe-based local storage actually manifests when the node-count is very large.

TABLE II: Experiment setup used in GERShWIN and xPic during the I/O measurements.

Experiment	**GERShWIN**	**xPic on QPACE3**	**xPic on the DEEP-ER prototype**
Written data per checkpoint	3 GB (P1) 6.6 GB (P3)	10 GB per node	8 GB
Number of CPs	1	2	11

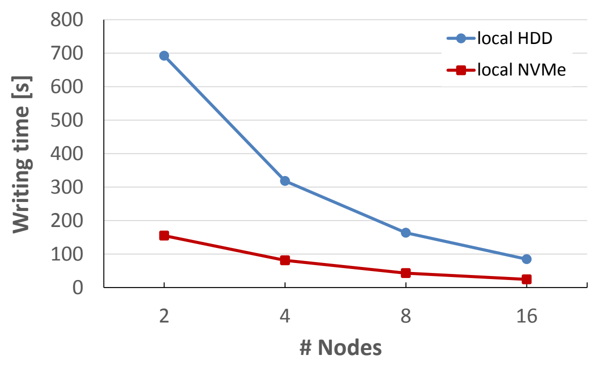


Fig. 7: I/O operations to node-local NVMe vs. node-local HDD storage, measured by xPic on the DEEP-ER Cluster.

B. *Resiliency*

During the DEEP-ER project multiple resiliency features have been developed (Section III-D). The setup of the resiliency experiments described here can be found in Table III.

Fig. 8 displays the overhead and benefit of using the SCR library to save checkpoints (CP) on the node-local NVMe (`SCR_PARTNER`). An xPic benchmark was selected that executes 100 iterations in the simulation. The benchmark was run with and without `SCR_PARTNER`. In the latter case checkpoints are written every 10 iterations. Two error scenarios were tested: the first completes without error; in the second an error happens after 60 iterations and then the application is restarted and runs through. The measurements show that the overhead incurred by writing checkpoints with SCR is in average only 8%, while it saves 23% of the execution time if a failure occurs according to the scenario.

The checkpointing overhead can be further reduced calculating parity data (Section III-D1). Fig. 9 shows a com-

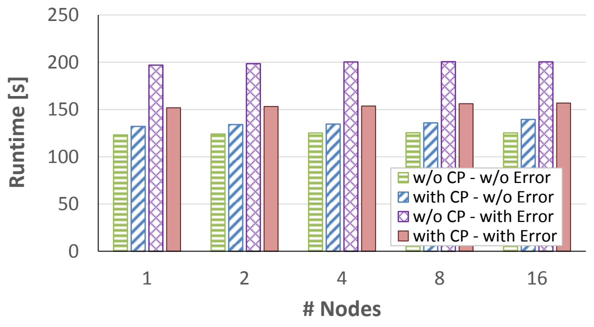


Fig. 8: xPic testing `SCR_PARTNER`. Tests done writing checkpoints (with CP) or not (w/o CP), for runs when an error occurs (with) or not (w/o).

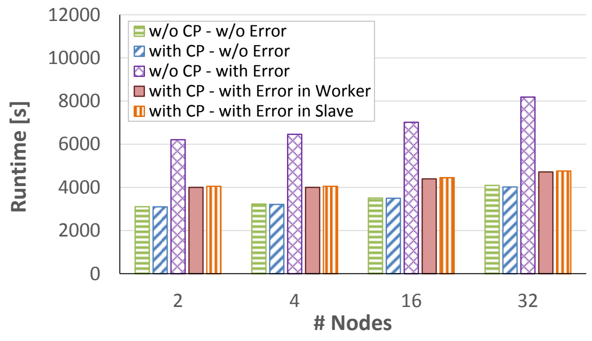


Fig. 10: OmpSs task-based resiliency tested with FWI (with and without (w/o) checkpoints, with error - in worker or slave - and w/o errors).

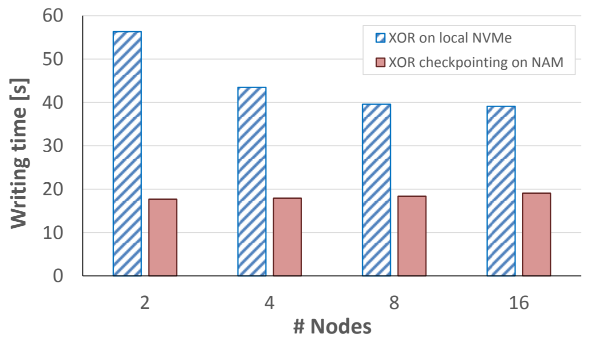


Fig. 9: *Distributed XOR* vs. *NAM XOR* checkpointing strategies, evaluated with xPic.

parison between the *Distributed XOR* and the *NAM XOR* checkpointing strategies. The latter realizes an up to 3× higher bandwidth, and leads to much better writing times: between 50% and 65% of time is saved when storing XOR data to the NAM instead of storing it to the node-local NVMe devices.

An alternative strategy applied in DEEP-ER to increase the applications' robustness against system failures is the OmpSs-offload resiliency functionality (Section III-D2). Fig. 10 shows the results achieved when testing this approach with the FWI code, on an Intel Sandy Bridge cluster (MareNostrum 3, at BSC). An error occurring right before the end of the execution nearly doubles the FWI runtime if no resiliency technique is activated. The new OmpSs feature enables up to 42% time savings (an only 15% longer execution when compared to a run without failures) and its overhead is negligible (<1%).

TABLE III: Experiment setup for the resiliency measurements.

Experiment	**xPic SCR**	**xPic NAM**	**FWI**
Processed data	32 GB per node 8 GB per CP 4 CPs	20 GB per node 2 GB per CP 10 CPs	1 GB per node

VI. Related work

The work on resiliency presented in this paper is based on the Scalable Checkpoint-Restart library(SCR) [14]. During the DEEP-ER project a tight integration of SIONlib [11] into SCR was established. Task-local I/O as done by SCR typically uses many independent files. SIONlib concentrates them into single or few shared files on a parallel file-system, what makes this type of I/O much more efficient. In DEEP-ER SIONlib is used as an abstraction-layer for both, buddy checkpointing (similar to SCR_PARTNER) and NAM integration – which might be seen as an hardware acceleration of SCR's XOR checkpointing feature. Both approaches significantly improve the I/O performance by preventing unnecessary read operations when creating partner and XOR checkpoints.

A similar approach as SCR is realized in the Fault Tolerance Interface (FTI) [20]. In the meantime an effort was started to create a common abstraction of SCR and FTI, to help application developers avoiding code-adaptations to the specific checkpointing tool installed on a given HPC system.

Transparent system level checkpointing is realised in Berkeley Lab Checkpoint Restart (BLCR) [21]. In this approach applications do not have to be modified in order to checkpoint them like it is necessary in SCR or FTI, where explicit store and load operations have to be introduced into the codes in order to create checkpoints with all relevant data. The drawback of transparent checkpointing is that checkpoint wills grow much larger, since the whole memory of the application has to be dumped onto the underlying storage system. Furthermore, this type of checkpointing is harder to implement since all MPI communication has to be brought into a globally consistent state in order to get properly checkpointed, too.

The same approach as BLCR is used by the Distributed MultiThreaded Checkpointing (DMTCP) efforts [22] therefore sharing the pros and cons.

VII. Conclusions and Outlook

The DEEP-ER project has introduced several hardware and software innovations to improve the I/O and resiliency capabilities of the Cluster-Booster architecture, and of HPC systems in general. The central component is a multi-level memory hierarchy employing non-volatile memory (NVM) devices, locally attached to each of the nodes in the system.

The DEEP-ER I/O software system combines proven file-systems and libraries with extensions that allow optimal exploitation of the capabilities of the memory and storage pool. The project's resiliency strategy relies on the combination of complementary functions to recover from different kinds of errors with reduced overhead. Building upon the infrastructure provided by the Scalable Checkpoint/Restart library SCR, the DEEP-ER extensions reduce the checkpointing overhead keeping the same level of resiliency. An example is the *Buddy* checkpointing functionality that employs SIONlib to optimize `SCR_PARTNER`. Application resiliency is achieved with even better performance employing the network-attached memory (NAM) technology developed in DEEP-ER. This special "memory node" is globally accessible from all nodes and enables calculating and storing parity data of application checkpoints much faster than if done on the nodes themselves. The achieved improvements in performance and resiliency have been demonstrated with real-world applications.

The Cluster-Booster architecture – which was first prototyped in the DEEP projects series – has gone into production in the meantime. The JURECA Cluster, running at JSC in Germany since 2015 [23], has been recently accompanied by a KNL-based, 5 PFlop/s Booster. The JURECA Booster is planned to become available to users in Q1/2018.

The DEEP-ER project is now completed and successfully evaluated by external reviewers. Building on its results, the successor (DEEP-EST) project generalizes the Cluster-Booster concept to create the *Modular Supercomputing architecture* [24]. It combines any number of compute modules into a single computing platform. Each compute module is (as the Cluster and the Booster) a system of a potentially large size, tailored to the specific needs of a given kind of applications. To demonstrate its capabilities, a three-module hardware prototype will be built, covering the needs of both HPC and high performance data analytics (HPDA) workloads.

Acknowledgements

The authors thank all members of the DEEP-ER consortium for their strong commitment in the project, which led to several results described in this paper. Special gratitude goes to J. Schmidt (University of Heidelberg) for the NAM results, A. Galonska (JSC) for buddy-checkpointing benchmarks, and S. Rodríguez (BSC) OmpSs resiliency tests with FWI.

Part of the research presented here has received funding from the European Community's FP7/2007-2013 and H2020-FETHPC Programmes, under Grant Agreement n° 287530 (DEEP), 610476 (DEEP-ER), and n° 754304 (DEEP-EST). The present publication reflects only the authors' views. The European Commission is not liable for any use that might be made of the information contained therein.

References

[1] S.A. McKee, *Reflections on the Memory Wall*, Proceedings of the 1st Conference on Computing Frontiers (CF '04), p. 162, (2004) isbn = 1-58113-741-9, [doi = 10.1145/977091.977115].

[2] http://www.deep-projects.eu

[3] N. Eicker and Th. Lippert, *An accelerated Cluster-Architecture for the Exascale*, PARS '11, PARS-Mitteilungen, Vol. 28, p. 110–119 (2011).

[4] A. Kreuzer, J. Amaya, N. Eicker, E. Suarez, *Application performance on a Cluster-Booster system*, Accepted for publication at the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops Proceedings (HCW), IPDPS 2018 Conference, Vancouver (2018).

[5] *EXTOLL GmbH website:* http://www.extoll.de

[6] J. Schmidt, *Network Attached Memory*, Chapter 4 of the PhD Thesis: *Accelerating Checkpoint/Restart Application Performance in Large-Scale Systems with Network Attached Memory*, Ruprecht-Karls University Heidelberg (Fakultät für Mathematik und Informatik) http://archiv.ub.uni-heidelberg.de/volltextserver/23800/1/dissertation_juri_schmidt_publish.pdf

[7] http://www.uni-heidelberg.de/openhmc

[8] N. Eicker, Th. Lippert, Th. Moschny, and E. Suarez, *The DEEP Project - An alternative approach to heterogeneous cluster-computing in the many-core era*, Concurrency and computation: Practice and Experience, Vol. 28, p. 2394—2411 (2016), [doi = 10.1002/cpe.3562].

[9] A. Duran, E. Ayguadé, R.M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, *OmpSs: A proposal for programming heterogeneous multi-core architectures*, Parallel Processing Letters, Vol. 21(2), p. 173-193 (2011) [doi = 10.1142/S0129626411000151].

[10] F. Sainz, J. Bellón, V. Beltran, and J. Labarta, *Collective Offload for Heterogeneous Clusters*, 2015 IEEE 22nd International Conference on High Performance Computing (HiPC), p. 376-385 (2015) [doi = 10.1109/HiPC.2015.20].

[11] W.Frings, F. Wolf, and V. Petkov, *Scalable Massively Parallel I/O to Task-Local Files, Proceedings of SC'09, Portland, USA New York, ACM, Technical papers, Article. No. 17, p.1-11 (2009) isbn = 978-1-60558-744-8. http://juser.fz-juelich.de/record/4447*

[12] https://www.beegfs.io/content/

[13] https://www.beegfs.io/wiki/BeeOND

[14] A. Moody, G. Bronevetsky, K. Mohror, B.R. Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, IEEE Computer Society, p. 1-11 (2010) isbn = 978-1-4244-7559-9, [doi = 10.1109/SC.2010.18].*

[15] S. Markidis and G. Lapenta, Multi-scale simulations of plasma with iPIC3D*, in Mathematics and Computers in Simulation, Vol. 80, No. 7, pp. 1509-1519 (2010).*

[16] R. Leger, D. Alvarez Mallon, A. Duran, S. Lanteri, Adapting a Finite-Element Type Solver for Bioelectromagnetics to the DEEP-ER Platform*, Chapter in Book "Parallel Computing: On the Road to Exascale", Advances in Parallel Computing, Vol. 27, p. 349-359 (2015). [doi = 10.3233/978-1-61499-621-7-349].*

[17] S. Lanteri and C. Scheid, Convergence of a Discontinuous Galerkin scheme for the mixed time domain Maxwell's equations in dispersive media*, Article in Journal "IMA Journal of Numerical Analysis", https://hal.archives-ouvertes.fr/hal-00874752, Vol. 33, No. 2, pp. 432-459 (2013). [doi = 10.1093/imanum/drs008].*

[18] http://www.deep-projects.eu/applications/project-applications/enhancing-oil-exploration.html

[19] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/QPACE3/_node.html

[20] L. Bautista-Gomez, et al. FTI: high performance fault tolerance interface for hybrid systems. *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis. ACM, 2011.*

[21] P. H. Hargrove, J. C. Duell, Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters. *Proceedings of SciDAC 2006: June 2006*

[22] J. Ansel, K. Arya, G. Cooperman, DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop*, Proceeding of the 23rd IEEE International Parallel and Distributed Processing Symposium", May 2009, Rome, Italy*

[23] D. Krause, and Ph. Thörnig, JURECA: General-purpose supercomputer at Jülich Supercomputing Centre*, Journal of large-scale research facilities, Vol.2, A62, (2016), [doi = 10.17815/jlsrf-2-121].*

[24] E. Suarez, N. Eicker, Th. Lippert, Supercomputing Evolution at JSC*, Proceedings of the 2018 NIC Symposium, Vol.49, p.1-12, (2018), [online: http://juser.fz-juelich.de/record/844072].*