SEVENTH FRAMEWORK PROGRAMME

FP7-ICT-2013-10

DEEP-ER

DEEP Extended Reach

Grant Agreement Number: 610476

D3.3

Non-Volatile Memory (NVM) assessment

Approved

Version: 2.0

Author(s): M. Cintra (Intel)

Contributor(s): H. Ch. Hoppe (Intel)

Date: 08.06.2016

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="7">DEEP-ER Project</td><td colspan="2">Project Ref. №: 610476</td></tr>
<tr><td colspan="2">Project Title: DEEP Extended Reach</td></tr>
<tr><td colspan="2">Project Web Site: http://www.deep-er.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D3.3</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td>Deliverable Level:
PU*</td><td>Contractual Date of Delivery:
31 / January / 2016</td></tr>
<tr><td>Actual Date of Delivery:
29 / January / 2016
(Updated before review 17.05.2016)</td></tr>
<tr><td colspan="3">EC Project Officer: Panagiotis Tsarchopoulos</td></tr>
</table>

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

<table>
<tr><td rowspan="5">Document</td><td colspan="2">Title: Non-Volatile Memory (NVM) assessment</td></tr>
<tr><td colspan="2">ID: D3.3</td></tr>
<tr><td>Version: 2.0</td><td>Status: Approved</td></tr>
<tr><td colspan="2">Available at: www.deep-er.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word</td></tr>
<tr><td colspan="2">File(s): DEEP-ER_D3.3_NVM_Assessment_v2.0-ECapproved</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>M. Cintra (Intel)</td></tr>
<tr><td>Contributors:</td><td>H. Ch. Hoppe (Intel)</td></tr>
<tr><td>Reviewed by:</td><td>G.Brietzke (BADW-LRZ), E.Suarez (JUELICH)</td></tr>
<tr><td>Approved by:</td><td>BoP/PMT</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
1.0	29/January/2016	Final version	EC-submission
1.1	17/May/2016	Extended version	Updated before review M33 to include additional results collected after the initial submission
2.0	08/June/2016	Approved	EC approved

Document Keywords

Keywords:	DEEP-ER, HPC, Exascale, Non-volatile memory, NVM, memory technologies, SSD.

Table of Contents

List of Figures

Executive Summary

This document presents an assessment of the functional and performance characteristics of non-volatile memory (NVM) storage in DEEP-ER. Within the scope of DEEP-ER, it is envisioned to use NVM storage in the Booster nodes. The goal is to provide persistent storage local to the nodes that can be used to accelerate the storage sub-system as well as to enable extended functionality such as local checkpoints for reliability.

The evaluation presented in this document concentrates on establishing the performance gains that can be expected with local NVM storage versus system configurations with only remote storage. For this purpose, two platforms were used: a single node connected to networked storage (NFS) and the DEEP Cluster prototype [1], which is connected to GPFS. Both systems have local NVM SSDs. The systems were evaluated using a variety of workloads.

Since the original submission of this document, the installation of the DEEP-ER SDV (Software Development Vehicle) has been completed and the system and made available. The SDV also contains NVM local SSDs in the compute nodes as well as an intermediate level of storage with HDDs in a storage server. In this extended version of the document we show results comparing local storage versus intermediate storage in this system using one of the DEEP-ER applications.

1 Introduction

In Deliverable 3.2 we performed an early evaluation of the performance characteristics of PCIe attached SSDs using industry-standard micro-benchmarks, and in Deliverable 8.1 we presented our proposal of how to integrate these SSDs in the DEEP-ER Aurora Blade design. In this document we extend that early evaluation with performance studies using real-world workloads taken from the set of workloads expected to be run on the DEEP-ER system. Additionally, we evaluate the SSDs against remote storage. We note that the use of node-local storage was one of the key novel concepts in terms of storage of the DEEP-ER project.

In the following sections we first present an evaluation of PCIe attached SSDs in a single-node system using mock-ups of two DEEP-ER applications, as well as a tracing tool. The focus of these experiments is to assess the performance advantage of node-local storage against NFS. We then present an evaluation of SATA attached SSDs in the DEEP Cluster using a set of benchmarks designed to assess storage I/O scalability. This is followed by some early results evaluating the storage I/O overhead of one of the DEEP-ER applications also running on the DEEP Cluster. Finally – and post original submission of the document – we present a comparison of local PCIe attached SSDs versus intermediate HDD storage in storage servers in the DEEP-ER SDV. A description of the hardware configuration of all three systems is presented in the Annex.

2 Evaluation of node-local NVM versus NFS in single-node system

In this section we present a performance evaluation of local versus remote storage in the context of the DEEP-ER project. For this purpose we use a single-node system (called KNC2 hereafter) configured with PCIe attached SSDs and connected to NFS-based remote storage. While the focus is on comparing local and remote storage, we also provide a comparison between the PCIe attached SSDs and SATA attached SSDs as a secondary result. The evaluation is done using mock-ups of two DEEP-ER applications and one tracing tool. A description of the hardware configuration can be found in the Annex, and detailed descriptions of the two DEEP-ER applications can be found in Deliverable 6.1.

2.1 Evaluation using iPic3D mock-up

We evaluated a mock-up that emulates the storage I/O behaviour of the iPic3D application from KULeuven. This application consists of a particle-in-cell (PIC) code that simulates plasma using a semi-implicit method called the Moment-Implicit method. Like most PIC codes, it consists of two parts, a particle solver that simulates the motion of charged particles in response to the electromagnetic field, and a field solver that simulates the electromagnetic field evolution in response to "moments" (e.g. net current and charge density) of the particles. Each cycle of the algorithm advances the system state by one time step and consists of three sub-steps: advance fields implicitly using particle moments, move particles implicitly using fields, and calculate moments of particles. The application and its I/O characteristics are described in detail in Deliverable 6.1. In its current implementation the mock-up does not emulate the computation and communication behaviours of the application. Thus, any performance difference reported here between node-local NVM storage and remote storage are only representative of the I/O phase and should be seen as loose upper bounds for the entire application.

As explained in Deliverable 6.1, the iPic3D application generates two main output data-structures: fields and particles. The amount of data generated for fields and particles is mainly controlled by the number of cells in the grid. In addition to the grid size, the other main parameter of the application is the number of time steps. In terms of storage I/O volume, another important parameter is the number of time steps between two outputs. The parameters used for this initial evaluation are those recommended by the application developers: from 64×64×1 cells to 256×256×1 cells per node, 30K time steps, and both fields and particles output every 1K time steps.

Figure 1 shows the number of cells (bars and right y-axis) and the size of the periodic output files (lines and left y-axis) for three grid sizes. The amount of field data, as measure by the corresponding output file size, is both negligible and insensitive to the number of cells. On the other hand, the amount of particle data is proportional to the number of cells and becomes significant for large grids. In the current workflow of iPic3D, particles are periodically saved only for checkpoint-restart, but future workflows are expected to use such periodic data for offline analysis and visualization.

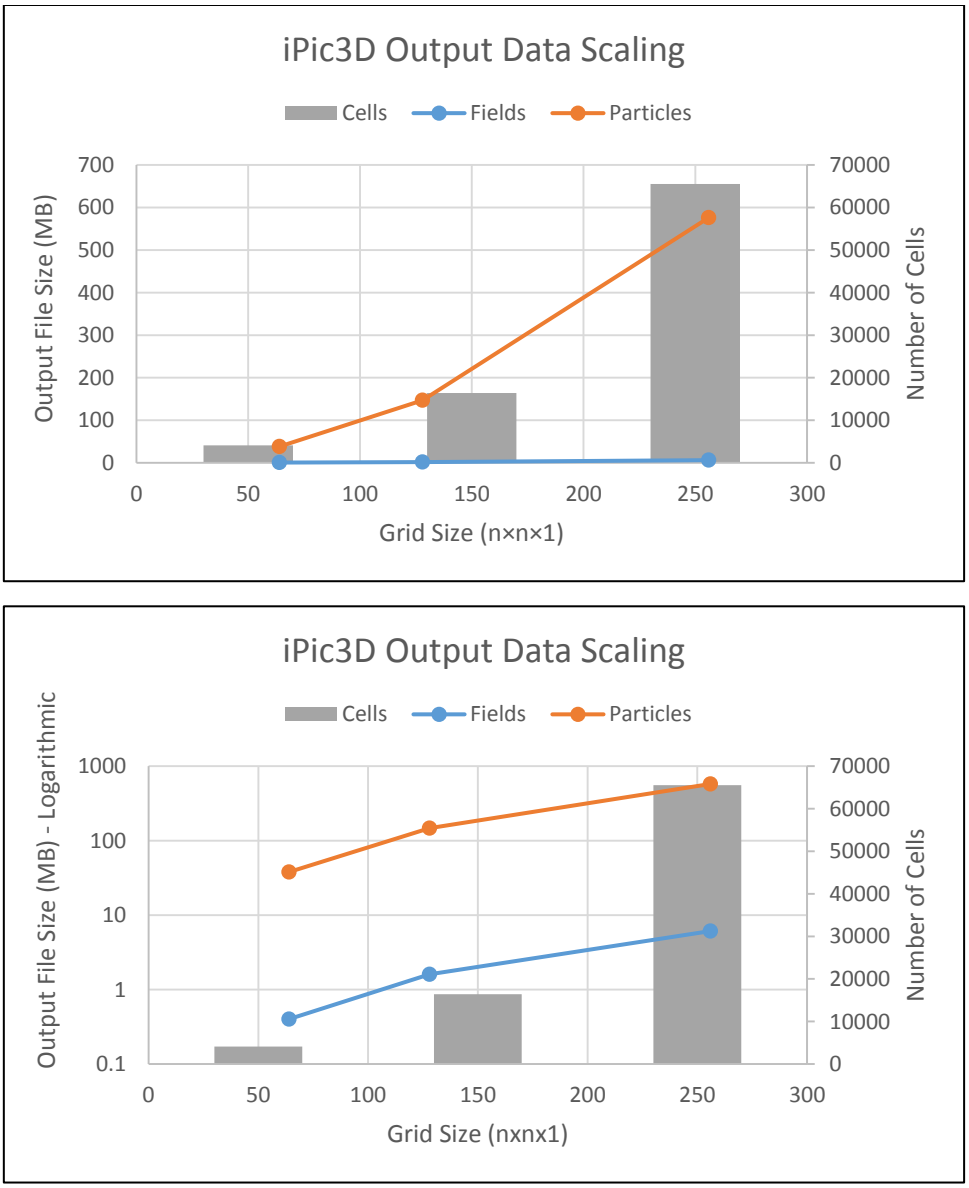

Figure 1: Output file sizes and number of cells as a function of grid size for the iPic3d I/O mock-up. The top chart shows the output file sizes in a linear scale while the bottom chart shows them in a logarithmic scale.

The execution time of the I/O storage mock-up of iPic3D is measured while directing the output files to either the locally attached NVMe SSD, the locally attached SATA SSD, or the NFS. The results are shown in Figure 2 for the three grid sizes that we used. From this figure we can see that the execution time of the I/O component of iPic3D increases, as expected, with grid size (and following the size of the particles output, as seen above). The execution time and its increase with larger grid sizes are significantly larger with NFS than with the SSDs. This clearly shows that having local storage to save the periodic output of iPic3D can lead to significant I/O performance improvements compared to remote storage. Comparing the execution time of the mock-up with NVMe and SATA we observe that the local NVMe SSD outperforms the SATA SSD by a significant margin (2× to 4× faster). This is in line with the results that we obtained earlier with I/O micro-benchmarks (discussed in Deliverable 3.2).

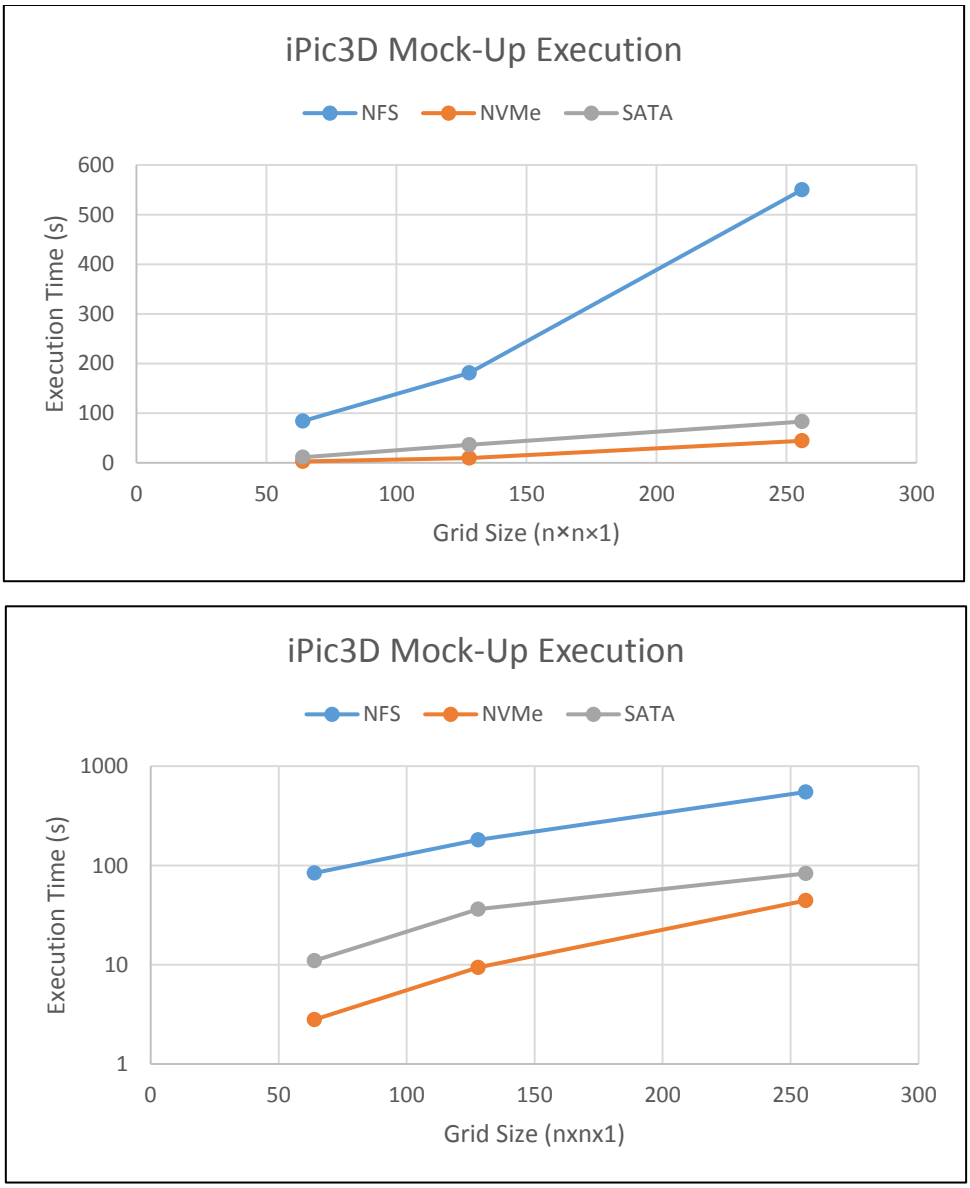

Figure 2: Execution time of the iPic3D I/O mock-up with output sent to NFS, local NVMe SSD, or local SATA SSD. The top chart shows the execution time in a linear scale while the bottom chart shows it in a logarithmic scale.

Figure 3 provides another view of the data, where we show the speedup of the NVMe SSD over NFS and the SATA SSD, as well as the speedup of the SATA SSD over NFS. From this figure we can see that the advantage of local NVMe SSD over NFS is indeed significant, reaching 30× for the smaller output. This advantage decreases, as expected, for larger outputs, but is still significant (12×) even for the largest output.

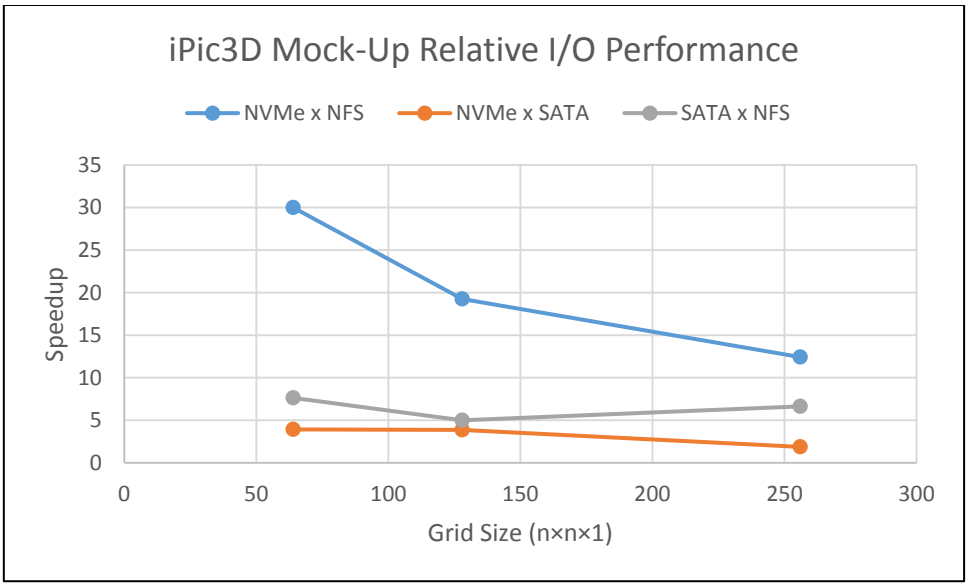

Figure 3: Speedup for iPic3D I/O mock-up of NVMe SSD over NFS and SATA SSD, and of SATA SSD over NFS.

2.2 Evaluation using BSIT mock-up

We evaluated a mock-up that emulates both the storage I/O as well as the computation/communication behaviours of the BSIT (Full Waveform Inversion) application from BSC. This application uses Full Waveform Inversion, a technique that aims to acquire the physical properties of the subsoil from a set of seismic measurements. Starting from a guess (initial model) of the variables being inverted (e.g., sound transmission velocity), the stimulus introduced and the recorded signals, Full Waveform Inversion performs several phases of iterative computations to reach the real value of the set of variables being inverted with an acceptable error threshold. The application and its I/O characteristics are described in detail in Deliverable 6.1. The application outputs the current state of the forward propagation wavefield every few iterations of the inner loop. The amount of data generated depends primarily on the frequency parameter, but also to a lesser degree on other simulation parameters. On a single node the largest application configuration is limited by the available main memory, which then sets the range of frequencies that may be used. The parameters used for this initial evaluation are then those that lead to the maximum main memory footprint: frequencies of 6Hz and 12Hz and grid sizes of 8K×8K×8K and 15K×15K×15K. Output is generated after every step.

Figure 4 shows the size of the periodic output file for three sets of input parameters. The three sets of parameters are:

1) 8K×8K×8K grid, frequency = 6Hz;
2) 15K×15K×15K grid, frequency = 6Hz; and
3) 8K×8K×8K grid, frequency = 12Hz.

As shown in this figure the periodic output of the BSIT application is very large and scales very quickly with the problem size and the frequency parameter, reaching over 21 GB for an input set that fills the node main memory.

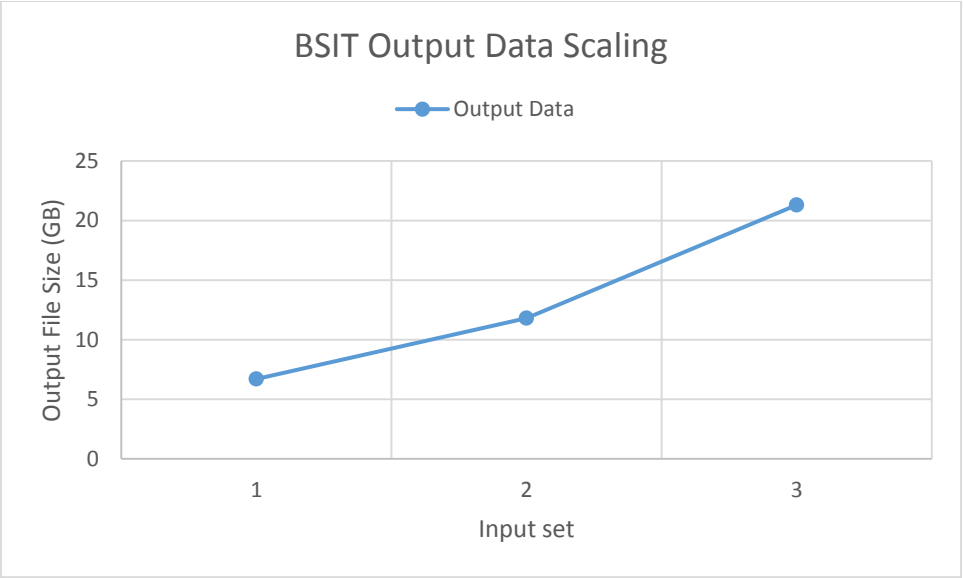

Figure 4: Output file size for three input sets for the BSIT mock-up.

The execution time of the BSIT mock-up is measured while directing the periodic output to either the locally attached NVMe SSD, the locally attached SATA SSD, or the NFS. The results are shown in Figure 5 for the three input sets described above. From this figure we can see that the execution time increases significantly with the problem size, including both computation/communication and I/O. The I/O component for this application is significant, which can be seen by the performance difference between NFS and local storage. We note that given the large I/O sizes, there is no noticeable performance difference between the local NVMe SSD and the SATA SSD.

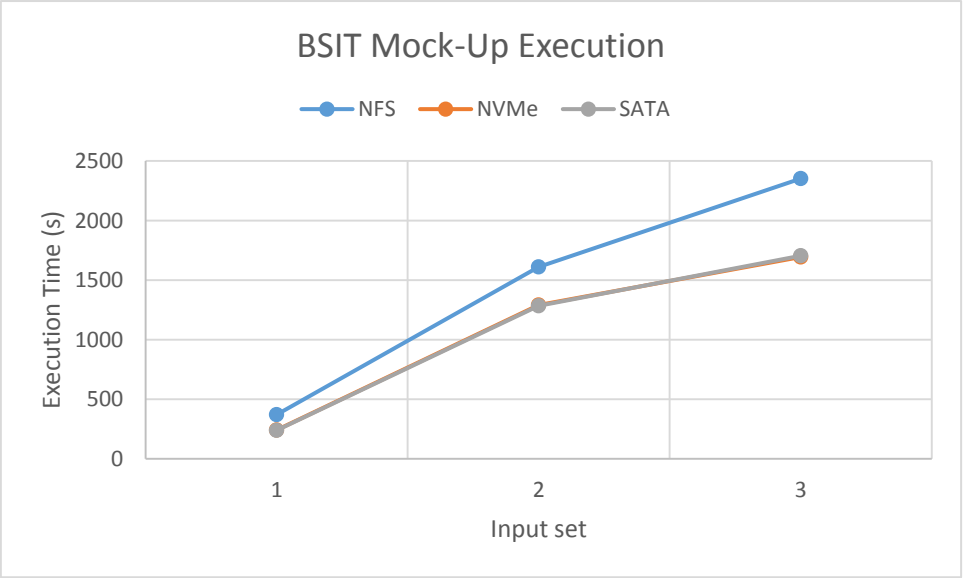

Figure 5: Execution time of BSIT mock-up with output sent to NFS, local NVMe SSD, or local SATA SSD.

Figure 6 provides another view of the data, showing the speedup of the NVMe SSD over NFS (we do not show the relative performance of the SATA SSD as it is very similar to that of the NVMe SSD). This figure shows that the benefit of local storage over NFS are in the 1.25× to 1.5× range, clearly demonstrating the benefits of local storage for this application. Moreover, the advantage appears to decline only slightly with increasing output sizes.

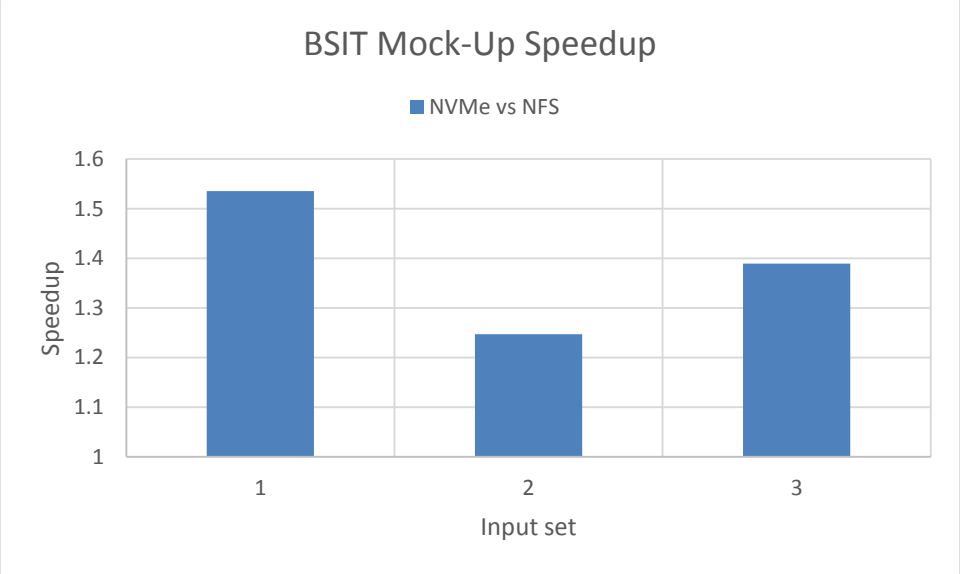

Figure 6: Speedup for BSIT mock-up of NVMe SSD over NFS.

2.3 Evaluation using the Extrae tracing tool

In addition to the two mock-ups of DEEP-ER applications described in the previous sections, we also evaluated another workload important to Exascale systems, namely performance/behaviour monitoring tools. Such tools are critical to assist program developers to achieve performance and scalability of their applications. The tools often generate large amounts of stored data that is then post-processed for analysis and visualization. We chose the Extrae tool for BSC, which is part of the Paraver/Dimemas instrumentation package and intercepts MPI, OpenMP and Pthreads events. We attached Extrae to various scientific computing benchmarks and compared, as before, the performance gains from local NVM storage versus remote storage. The analysis is divided in two parts. In the first section we evaluate Extrae with local and remote storage using some benchmarks from the Parsec suite [1] and using the standard input provided. We then evaluated Extrae using an FFT kernel, for which it is possible to vary the input size and, correspondingly, the output size.

2.3.1 Extrae evaluation with Parsec benchmarks

We first evaluated the Extrae tracing tool with a subset of the Parsec benchmarks (http://parsec.cs.princeton.edu). The experiments were done with the OpenMP versions of the benchmarks, the “native” problem sizes shipped with the benchmarks, and using 32 threads. The application codes were instrumented to link to the Extrae tool and to use the LD_PRELOAD mechanism for tracking events[1].

Figure 7 shows the size of the generated trace (line and right y-axis) and the various speedups (bars and left y-axis): speedup of no tracing versus tracing (i.e., the inverse of the slowdown introduced by the tracing tool) with trace files stored on NFS; speedup of no tracing versus tracing with trace files stored on the local NVMe SSD; the speed of tracing with files stored on the local SSD versus tracing with files stored on NFS. We do not show results for the local SATA SSD as its performance is very similar to that of the NVMe SSD. The figure shows that, as expected, tracing adds overhead to the execution time of the

[1] LD_PRELOAD is a Unix/Linux mechanism to load a desired shared object before any other library, and is often used to catch library calls for additional pre-processing, as done here by Extrae.

applications, reaching close to 30% in the case of *bodytrack* (a human body tracking workload) when output is directed to NFS. The trace files are relatively small, from just under a MB up to 40 MB. Nevertheless, directing the trace output to local storage leads to significant gains over NFS storage for two of the applications (*blackscholes* - an option pricing data analytics workload - and *bodytrack*) and cuts the overhead of tracing by more than half. With *freqmine* (a frequent itemset data analytics workload), the overhead of tracing is very small and the apparent worse performance of local NVM storage compared to NFS can be attributed to slight variations in the measurements.

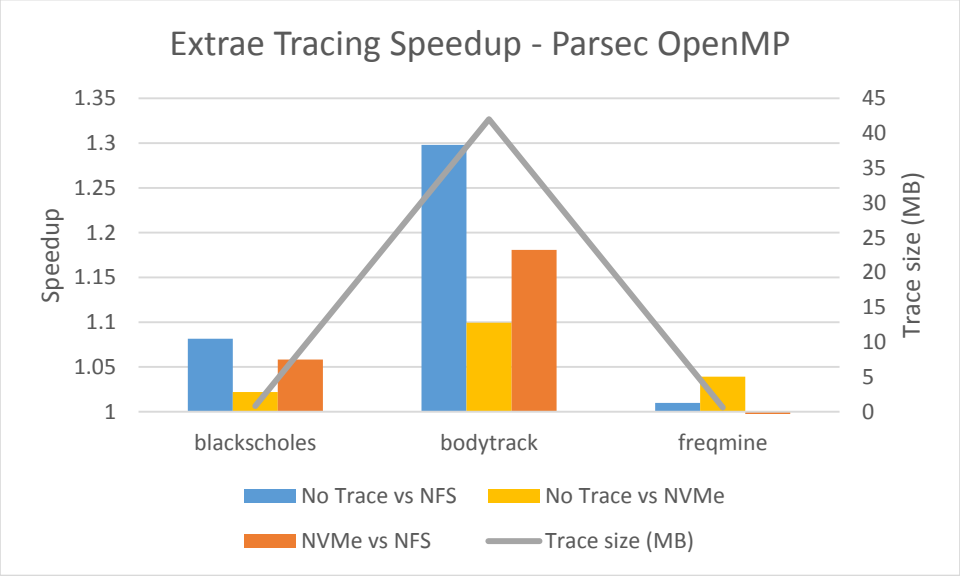

Figure 7: Trace file sizes and speedup for Extrae on three Parsec benchmarks.

2.3.2 Extrae evaluation with FFT kernel

As we were limited to the standard input sets of the Parsec benchmarks, we extended our evaluation of Extrae with a scientific kernel – FFT – for which it is straightforward to vary the input parameters. The used a public OpenMP implementation of FFT, again using the LD_PRELOAD mechanism to link to Extrae. For this evaluation we fixed the size of the array at 2^{28} and varied the number of iterations 1K to 1M. The number of threads was fixed at 32.

Figure 8 shows the various speed-ups as a function of the size of the generated trace (the size of the trace increases monotonically with the number of iterations). As before, we show speedup of no tracing versus tracing with trace files stored in NFS, the speedup of no tracing versus tracing with files stored on the local NVMe SSD, and the speedup of tracing with files stored on the local SSD versus tracing with files stored on NFS. Firstly, we can see in this figure that the size of the trace output can become very large (tens to hundreds of GB) as the running time of the application increases (with the increase in the number of iterations in this case). With the increase in trace size there is a corresponding slowdown from tracing as can be seen by the speedup of no tracing over NFS. As before, directing the output to local NVM storage leads to significant gains over NFS storage and can cut this overhead by more than half. Also for this workload we did not observe a difference in performance between the SATA SSD and the NVMe SSD.

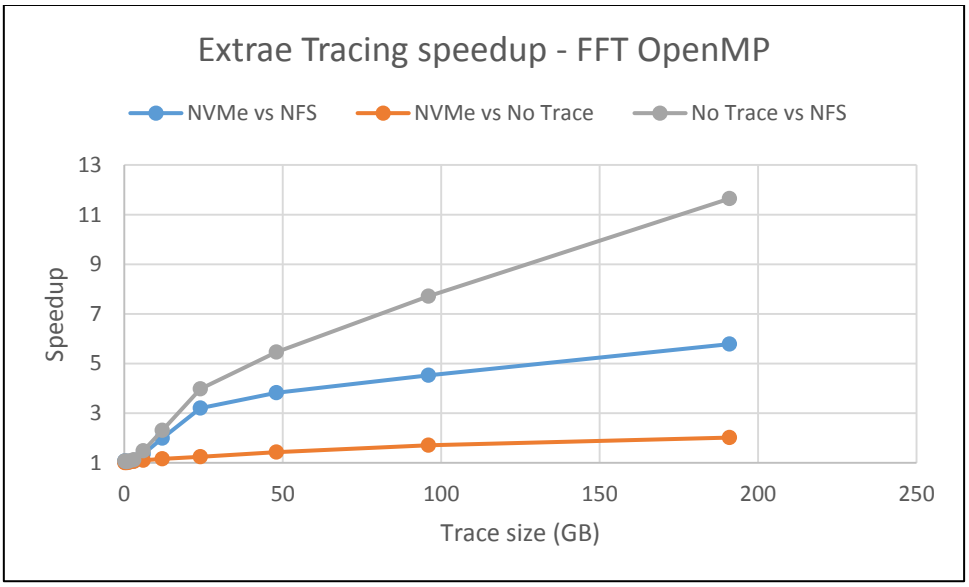

Figure 8: Trace file sizes and speedup for Extrae on FFT kernel with varying trace sizes.

3 Evaluation of node-local NVM versus GPFS in DEEP Cluster

3.1 Evaluation using collectives and micro-benchmarks

In this section we present a performance evaluation of local storage with SATA SSDs versus a HPC-grade parallel storage system (GPFS). This evaluation was done using the DEEP Cluster [1] and micro-benchmarks. It also uses a novel hint-based approach to perform data caching in local storage that was devised as part of the DEEP-ER project and is described in detail in Deliverable 4.3. In this scheme collective I/O from the application processes are collected by aggregator processes, which write the aggregated data to the filesystem. As before, the goal of the experiments is to evaluate the benefits of local storage (in this case through the caching mechanism) in DEEP-ER. A description of the hardware configuration can be found in the Annex. An extended description of these and related experiments has been submitted for publication in [3].

3.1.1 Evaluation using coll_perf benchmark

We started by evaluating storage I/O performance using the *coll_perf* (http://web.mit.edu/16.225/mpich-1.2.5.2/examples/io/coll_perf.c), which is a collective I/O synthetic benchmark distributed with the MPICH package. In this benchmark, every process writes/reads a tri-dimensional block of data to/from a distributed array that is stored in a 32 GB shared file. The experiments were done using 512 computation processes (64 nodes with 8 cores each). Computation between collective I/O is modelled as a fixed wait time of 30s, which was so chosen to ensure enough computation time (emulated) to mask background synchronization.

Figure 9 shows the speedup of write bandwidth[2] when using the local storage cache as compared to no caching. The values are shown for various configurations of buffer sizes and number of aggregators on the x-axis (buffer sizes vary from 4 MB to 64 MB and the number

[2] The speedup of write bandwidth is defined as the ratio of the improved bandwidth over the original bandwidth.

of aggregators varies from 4 to 64). From this figure we can see that for all configurations with more than 8 aggregators, there is a significant benefit from using local storage. This benefit appears to increase with more aggregators, which is expected since contention in the aggregators is reduced and the I/O bottleneck becomes more significant. Another benefit (not shown) is that the standard deviation of write bandwidth is reduced with caching and write bandwidth becomes much more stable [3].

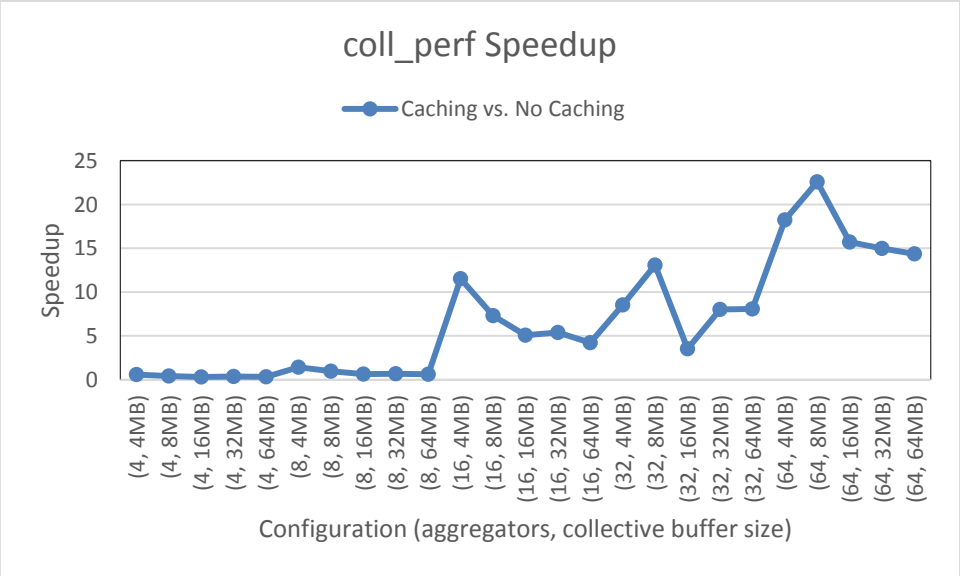

Figure 9: Speedup of coll_perf with collective I/O caching over no caching.

3.1.2 Evaluation using FLASH I/O benchmark

We next evaluated storage I/O performance using the *FLASH* I/O benchmark (http://www.ucolick.org/~zingale/flash_benchmark_io/). Like *coll_perf*, this benchmark also performs collective I/O, every process writes/reads a tri-dimensional block of data to/from a distributed array that is stored in a 32 GB shared file. The experiments were done using 512 computation processes (64 nodes with 8 cores each). Computation between collective I/O is modelled as a fixed wait time of 30s, which was so chosen to ensure enough computation time (emulated) to mask background synchronization.

Figure 10 shows the speedup of write bandwidth when using the local storage cache as compared to no caching. The values are shown for the same configurations of buffer sizes and number of aggregators as in Figure 9 As with *coll_perf* we can see a clear benefit from local storage for all configurations with sufficient aggregators (16 in this case).

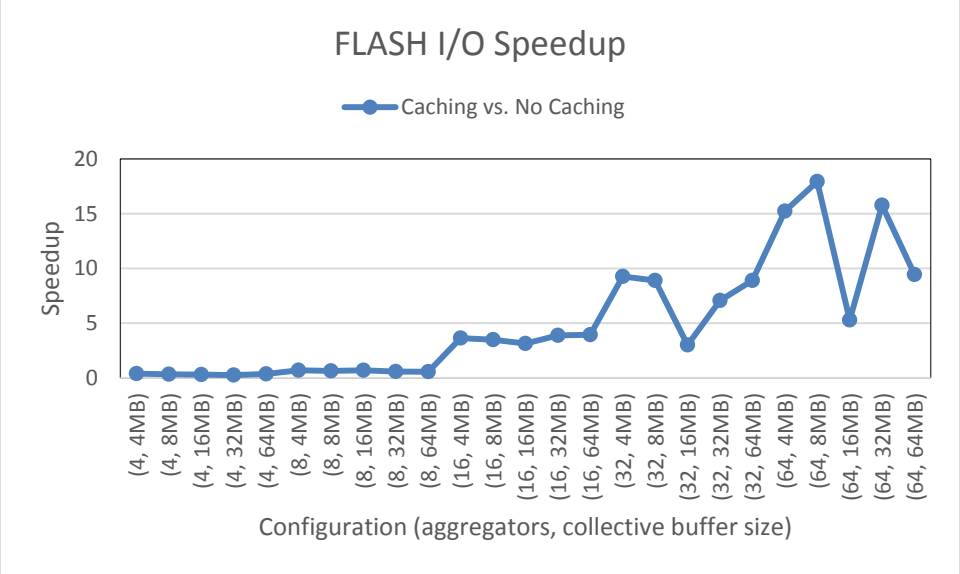

Figure 10: Speedup of FLASH I/O with collective I/O caching over no caching.

3.2 Evaluation using DEEP-ER application

In this section we present a performance evaluation of the I/O overhead in one of the DEEP-ER applications when running on multiple nodes of the DEEP Cluster [1] and performing storage I/O to the GPFS parallel storage system. The goal of this evaluation is to assess the impact that I/O has on a real application when running at a larger scale and with real inputs.

The application evaluated is the MAXW-DGTD from INRIA, which is described in detail in Deliverable 6.1. The application can be configured to optionally generate periodic checkpoints, which add to the storage I/O load. The input used was the “head” and the key parameter for the application is the number of spatial degrees of freedom (*k*), which has a direct impact on both the computation time and the size of the output. The parameters used for this evaluation are those recommended by the application developers: solver order *k*=3, final physical time attau=2.5, and 16 tasks per node (i.e., one per core). Checkpointing (when activated) is generated at every 100 iterations.

Figure 11 shows the execution time broken down into a computation/communication component and a storage I/O component. Results are shown from left to right for 4, 8, 16, 32, and 64 nodes and, for each number of nodes, with and without checkpointing. From this figure we can see that I/O time for MAXW-DGTD is negligible for all the tested number of nodes, if checkpointing is turned off. On the other hand, the I/O overhead becomes significant when checkpointing is enabled. Moreover, the relative overhead increases as the overall execution time scales down with strong scaling for higher number of nodes. This is expected as the amount of checkpoints is fixed with respect to the number of processes used. Overall, the relative overhead of storage I/O varies from 13% with 4 nodes to 68% with 64 nodes.

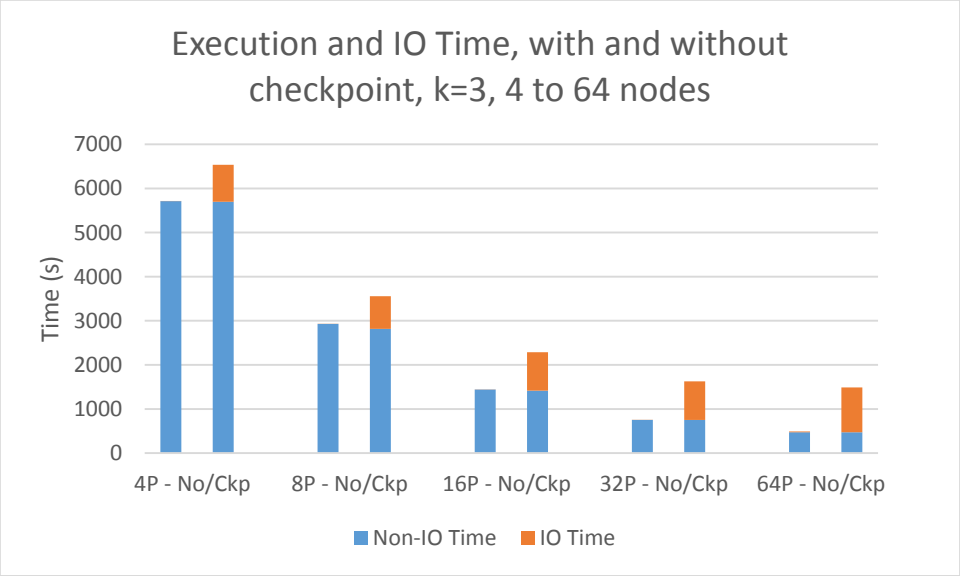

Figure 11: Execution time of MAXW-DGTD with and without checkpoint.

In order to assess the sensitivity of the I/O time with checkpointing frequency, we also varied this parameter to 50, 500, and 1000 (in addition to the initial value of 100 in the previous experiment). Figure 12 shows the computation/communication and I/O components as a function of the checkpointing frequency. From this figure we see that, as expected, the amount of I/O is a direct function of the checkpoint frequency. This means that lower I/O overhead can be achieved, but at the expense of decreased resiliency, which might compromise productivity in future very large scale systems. A more thorough analysis of this trade-off is ongoing as part of WP5.

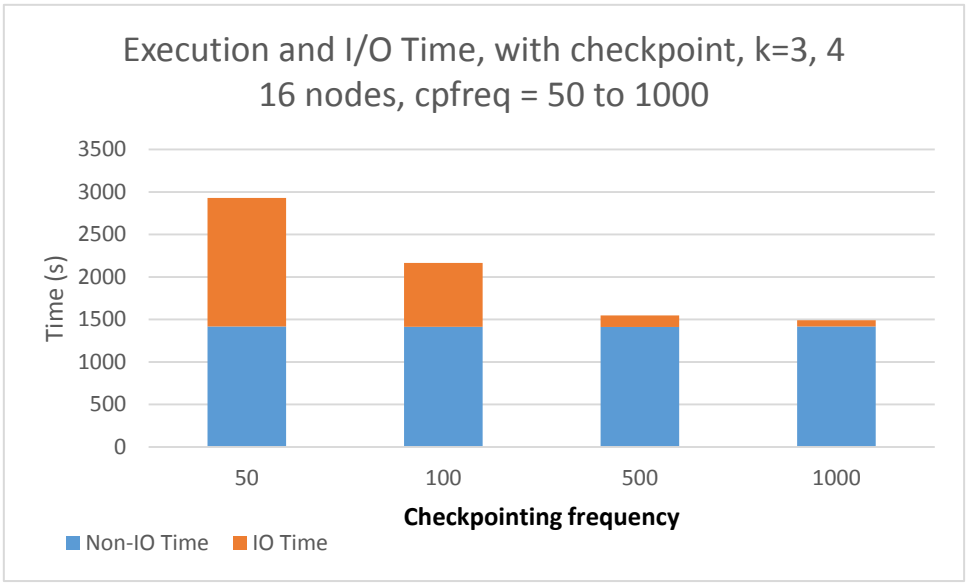

Figure 12: Execution time of MAXW-DGTD for various checkpointing frequencies.

Overall, from these early results we can anticipate that the use of local NVM storage could provide significant benefits for this DEEP-ER application.

4 Evaluation of node-local NVM versus intermediate storage in DEEP-ER SDV[3]

In this section we continue the analysis from the previous section using MAXW-DGTD, but now on the recently available DEEP-ER SDV. In these experiments the application is configured to perform storage I/O either to node-local NVM devices or to intermediate storage in the storage server. The goal of this evaluation is thus to quantify the benefits of using local I/O storage for a real application, which is one of the key innovations and contributions of this project. The local NVM storage is managed by the BeeOND ("BeeGFS on demand") variant of BeeGFS developed as part of this project.

As before, MAXW-DGTD was configured with and without periodic checkpointing enabled. The input was "head" and *k* was set to 3. Given the higher number of cores per socket in the DEEP-ER SDV, the number of tasks per node was set to 48. In order to evaluate realistic checkpointing scenarios in current and future large-scale systems, we set the checkpoint frequency to every 2500, 5000, and 10000 iterations. Since all tasks in the application each execute 34380 iterations, this means that checkpointing is performed approximately 16, 8, and 4 times, respectively. In other words, for a 2h execution time without checkpoint, the application will checkpoint every 7.5min, 15min, and 30min for the checkpoint frequencies of 2500, 5000, and 10000 iterations, respectively. Put it another way, the maximum amount of work lost upon a failure will be 1/16, 1/8, and 1/4 of total work. Our experience shows that these are reasonable expectations even for future larger systems.

Figure 13 shows the same execution time breakdown into storage I/O and non-I/O components. The top chart shows results without checkpointing and the bottom chart shows results with checkpointing. With checkpointing the frequency was set to every 10000 iterations, which is our lowest chosen frequency. Results are shown from left to right for 4, 8, and 16 nodes (the maximum size of the SDV). For each number of nodes the left bar shows time for storage I/O directed to the BeeGFS file system running on the storage server (i.e., intermediate storage) and the right bar shows time for storage I/O directed to the BeeOND file system running on each local NVM SSD. On the right axis of the charts we plot (red diamond markers) the improvement observed when moving storage from the servers to the local NVM devices. From this figure we can see that even though I/O overhead is small when checkpointing is turned off, there is still a benefit from using local storage (up to 8% for 16 nodes). The benefit of local storage is significantly amplified when checkpointing is turned on, with the relative overhead of storage I/O varying from 14% with 4 nodes to 37% with 16 nodes.

[3] This section contains new material added since the original submission of this document.

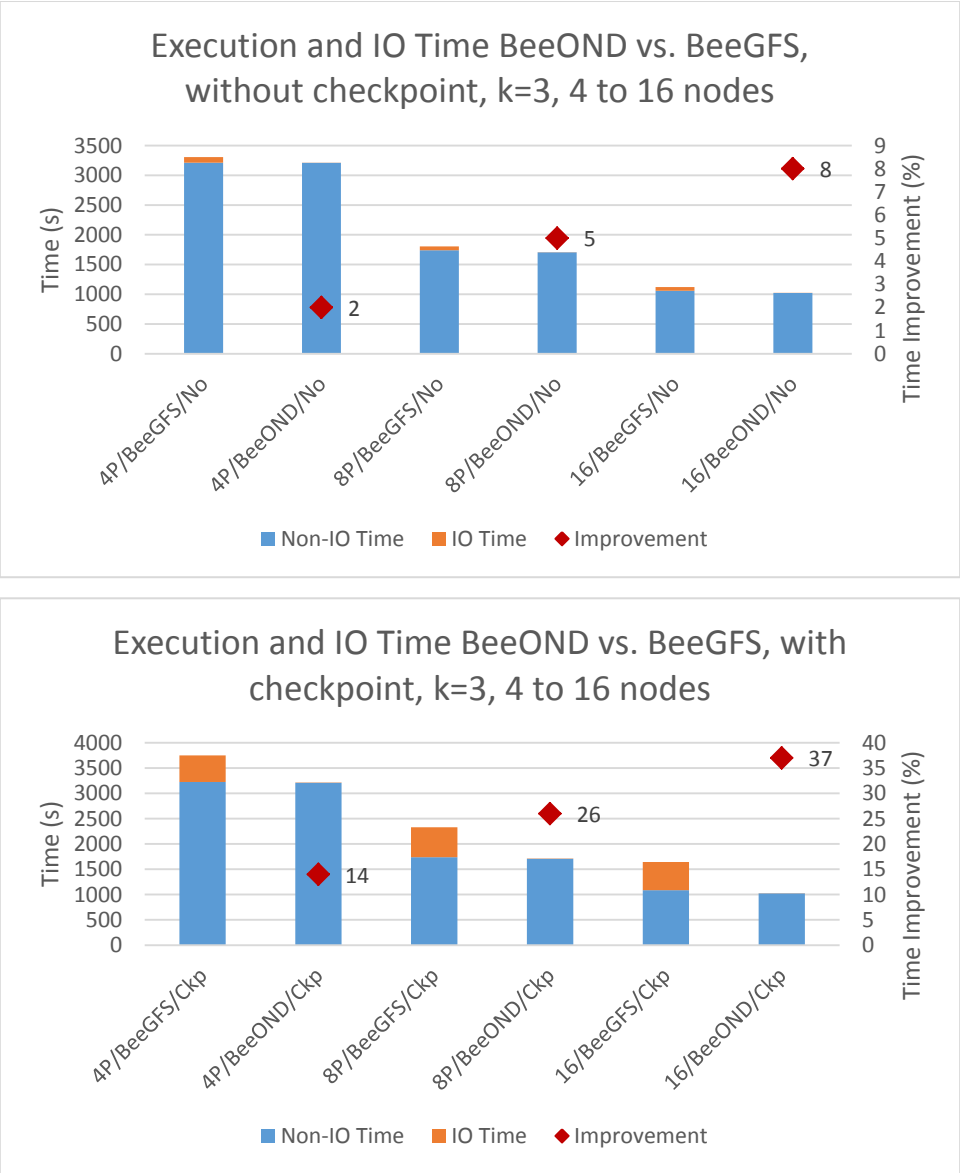

Figure 13: Execution time of MAXW-DGTD with and without checkpointing and with local and intermediate storage.

In order to assess the sensitivity of the I/O time with checkpointing frequency, we varied the frequency as described earlier while fixing the number of nodes to 16. Figure 14 shows the storage I/O and non-I/O components as a function of checkpoint frequency (every 10K, 5K, or 2.5K iterations). From this figure we see that the amount of I/O is, as before, a direct function of checkpoint frequency. However, local storage is so successful at reducing the overhead of I/O that even at high checkpoint frequencies the relative overhead is negligible (the maximum relative overhead of I/O with local storage increases only from 4% to 5%). Comparing the total execution time with local storage and intermediate storage shows that the first leads to improvements ranging from 37% for checkpoint frequency of every 10K iterations to 52% and 69% for checkpoint frequencies of every 5K and 2.5K iterations.

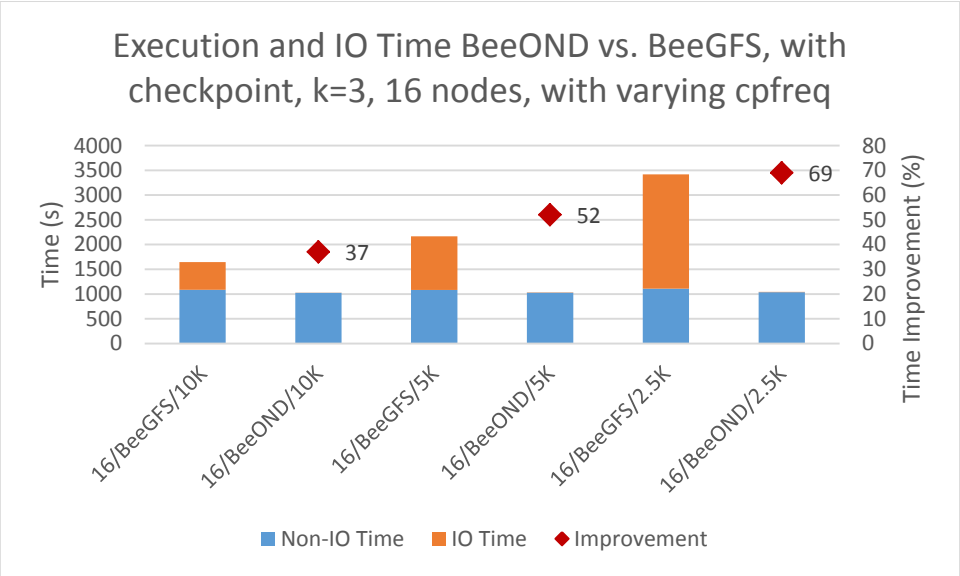

Figure 14: Execution time of MAXW-DGTD for various checkpointing frequencies and with local and intermediate storage.

5 Summary and Conclusions

This deliverable presents a first quantitative assessment of the benefits of local NVM storage in the scope of DEEP-ER. Results with a single-node system with the latest NVMe SSD show that significant advantage can be expected when using such local storage in comparison to only remote storage. It also corroborates the often observed superiority of the recent NVMe devices over current SATA devices [4] and validates our preference for such state-of-the-art technology for the DEEP-ER prototype. Additionally, results with the DEEP Cluster also show significant advantage of local storage in comparison to even a HPC-grade parallel storage system. Finally, results with one of the DEEP-ER applications running on this system show that storage I/O can lead to significant overheads, which gives the techniques being developed in the DEEP-ER project much room to improve performance.

Overall, the assessment in this deliverable validates our approach of using node-local NVM storage to improve the overall performance of the DEEP-ER applications. This assessment will continue with more DEEP-ER applications, which will be run on the DEEP Cluster and the Software Development Vehicle (SDV) recently available.

6 Bibliography

[1] DEEP Deliverable 6.3, available at http://www.deep-project.eu/deep-project/EN/Project/Deliverables/_node.html

[2] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh and Kai Li. “The PARSEC benchmark suite: characterization and architectural implications”. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008.

[3] Giuseppe Congiu, Sai Narasimhamurthy and Andre Brinkmann. “Improving Collective I/O Performance Using Non-Volatile Memory Devices”. Submitted to CCGRID 2016.

[4] NVM Express, http://www.nvmexpress.org

Annex A

The evaluation described in this document was performed in three different systems installed at the JSC. The first system, called KNC2, is a single-node machine equipped with early engineering samples (donated by Intel) of Intel's NVMe SSD. This system contains a "Sandy Bridge" processor with 2 sockets, 8 cores per socket, running at 2.6 GHz. The system is equipped with 64 GB of main memory and runs a version of Cent OS 6.4 that was patched to support the NVMe driver. The NVMe SSD is a "Fultondale" NAND device with 400GB of storage.

The second system used in this evaluation is the DEEP Cluster. This system consists of 128 nodes with dual-socket "Sandy Bridge" processors for a total of 2048 cores. Each node contains 32GB of main memory. The compute cluster is connected to a set of 6 storage servers that consist of dual-socket quad-core Xeon processors with 32 GB of main memory. The storage servers run the BeeGFS file system developed by the DEEP-ER partner Fraunhofer Institute (FHG-ITWM). The storage consists of a JBOD with 45 2TB SAS disks connected to the servers through a SAS switch using two 4x ports running at 6Gb/s, for a total of four 8+2 RAID6 storage targets and 2 RAID1 targets for metadata and management data (1 drive is left as spare). Of the six storage servers, one is dedicated for metadata, one is dedicated to management, and the remaining four are used as data servers. In addition to the storage servers, each compute node is equipped with an 80GB SATA SSD, which contains a 50 GB partition dedicated for general purpose storage. It is these SSDs that we use to evaluate the benefits of local storage. Finally, the compute nodes are connected through an InfiniBand QDR network.

Figure 15: DEEP Cluster at JUELICH

The main characteristics of the DEEP Cluster installed at JUELICH are summarised here.

Node type	Eurotech Aurora nodes
Number of nodes	128
Processors per node	2x Intel Xeon E5-2680 (Sandy Bridge)
Cores per node	2x 8 cores (@2.7 GHz)
Threads per node	using hyper-threading: 32 threads/node

Total number of cores	2048 CPU cores
Memory per node	32 GB
Total memory	4 TB
Peak performance	44 Teraflops
Interconnect network	InfiniBand 4x QDR 40 Gbps
Filesystem	GPFS from JUST cluster exported through NFS to compute nodes FhGFS as a parallel scratch.
Operating system	CentOS 6.5
Batch system	Torque + Moab
MPI runtime	ParaStation MPI
Compilers	Intel Professional Fortran, C + C++

The third system is the DEEP-ER SDV. This system consists of 16 nodes with dual-socket “Haswell” processors for a total of 384 cores. Each node contains 128 GB of main memory. The compute cluster is connected to a set of 3 storage servers. One of the servers is dedicated for metadata. It consists of a dual-socket six-core Xeon processor with 32 GB of main memory and is equipped with two 200GB SATA SSDs in RAID1 configuration. The other two servers consist of quad-socket six-core Xeon processors with 100 GB of main memory. They are connected through a SAS switch using 4x ports running at 8Gb/s to 24 6TB SAS disks. The storage servers run the BeeGFS file system developed by the DEEP-ER partner Fraunhofer Institute (FHG-ITWM). In addition to this storage servers, each compute node in the SDV is equipped with a 400GB “Fultondale” NVMe SSD. Direct node-local access to the NVMe SSD is possible either through a local file system mount point or through the dynamic BeeGFS on demand (BeeOND) service. The first is shared by jobs in the node and is periodically cleared by the system. BeeOND creates a BeeGFS instance on the fly and is used to provide a shared parallel filesystem on a "per job basis" across all compute nodes of a job. The compute nodes are connected through the network developed by the DEEP-ER partner EXTOLL.

List of Acronyms and Abbreviations

A

API: Application Programming Interface
Aurora: The name of Eurotech's cluster systems

B

BADW-LRZ: Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften. Computing Centre, Garching, Germany
BeeGFS: The Fraunhofer Parallel Cluster File System (previously acronym FhGFS). A high-performance parallel file system to be adapted to the extended DEEP Architecture and optimised for the DEEP-ER Prototype.

BN: Booster Node (functional entity); refers to a self-booting KNL board (Node board architecture) including the NVM and NIC devices connected by PCI Express or a Brick (Brick architecture).
BoP: Board of Partners for the DEEP-ER project
Brick: Modular entity forming a Booster Node in the Brick Architecture, composed of Host modules, NVMe and NIC devices all connected by an PCI Express switch.
Brick Architecture: Two-level hierarchical architecture for the DEEP-ER Booster, based on the "Brick" element as a Booster node.
Brick Module: Smallest functional HW entity. Up to 6 modules are aggregated into a Brick
BSC: Barcelona Supercomputing Centre, Spain
CN: Cluster Node (functional entity)

D

DEEP: Dynamical Exascale Entry Platform
DEEP-ER: DEEP Extended Reach: this project
DEEP-ER Booster: Booster part of the DEEP-ER Prototype, consisting of all Booster nodes and the NAM devices.
DEEP-ER Global Network: High performance network connecting Bricks, CN, NAM and other global resources to form the DEEP-ER prototype system
DEEP-ER Interconnect: High performance network connecting the Booster and Cluster nodes, the NAM and service nodes with each other to form the DEEP-ER Prototype.
DEEP-ER Network: High performance network connecting the DEEP-ER BN, CN and NAM; to be selected off the shelf at the start of DEEP-ER
DEEP-ER Prototype: Demonstrator system for the extended DEEP Architecture, based on second generation Intel® Xeon PhiTM CPUs, connecting BN and CN via a single, uniform network and introducing NVM and NAM resources for parallel I/O and multi-level checkpointing

DEEP Architecture: Functional architecture of DEEP (e.g. concept of an integrated Cluster Booster Architecture), to be extended in the DEEP-ER project

DEEP System: The prototype machine based on the DEEP Architecture developed and installed by the DEEP project

E

EC: European Commission

EU: European Union

Eurotech: Eurotech S.p.A., Amaro, Italy

Exascale: Computer systems or Applications, which are able to run with a performance above 10^{18} Floating point operations per second

I

Intel: Intel Germany GmbH Feldkirchen,

iPic3D: Programming code developed by the University of Leuven to simulate space weather

I/O: Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation

J

JUDGE: Juelich Dedicated GPU Environment: A cluster at the Juelich Supercomputing Centre

JUELICH: Forschungszentrum Jülich GmbH, Jülich, Germany

K

KNC: Knights Corner, Code name of a processor based on the MIC architecture. Its commercial name is Intel® Xeon PhiTM.

KULeuven: Katholieke Universiteit Leuven, Belgium

N

NVM: Non-Volatile Memory. Used to describe a physical technology or the use of such technology in a non-block-oriented way in a computer system

NVMe: Short form of NVM-Express

NVM-Express: An interface standard to attach NVM to a computer system. Based on PCI Express it also standardises high level HW interfaces like queues.

O

OmpSs: BSC's Superscalar (Ss) for OpenMP

OpenMP: Open Multi-Processing, Application programming interface that support multiplatform shared memory multiprocessing

OS: Operating System

P

Paraver: Performance analysis tool developed by BSC

PCI: Peripheral Component Interconnect, Computer bus for attaching hardware devices in a computer

PCIe: Short form of PCI Express

PCI Express: Peripheral Component Interconnect Express started as an option for a physical layer of PCI using high-performance serial communication. It is today's standard interface for communication with add-on cards and on-board devices, and makes inroads into coupling of host systems. PCI Express has taken over specifications of higher layers from the PCI baseline specification.

S

SDV: Software Development Vehicle: a HW system to develop software in the time frame where the DEEP-ER Prototype is not yet available.

SSD: Solid State Disk