SEVENTH FRAMEWORK PROGRAMME

FP7-ICT-2013-10

DEEP-ER

DEEP Extended Reach

Grant Agreement Number: 610476

D7.2

Report on projections and improvements for the DEEP/DEEP-ER concept

Approved

Version: 2.0

Author(s): J.Schmidt (UHEI)

Contributor(s): H.Ch.Hoppe (Intel), J.Gimenez (BSC), V.Beltran (BSC), G.Congiu (Seagate), N.Eicker (JUELICH), C.Clauss (ParTec), A.Galonska(JUELICH)

Date: 04.05.2017

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="6">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: D7.2</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td rowspan="2">Deliverable Level:
PU*</td><td>Contractual Date of Delivery:
31 / March / 2017</td></tr>
<tr><td>Actual Date of Delivery:
31 / March / 2017</td></tr>
<tr><td colspan="2">EC Project Officer: Juan Pelegrín</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="6">Document</td><td colspan="2">Title: Report on projections and improvements for the DEEP/DEEP-ER concept</td></tr>
<tr><td colspan="2">ID: D7.2</td></tr>
<tr><td>Version: 2.0</td><td>Status: Approved</td></tr>
<tr><td colspan="2">Available at: http://www.deep-er.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word</td></tr>
<tr><td colspan="2">File(s): DEEP-ER_D7.2_Report_on_projections_and_improvements_for_the_DEEP_DEEP ER_concept_v2.0-ECapproved</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>J.Schmidt (UHEI)</td></tr>
<tr><td>Contributors:</td><td>H.Ch.Hoppe (Intel), J.Gimenez (BSC), V.Beltran (BSC), G.Congiu (Seagate), N.Eicker (JUELICH), C.Clauss (ParTec), A.Galonska(JUELICH)</td></tr>
<tr><td>Reviewed by:</td><td></td></tr>
<tr><td>Approved by:</td><td>BoP/PMT</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
1.0	31/March/2017	Final	EC submission
2.0	04/May/2017	Approved	EC approved

Document Keywords

Keywords:	DEEP-ER, HPC, Exascale, design, extrapolation, modelling

Table of Contents

List of Figures

Executive Summary

This document discusses how to scale the DEEP-ER prototype up to Exascale levels and identifies the challenges involved in doing so. It draws on results of the DEEP-ER work packages 3 to 6 and 8, presents a model for analysing compute and I/O performance, and discusses the scaling of performance, energy and reliability. To provide guidance to European R&D efforts, the document also sketches the technology evolution as apparent at the time of writing, and puts the challenges identified into perspective. The unavailability of the DEEP-ER Booster at the time of writing the Deliverable has made meaningful extrapolations to large scale systems impossible.

1 Introduction

The performance of supercomputer systems is steadily increasing, as shown by the sequence of Linpack results in the TOP500 lists. The growth of system performance according to that list has seen a certain slowdown compared to historical rates, yet it is clear that the Exascale performance level will be reached in the foreseeable future[1].

On the one hand, it is obvious that entering the Exascale era will require a significant increase in performance compared to current systems. On the other hand, efficiency on a system level has become a major concern since these systems will have to operate within a reasonable power budget. This applies to individual components and how user applications are programmed likewise. Therefore, future components and systems have to be properly architected, while software developers must strive to optimize their applications to run optimally on a given architecture. It is, however, inevitable that system and component design is conducted through a tightly coupled co-design process between these two groups to maximize performance and power efficiency. It has to be stressed that the ultimate objective is to deliver top performance and efficiency for the relevant scientific and industrial applications, rather than for “just” dense linear algebra (as measured by Linpack). Taking in and accommodating such application’s requirements at the start of the HW/SW co-design is of paramount importance.

In addition to performance and energy efficiency, a third important criterion is resilience. As systems grow in size, more and more components are added and each of these components is subject to hard or soft errors according to certain per-component rates. For systems with 100000s to several millions of nodes, failures somewhere in the system will therefore regularly occur, and operation and scientific output of these machines have to be protected against lengthy outages or the need for costly re-compute of lost system state. To avoid restarting an application from scratch upon a failure, techniques to regularly capture and store the current system status have been introduced. These techniques, however, can have a notable impact on execution time and it is mandatory to understand how they will scale and impact larger systems.

We therefore analysed and measured these three criteria mentioned on the DEEP-ER SDV, with the full DEEP-ER prototype not becoming available at the time of writing. Using this data, we estimate their scaling behaviour and furthermore outline the expected technological evolution of individual system components.

This report is structured into two major parts: first, we summarize the performance, energy, and reliability modelling approaches applied in DEEP-ER and conclude by analysing the challenges associated with further scaling of the DEEP-ER architecture. The second section reflects on the likely technology progress with regards to potential future DEEP-ER systems and the fundamental requirements that have shaped the DEEP-ER architecture.

[1] The Chinese Sunway TaihuLight system clearly shows one way to reach Exascale, albeit with a point design for dense linear algebra. More general HPC systems are farther away from the 10^{18} Flop/s goal at this point.

2 Modeling and Extrapolation

The basic approach taken by DEEP-ER for modelling and extrapolation was described at length in the previous Deliverable D7.1 [1], and a publication on the method can be found at [2] and [3]. In short, the method relies on obtaining execution traces from running an application on a set of system configurations. These traces contain event information on all the MPI communication, I/O operations (as discussed in D7.1), calls to application routines and potentially performance counter information (such as instructions per cycle). For each configuration (number of MPI ranks), three factors that determine the parallel efficiency of that application run are calculated:

- Load balance (LB), measures the efficiency losses due to differences on the computing time between processes. If some processes take more time in the computation, the other processes would have to wait for them in the synchronization of the MPI calls for instance.
- Transfer, measures the reductions on the efficiency due to the need to transfer data between processes. If the application is dominated by communications, the transfer efficiency would be low.
- Serialisation (µLB), measures the efficiency losses due to dependencies during the execution. This factor also reflects load imbalances that can be compensated along time. If on the even iterations half of the processes do more work than the other half but on the odd iterations the behaviour is just the opposite, the load balance would report a good efficiency while the serialisation would reflect the loss of efficiency.

The parallel efficiency is the product of these three factors. Since we have a series of application runs (using weak or strong scaling), each of the three factors is fitted with an appropriate model, like for instance the Amdahl function

$$Amdahl_{fit} = \frac{metric_0}{f_{metric} + (1 - f_{metric}) * P}$$

From the fitted model functions, predictions can be made for larger configurations, as for instance those contained in D7.2 or [3]. Another important capability is that certain machine characteristics (such as MPI latency and bandwidth parameter, network contention behaviour, relative CPU speed) can be changed and the application execution replayed with these new parameters. This allows “what-if” studies for non-existing systems.

In the following sections, we discuss the progress in applying such a technique to I/O (which was requested by the M12 review), power use/efficiency (likewise), and resiliency (as promised in the DoW).

The delays in making the DEEP-ER Booster system available have severely impacted the modelling and extrapolation work. The only representative system with local, fast I/O available was the DEEP-ER SDV, which only goes up to 16 Intel® Xeon® and/or 8 Intel® Xeon Phi™ nodes. From such a shallow range of measurements, meaningful extrapolation is not possible. Use of the larger QPACE3 system (which has up to 352 KNL nodes, Intel® Omni-Path Architecture interconnect and BeeGFS file system and is in its bring-up phase) was tried, yet instability of that system prevented traces to be taken.

2.1 Performance Modeling and Extrapolation

Following the recommendation of the reviewers, the objective of the performance modeling and extrapolation activity has focused on including I/O efficiency in the analysis. It was clear

that a detailed modeling of I/O operations, including transmission over the network and processing by I/O nodes, would be out of scope of the DEEP-ER project and not possible within the effort allocated to task 7.1. In light of this, different strategies have been considered and evaluated.

The first concept, presented at the last review did foresee to compute the I/O efficiency as an additional factor in the BSC multiplicative efficiency model. When implementing this approach, problems and limitations became apparent. The most important issue was that the order in which the efficiency factors are isolated would determine their value, leading to different conclusions. This is ultimately caused by the fact that I/O operations (which are recorded in the trace) do influence both compute and communication, and cannot be, without a much more detailed model, be separate out from these.

For that reason, we were forced to reconsider how to isolate and measure the I/O efficiency, as well as enable its extrapolation to larger scales or to a different platform than the one used to collect the traces. Maintaining the same goal, we decided not to include I/O as a component of the efficiency model but to define a methodology to measure the I/O efficiency and diagnose its impact on the efficiency of the application amongst the different components of the BSC model. The impact on the different efficiency factors is obtained comparing the real scenario with I/O and an estimator of the scenario without I/O. This comparison gives us hints about how the I/O is affecting the efficiency of the application execution.

To facilitate the evaluation of different alternatives, we defined a set of very small synthetic test cases that we could intuitively diagnose. These test cases included scenarios with balanced and unbalanced I/O operations having different impacts on the global load balance and the serialization of the run. Analyzing these synthetic test cases, we finally have selected two different approaches. The first one uses the Dimemas simulator (which replays an execution captured in a trace with different system parameters) to model the execution as if no I/O operations were occurring. Dimemas may introduce small errors, since it does not model details of the MPI progress engine, and to limit these, we also evaluate the I/O impact assuming that the time spent in MPI routines will not be significantly modified when the I/O is eliminated. This scenario may be realistic when the percentage of I/O is small or at least is concentrated in only a few regions of the execution.

Both approaches have a common first step that measures the I/O weight. Based on a profile measurement, we can split the execution time into three parts: MPI, computation and I/O. Using that profile, we can consider the time efficiency of I/O as the percentage of time outside I/O calls.

$$I/O\ global efficiency = \frac{(Total time - I/O\ time)}{Total time}$$

As the I/O time is basically increasing the time outside the parallel runtimes (OpenMP/OmpSs and MPI), we can also measure how the I/O impacts the computational efficiency without including the MPI time.

$$I/O\ computational efficiency = \frac{Computing time}{(Computing time + I/O\ time)}$$

These metrics are based on time, but in this step we can also characterize the I/O by the volume of data and the number of calls. Correlating these metrics can help us for the extrapolation for larger scales.

The second step is to compute the efficiencies for both the real run that included I/O and an approximation of the execution without I/O. For this step we use the two approaches previously described. To evaluate the impact of the I/O on the different factors of the BSC efficiency model, we compute the ratio between the two scenarios. As an example for Load Balance:

$$RatioLoadBalance = \frac{LoadBalanceeff.with\,I/O}{LoadBalanceeff.without\,I/O}$$

If the ratio is bigger than 1, it means executing the I/O somehow improved the efficiency factor. For instance, if the I/O is unbalanced and the application is also unbalanced, the I/O may compensate the imbalance improving the global balance.

If the ratio is smaller than 1, it means executing the I/O has a negative impact on the efficiency factor. In the previous example, if the imbalance of I/O is correlated with the computation imbalance (the processes that do more computation also do more I/O) then the global balance of the application decreases due to the I/O.

Finally, if the ratio is equal to 1, it means the I/O had no impact on the corresponding efficiency factor.

2.1.1 I/O modeling applied to xPic

We applied this methodology to the xPic application, which is a combined particle and field simulation for space weather. Traces were obtained with 2,4, 8 and 16 MPI ranks on the DEEP-ER SDV Cluster (Xeon) nodes. Reason for not using the Booster (KNL) nodes was that the scale would have been even lower (only 8 nodes available). xPic was used in strong scaling mode, with 32768 cells in the X dimension, one cell in the Y and Z dimensions, and 512 particles per cell. Field I/O was performed every 100 iterations, and the particles were written out every 5000 iterations. xPic was run for a total of 10000 iterations.

The below figures show the results of the I/O analysis. First we analyzed the I/O time efficiency with respect to the full execution and with respect to its impact on the computational efficiency. As we can see in Figure 1, the I/O represents around 20% of the application time having a small decrease with the scale, except when comparing the first two runs with 2 and 4 processes where the efficiency increases reporting a reduction on the I/O time. The I/O compute efficiency drops more noticeably.

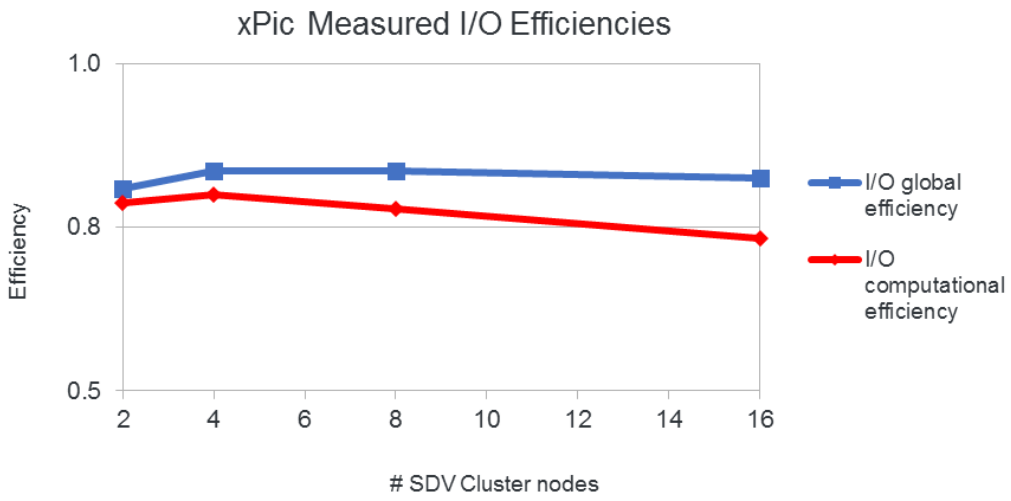


Figure 1: xPic I/O time efficiencies

The next two figures characterize the I/O with respect to the number of calls (Figure 2) and the volume of data (Figure 3).

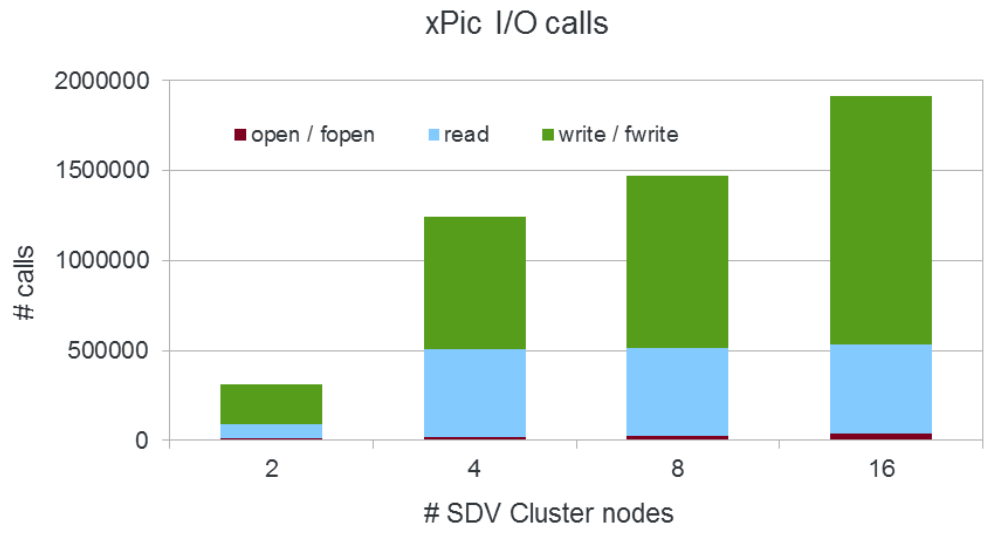


Figure 2: xPic I/O calls

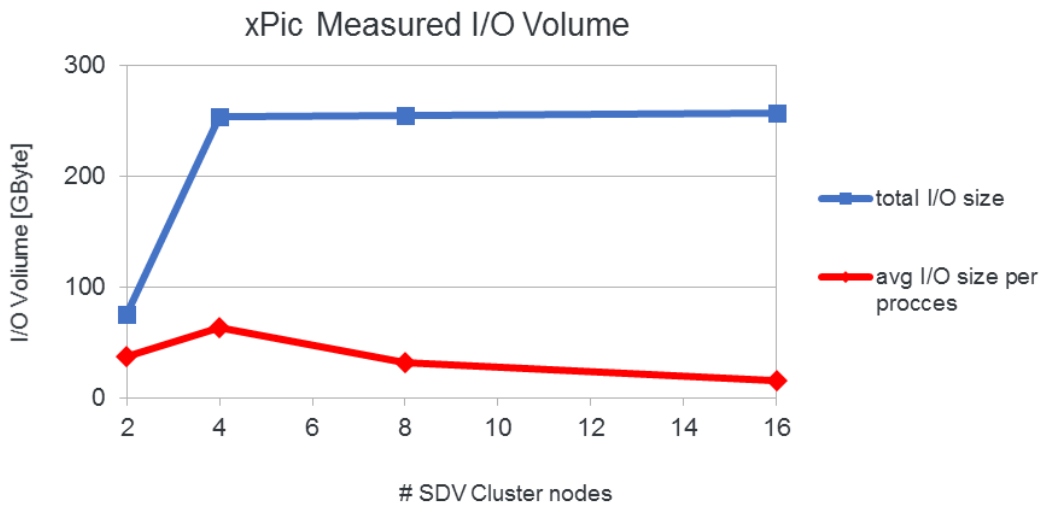


Figure 3: xPic I/O size

Again we can see a significant different behavior with the run on 2 cores. Comparing the other runs we can see the number of read calls is maintained while the number of write/fwrite calls increases with the scale. Looking at the volume of data we can see that the total volume of data is constant, as would be expected from strong scaling, and the average I/O volume per process goes down.

We will ignore the run on 2 nodes, since it behaves very differently, and we suspect a problem with this particular run. This first analysis step did identify that I/O has an important impact on the execution and with its current configuration it would limit the scaling due to the significant increase in the number of calls with the scale.

This analysis also indicates that the approach that uses Dimemas may be a better estimate of the scenario without I/O due to the large number of I/O calls and the relevant percentage of time it represents. Nevertheless, we used both the approach with Dimemas replaying execution and changing MPI times and the one that keeps the MPI times constant. Like in the synthetic traces, both methods gave the same diagnoses despite a small variability on the metrics.

Figure 4 through Figure 6 plot the evolution of the efficiency factors for the instrumented run and the two approaches for the estimation of the run without I/O. Analysing first the efficiency of the instrumented run (which includes the I/O operations) in Figure 4, we can see the main loss of efficiency is highly correlated with the global load balance. Serialization (efficiency of the application parallelization) and transfer (efficiency associated to the message transfer) show good values on the range of processes analysed.

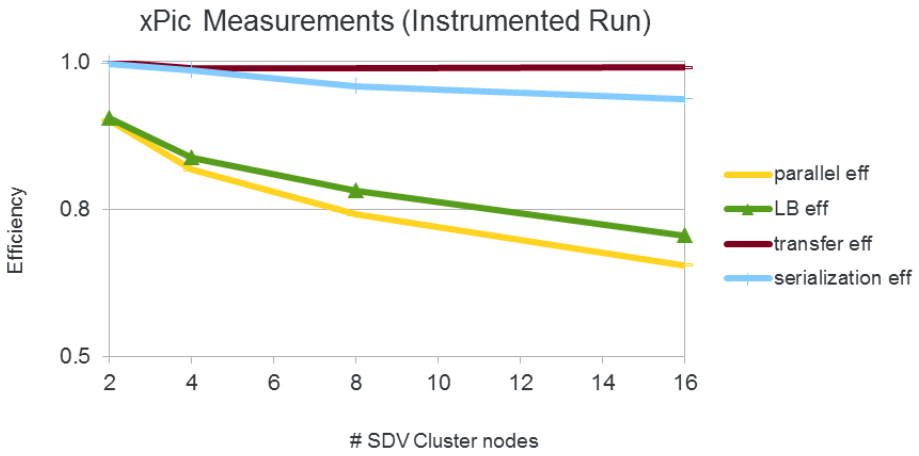


Figure 4: Measured xPic efficiencies.

Comparing the instrumented run with respect to the Dimemas prediction for a run without I/O as shown in Figure 5, we can see that the parallel efficiency would even improve a little bit without I/O due to an improvement in the global load balance. On the other hand, both serialization and transfer efficiencies will be a little bit worse. In this scenario the main loss of efficiency is still highly correlated with the application load balance.

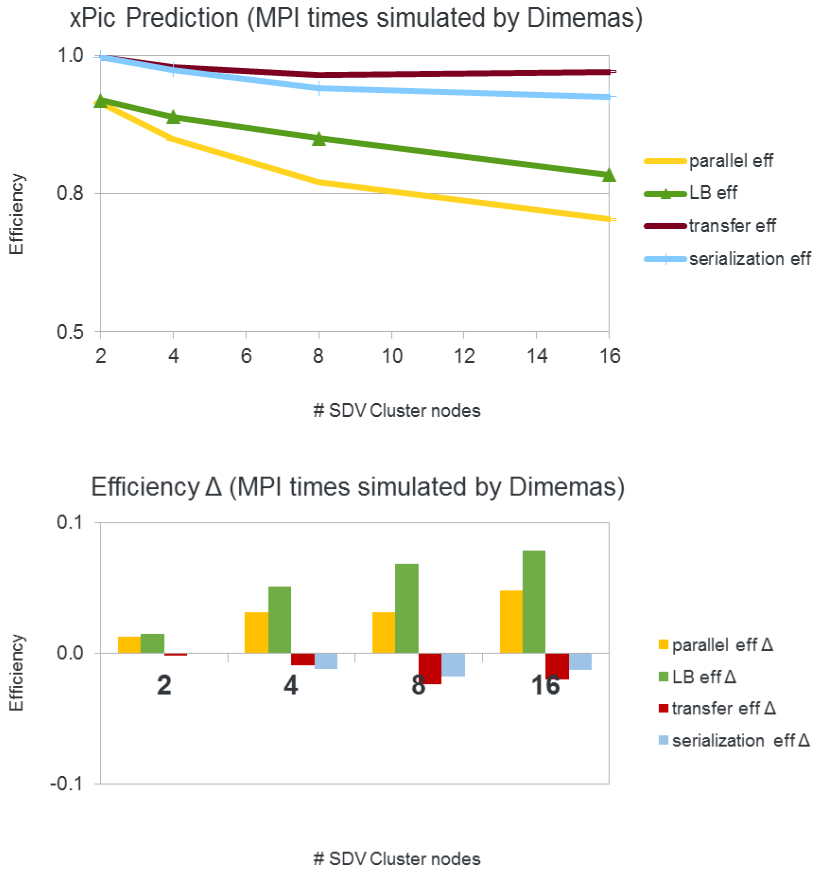


Figure 5: Efficiencies (absolute and delta to measured) for a xPic run without I/O (MPI times simulated by Dimemas)

Despite that we already know that this for xPic this would not be a good estimator, we can compare the prediction of a run without I/O considering without impact on the MPI time with the two previous scenarios. In this case, the parallel efficiency is reduced without the I/O and both global load balance and data transfer are highly correlated with the parallel efficiency reduction.

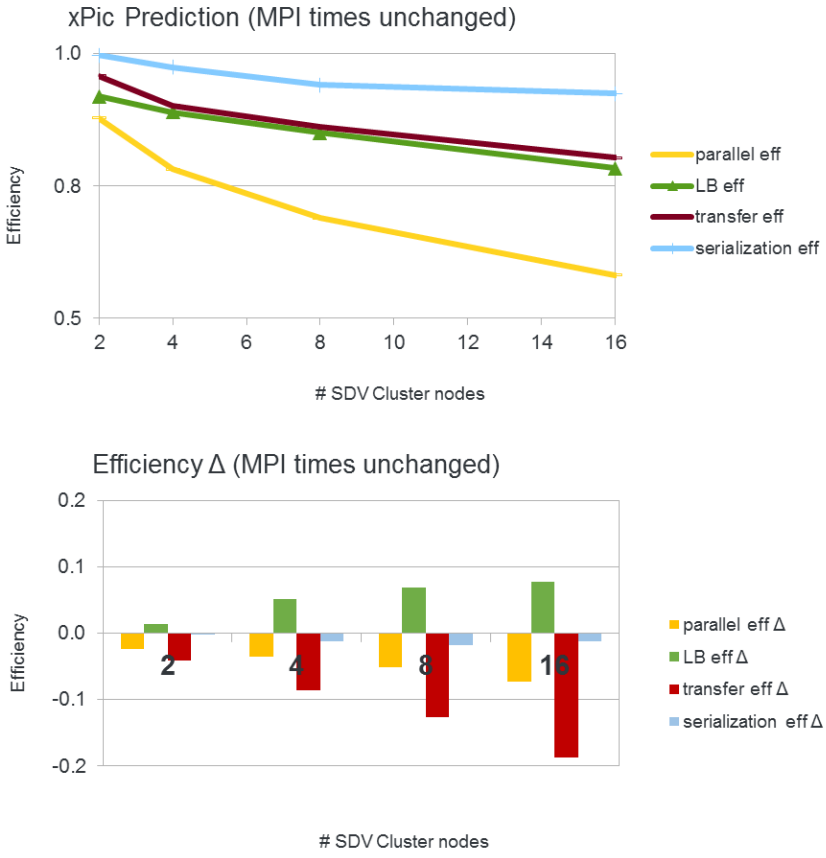


Figure 6: Efficiencies (absolute and delta to measured) for a xPic run without I/O (MPI times unchanged)

The estimations for both global load balance and serialization efficiencies give the same values with both approaches. The difference is only related to the data transfer efficiency, which is significantly worse assuming that the MPI time is constant, since the Dimemas simulation did reduce the time spent in MPI. This is a sign that the I/O activities do indeed affect the MPI times.

To complete the analysis, we compute the ratios per efficiency factor, scale and approach. In Figure 7 d4, for example, stands for the full Dimemas prediction on four nodes, *and w2 corresponds to the analysis on two nodes that keeps MPI time constant. Each bar shows the predicted metric divided by the one actually measured for the instrumented run.

We grouped the ratios by efficiency factors first, second by simulation approach and finally by number of cores. We know that the Dimemas prediction is better suited for this scenario, yet we include some comments on the second approach.

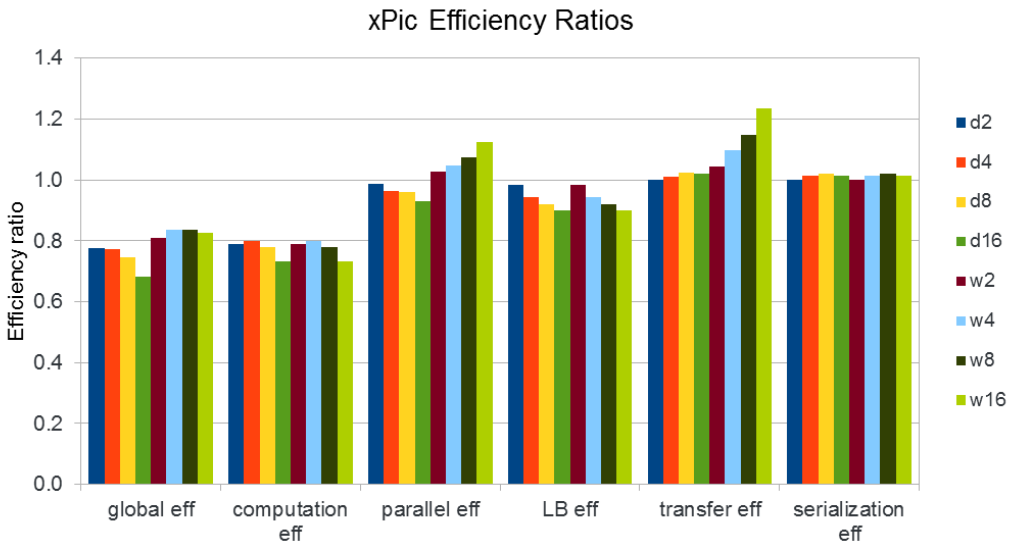


Figure 7: Ratio by efficiency factor, scale and approach

The ratios for the global efficiency are lower than 1 and decrease with the scale. This indicates the I/O has a negative impact on the application scalability. For this metric, the second approach provides as value the actual global I/O efficiency, so we can see that the percentage on I/O is quite similar for all the runs.

The computational efficiency ratios are also lower than 1 and show a small degradation with scale. This metric corresponds to the I/O computational efficiency and it is the same in both approaches. I/O is considered part of the “useful computation”, and removing it would always degrade the computational efficiency as it increases the relative portion of runtime spent in MPI or OpenMP. The fact that the impact on the global efficiency is higher than on the computational efficiency confirms that I/O is also affecting the other factors.

The ratios with respect to the parallel efficiency confirm our last statement, as it is reduced with the scale. These factors are closer to 1, reporting a smaller impact than for the previous metrics.

From the parallel efficiency components, the highest scaling impact is shown for load balance. From a value of close to 1 for two nodes, there is a clear degradation with scale showing that I/O is unbalanced and that this unbalance is increases with scale. Finally, both efficiencies for transfer and serialization have a small variability with scale and are very close to 1, indicating I/O is not affecting these on the scale of system that could be analysed.

As summary for the I/O analysis of xPic, the methodology we have defined reports a problem of unbalanced I/O that is added to the application unbalance, limiting its efficiency (0.7 with only 16 threads). The degradation seen when increasing the number of processes indicates that I/O is limiting the application scalability. The first target should not be to reduce the I/O but to balance both I/O and computations.

The alternative analysis approach that keeps MPI times constant shows a large difference in the transfer efficiency; actually, this factor is shown to improve significantly with scale, and as a result, parallel efficiency goes up. This is contrary to the actual benchmark results, and shows that the full Dimemas simulation approach is indeed more accurate for this workload.

2.1.2 I/O modelling applied to FWI

We applied this methodology to the full waveform inversion (FWI) application developed by partner BSC. FWI is an advanced method for computing highly detailed subsurface velocity fields from the data created by seismic experiments. It performs a significant amount of I/O relative to the computation. Traces were obtained with 4, 8 and 16 MPI ranks on the Cluster part of the DEEP-ER SDV, suing a 3000×4000×3000 grid, using one shot, one gradient, and one test, a vmin parameter of 1500, srclen of 8 and rcvlen of 8.

The following figures show the results of the I/O analysis. First we analyzed the I/O time efficiency with respect to the full execution and with respect to its impact on the computational efficiency. As we can see in Figure 8 the I/O represents around 5% of the application time having a small decrease with the scale. In comparison with the xPic code described above, FWI has lower I/O time with a better scaling.

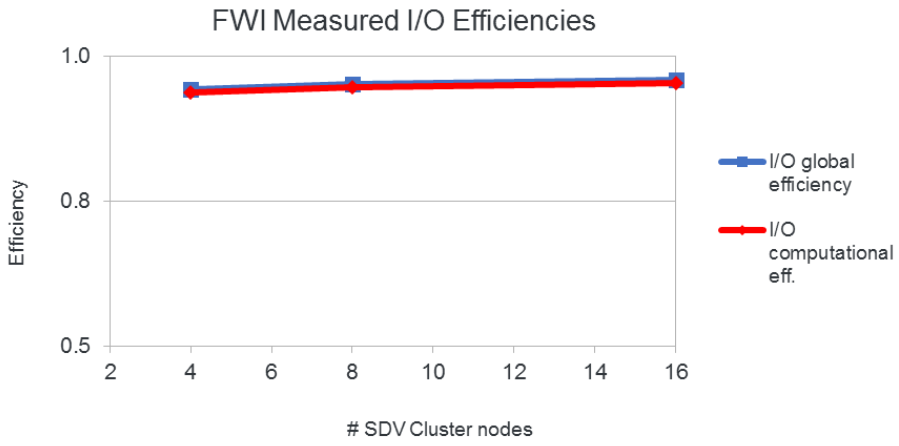


Figure 8: FWI I/O efficiencies

Figure 9 and Figure 10 characterize the I/O with respect to the number of calls and the volume of data read and written, respectively.

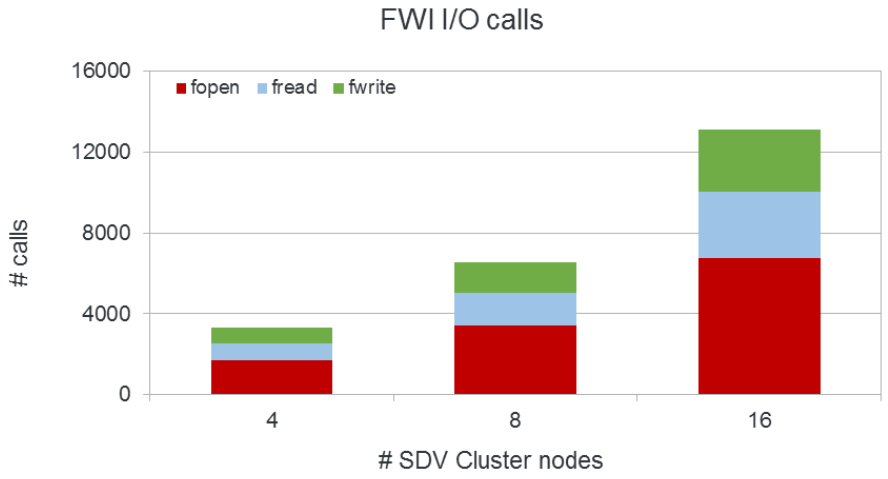


Figure 9: FWI I/O calls

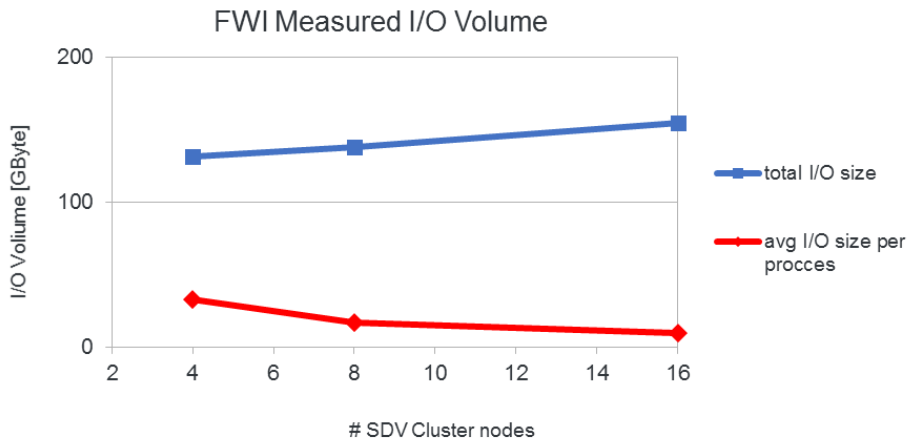


Figure 10: FWI I/O size

Comparing the runs, we can see the number of calls per process is constant (doubling the number of ranks doubles the number of calls). Looking at the volume of data we can see the total volume of data increases with the scale but with a reduction of the average size per process.

Figure 11 through Figure 13 plot the evolution of efficiency factors for the instrumented run and the two approaches for the estimation of execution without I/O. Analyzing first the efficiency of the instrumented run in Figure 11, we can see very good efficiency and scalability on the range of processes analyzed with all the components having a similar impact.

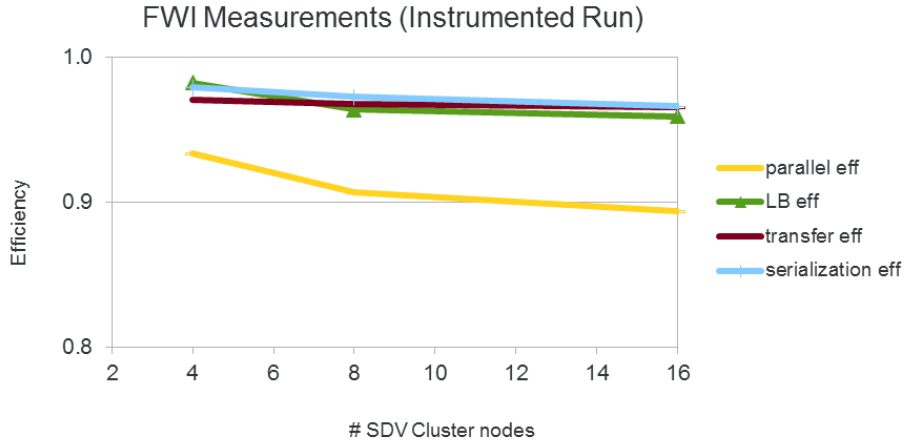


Figure 11: Measured FWI efficiencies.

Comparing the instrumented run to a Dimemas prediction for a run without I/O (like shown in Figure 12), we can see that the parallel efficiency decreases very slightly, due to a slightly worse transfer efficiency. Both load balance and serialization seems not affected by the application I/O.

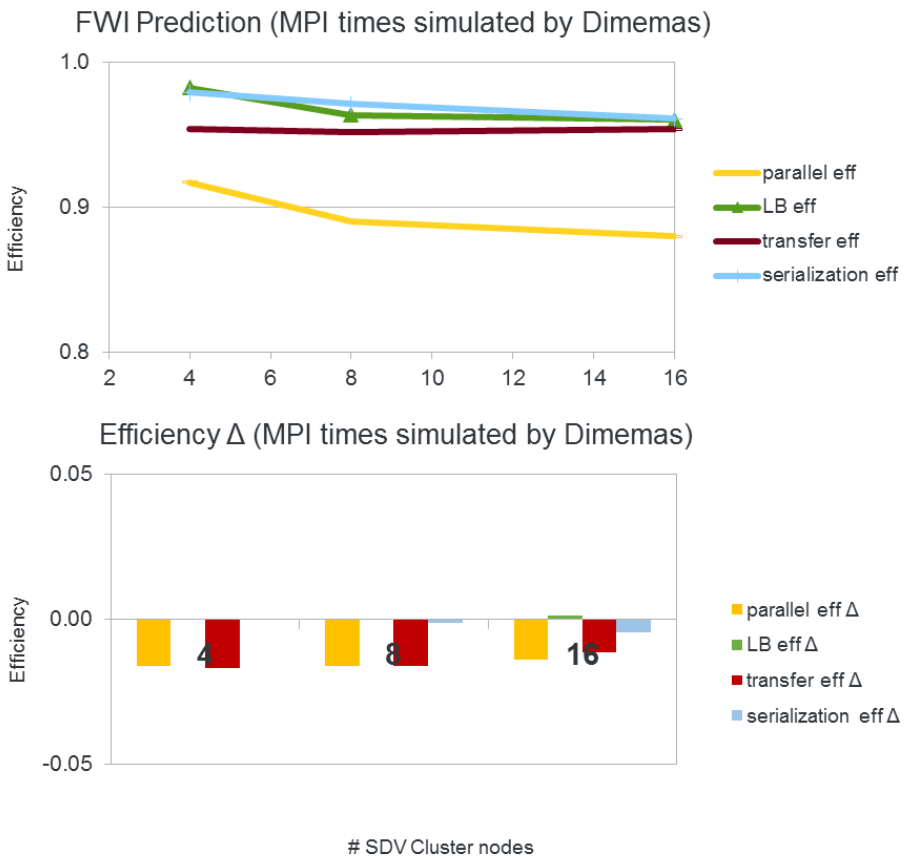


Figure 12: Efficiencies (absolute and delta to measured) for a FWI run without I/O (MPI times simulated by Dimemas).

Comparing the prediction of a run without I/O under the assumption that it has no impact on the MPI time (Figure 13) with the two previous scenarios we can see that for FWI, this approximation provides a result very similar to the Dimemas approach.

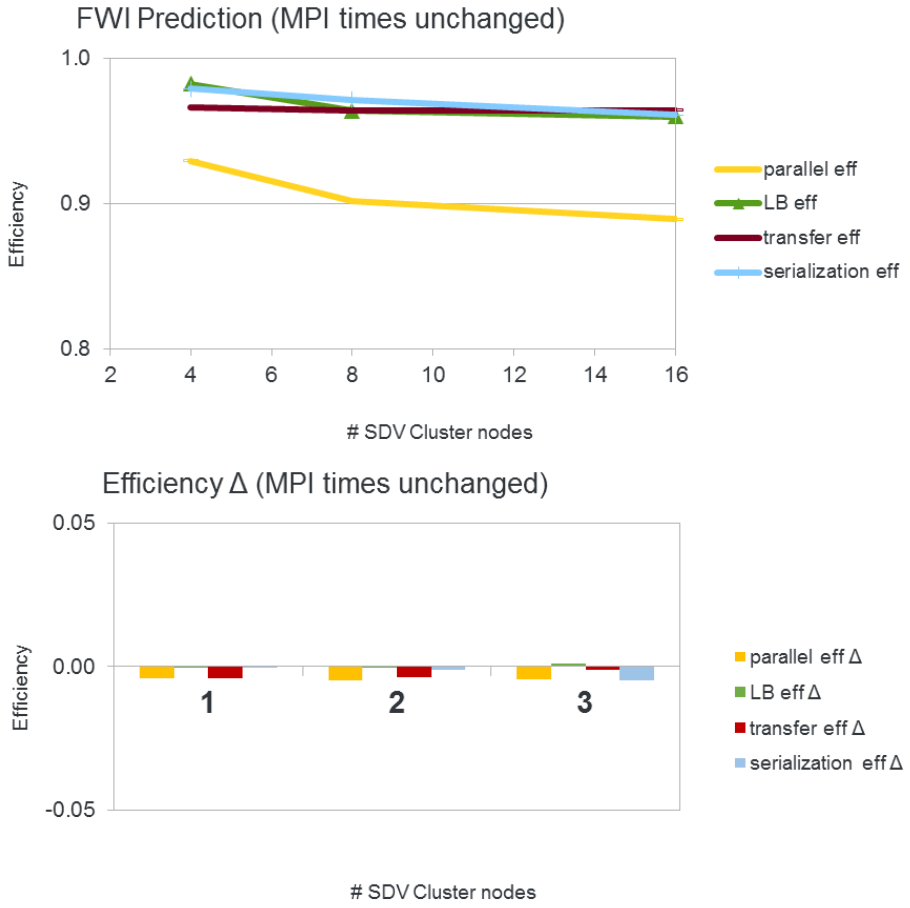


Figure 13: Efficiencies (absolute and delta to measured) for a FWI run without I/O (MPI times unchanged).

The estimation for both global load balance and serialization efficiencies give the same values with both approaches. There is only a very small difference in data transfer efficiency. As in the xPic analysis, we conclude that this is a sign that the I/O activities do affect the MPI times only very lightly.

To complete the analysis, we compute the ratios per efficiency factor, scale and approach. In Figure 14 d4, for example, stands for the full Dimemas prediction on four nodes, and w2 corresponds to the analysis on two nodes that keeps MPI time constant. Each bar shows the predicted metric divided by the one actually measured for the instrumented run.

We grouped the ratios by efficiency factors first, second by simulation approach and finally by number of cores. We know that the Dimemas prediction is better suited for this scenario, yet we include some comments on the second approach.

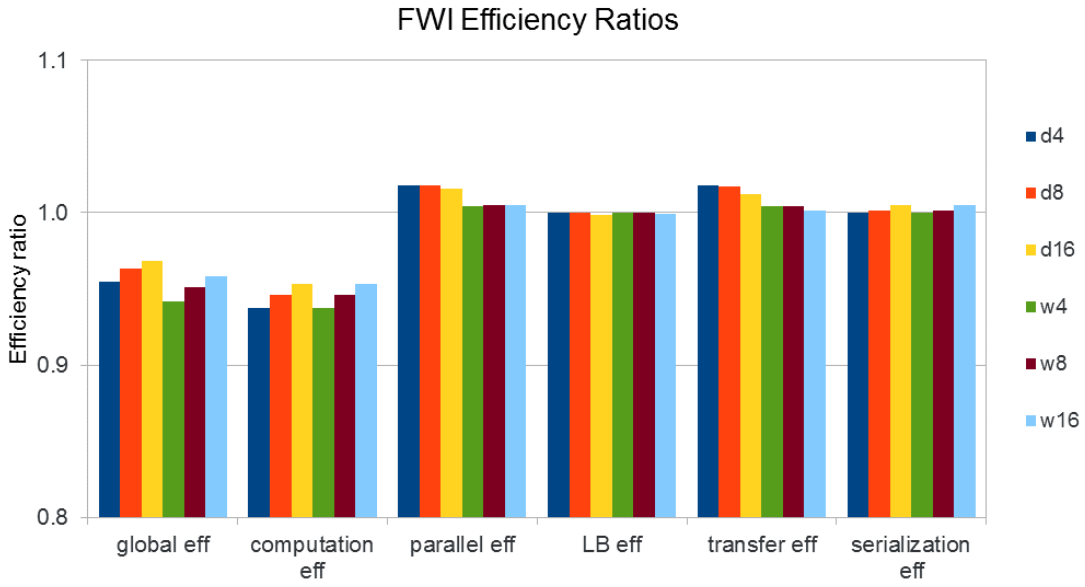


Figure 14: Ratio per efficiency factor, scale and approach

The ratios for the global efficiency are closer to 1 and increase with scale. This indicates the I/O does not have a negative impact on the application scalability. For this metric, the second approach provides as value the global I/O efficiency as shown in Figure 8, so we can also see that the percentage on I/O is reduced with the scale.

The computational efficiency ratios are also a little bit lower than 1 and show a similar behaviour when increasing the scale. I/O is considered part of the “useful computation”, and removing it would always degrade the computational efficiency as it increases the relative portion of runtime spent in MPI or OpenMP. The fact that the impact on the global efficiency is very similar to the computational efficiency confirms that I/O has a very limited effect on the other factors.

The ratios with respect to the parallel efficiency are bigger than 1, reporting a small improvement of the parallel efficiency due to the I/O. The three runs analysed show a very similar value indicating it may not have a correlation with the scale.

From the parallel efficiency components, the impact is reflected on the data transfer that shows ratios a little bit higher than 1. Both efficiencies for load balance and serialization have a constant trend and are very close to 1, indicating I/O is not affecting on the scale analyzed.

As summary for the I/O analysis of FWI, the methodology we have defined reports that I/O is well implemented and distributed in this application, with a very small increase in computation time that do not affect the parallel efficiency and that do not show degradation when increasing the scale.

2.2 Energy Modelling and Extrapolation

There are multiple sources for retrieving information on power and energy consumption of an application. From the processor side there are counters that provide the information while a program is running. This information can be retrieved at the OS level (for instance with the

`turbostat` command), and it only considers the CPU package (actual CPU die plus on-package MCDRAM). Independently Eurotech has provided the capability to measure energy and power while converting the 48V (Aurora power distribution voltage) to 12V needed to run the KNL board. These node energy sensors provide the power and the accumulated energy consumption for the entire node. This includes the DRAM memory, the chipset with all the support functions, and the on-board power converters (from 12V) but excludes the peripheral devices (EXTOLL NIC, NVMe storage) that are powered independently.

To show the usage of this information we performed a LINPACK run on a single node. The program has run twice with immediate restart of the 2nd run after the first run was finished. The LINPACK benchmark performed at the top speed of 1.75 TF/s and each benchmark run took about 500 seconds.

The parameters of the run as monitored from the OS are shown in Figure 15.

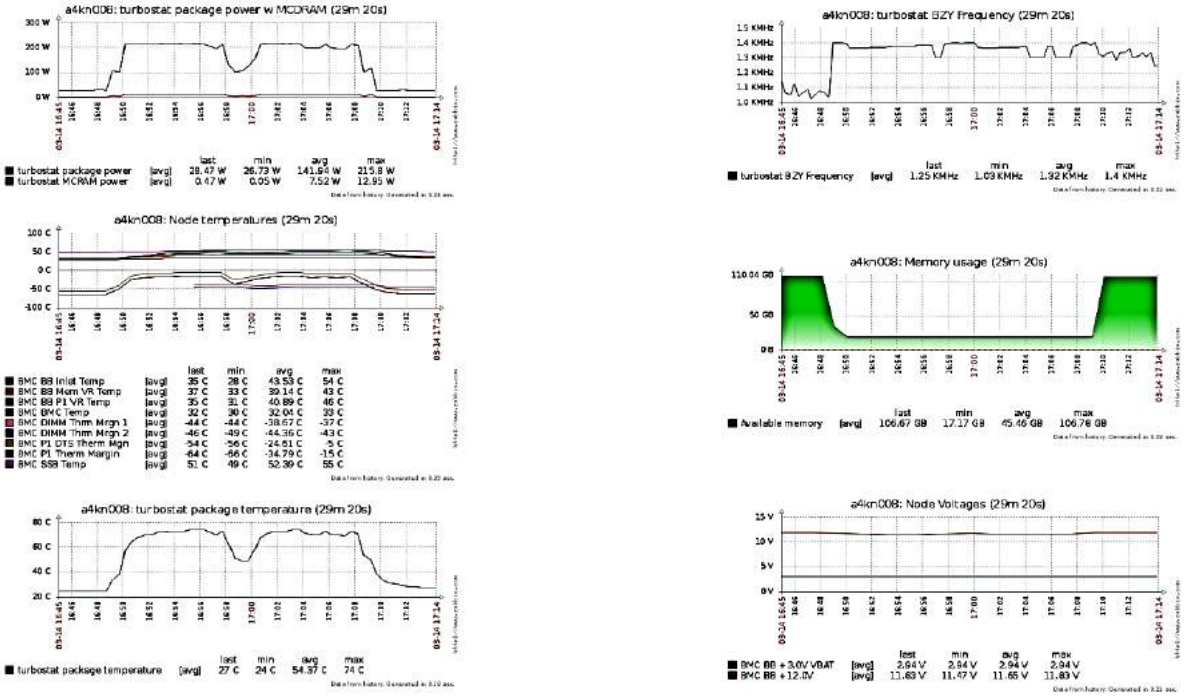
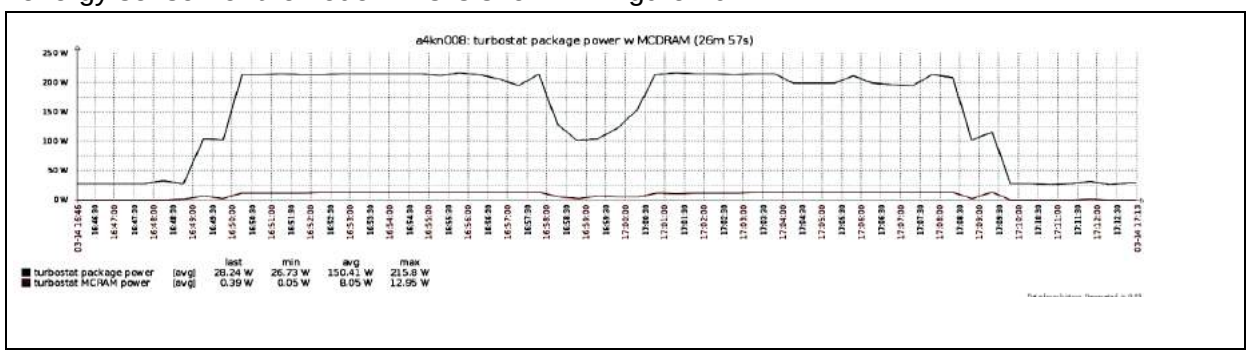


Figure 15.Parameters of the LINPACK run monitored from the OS side. Top left plot displays the power consumption of the CPU and MCDRAM. The top right plot shows core(s) frequency. The temperature of the CPU reached 74°C (left low plot), while the average measured temperature on the board was 48.5°C.

The power displayed by the CPU counters (top left plot in Figure 15) is correlated to the energy sensor for the node. This is shown in Figure 16.



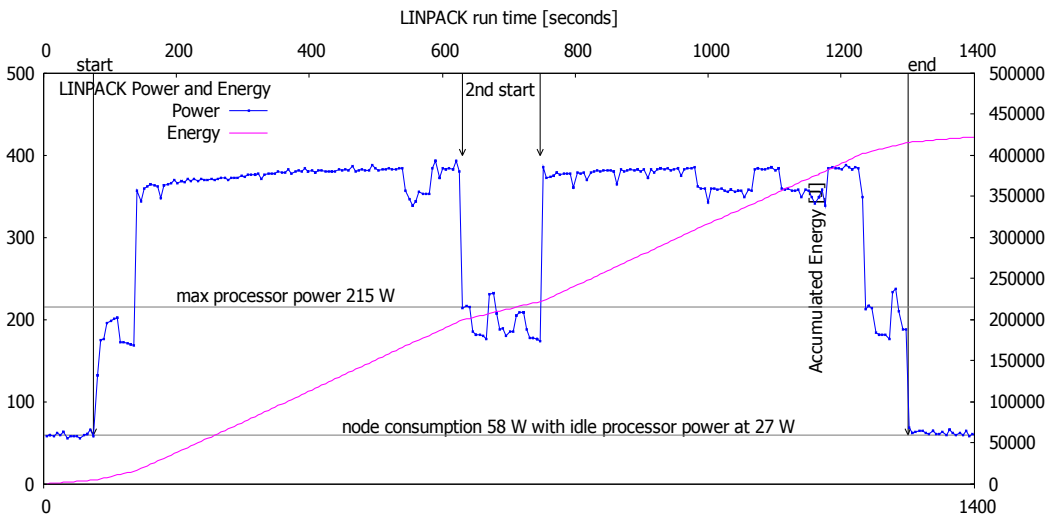


Figure 16. Measurement from the Energy sensor for the node. The LINPACK run between the „start“ and „end“ labels is 1227 seconds, accumulating to 411104 Joules. Max power is 395 W, average 335 W.

As Figure 16 shows we measure the instantaneous power and the cumulative energy spent while running the node. The energy is shown as a purple line, which is normalized to start at zero at the beginning of the two LINPACK runs. If we consider the measurement between the two lines in Figure 16 labeled “start” and “end” the measurement period is 1227 seconds. In this time to total energy spend is 411104 Joules. The LINPACK runs with performance of 1750 Gflop/s. We calculate the LINPACK power density from the energy measurement as 1750 Gflop/s *1227 s/411104 J = 5.2 Gflop/s/W. The average power consumed by the node is 411104/1227=335 W.

The energy sensor measures the maximum power during the LINPACK run at 395 W. In comparison to the processor package maximum of 215 W, the memory and support chips consume 180 W when running LINPACK. In the processor idle state, when only the OS is running on the node and the processor consumes at most 27 W. This is consistent with the chipset and the support electronics consuming about 25 W power. This includes the S7200AP BMC, CPLD, SSD/mSATA and other miscellaneous resources. It should be noted that the Point of Load (PoL 12V) power conversion will consume 17% of the power, while the DC/DC (on the Eurotech CROW board) is another 2-3% loss.

These considerations lead to the following table:

Items	Power idle	Power peak	Comments
BMC, CPLD, SSD/mSATA, Misc logic	25 W	25 W	Fixed contribution
DC/DC and PoL conversion	12 W	80 W	20% of total power
Processor package	23 W	215 W	Measured
Memory subsystem	< 1 W	75 W	Calculated/estimated
Total for the Aurora node	60W	395 W	Measured

We note that due to the direct water cooling there is a saving of 20-50% on the total energy spent, because there is no energy needed for air cooling or refrigeration.

The above figures are strictly for an Aurora KNL node; in the DEEP-ER Booster, the power consumed by the EXTOLL TOURMALET NIC and the Intel SSD device has to be added – from the data sheets, this amounts to an additional power rating of 25 W and 7 W, respectively, for a total of 32 W. It was not possible at the time of writing to measure the actual power use – it is expected that TOURMALET power rating is independent of network use, while the NVMe device rating does depend on I/O activity. Since we do not at this point have Linpack numbers for a full chassis, no energy efficiencies can be computed at this time.

The peak power of the 18 nodes in the chassis may be estimated as 6.9 kW when LINPACK is running (most intensive work load). To estimate the total power consumption of the chassis we add the contribution of the end-points (about 50 W per node) and the overhead of other electronics (100 W). The total comes to about 8 kW.

2.3 Reliability Modeling and Extrapolation

In Deliverable *D5.3 Resiliency Software* [4] we presented our abstract model to study the *asymptotic efficiency* of applications using persistent checkpoint/restart techniques. In this section, we will use and extend this abstract model to predict and study bottlenecks that might appear on future Exascale systems using persistent checkpoint/restart techniques.

Our abstract model is based on the work done by J.T. Daly in [5]. In that work, he defines the application *wall-clock execution time* (T_W), as a function of several relevant parameters of the system and application (e.g., mean time between failures of the system, checkpoint interval length, checkpoint creation time, etc.) resulting in the following formula for T_W:

$$T_W = M e^{\frac{R}{M}} \left(e^{\frac{\tau+\delta}{M}} - 1 \right) \left(\frac{T_I}{\tau} - \frac{\delta}{\tau+\delta} \right) \quad (1)$$

Where τ is the checkpoint interval length, M is the system's mean time between failures (MTBF), δ is the time required for creating a checkpoint, R is the time required for restarting the application from a saved checkpoint, and T_I is the ideal, uninterrupted execution time of the application. The major contribution of the work of Daly is the formula below, used to accurately estimate the checkpoint interval length that minimizes T_W (assuming that all parameters but τ are fixed):

$$\tilde{\tau}_{opt} = \begin{cases} \sqrt{2\delta M} \left(1 + \frac{1}{3}\sqrt{\frac{\delta}{2M}} + \frac{1}{9}\frac{\delta}{2M} \right) - \delta & for\ \delta < 2M \\ M & for\ \delta \geq 2M \end{cases} \quad (2)$$

Our previous contribution presented on D5.3 was to define the *asymptotic efficiency* as: $\eta = \lim_{T_I \to \infty} \frac{T_I}{T_W}$. This *asymptotic efficiency (η)* can be considered to be an *ideal* value that actual efficiency values approach more and more as T_I increases and can be directly expressed as a function of the MTBF (M), the time required for creating a checkpoint (δ) and the checkpoint interval length (τ)

$$\eta = \lim_{T_I \to \infty} \frac{T_I}{T_W} = \frac{\tau}{M \cdot e^{\frac{R}{M}} \cdot \left(e^{\frac{\tau+\delta}{M}} - 1 \right)}$$

Figure 17 shows the optimal checkpoint interval length $\tilde{\tau}_{opt}$, for a wide range of MTBF (from 1 to 512 minutes) and δ (from 1 to 512 seconds) combinations.

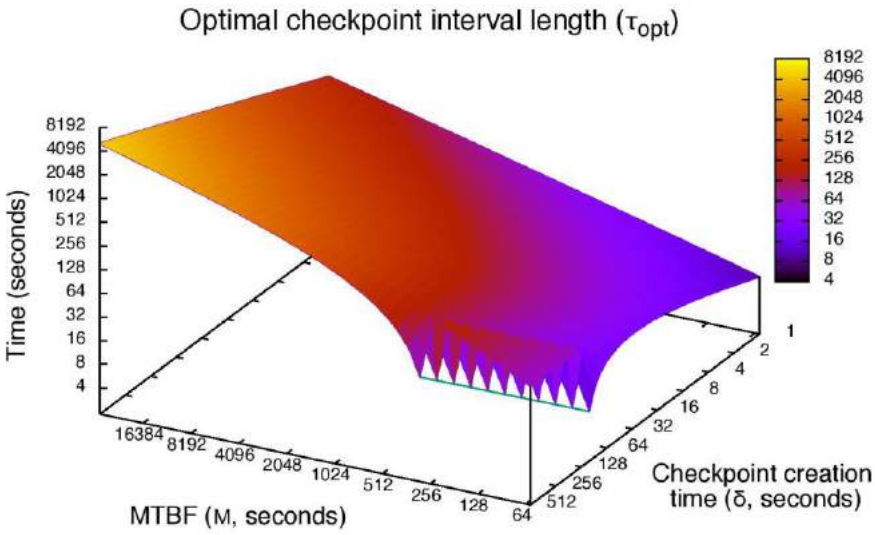


Figure 17: Optimal checkpoint interval length (τ_{opt})

Using the optimal checkpoint interval length ($\tilde{\tau}_{opt}$) calculated on Figure 17 we have depicted the *asymptotic efficiency* for each combination of MTBF and δ values on Figure 18.

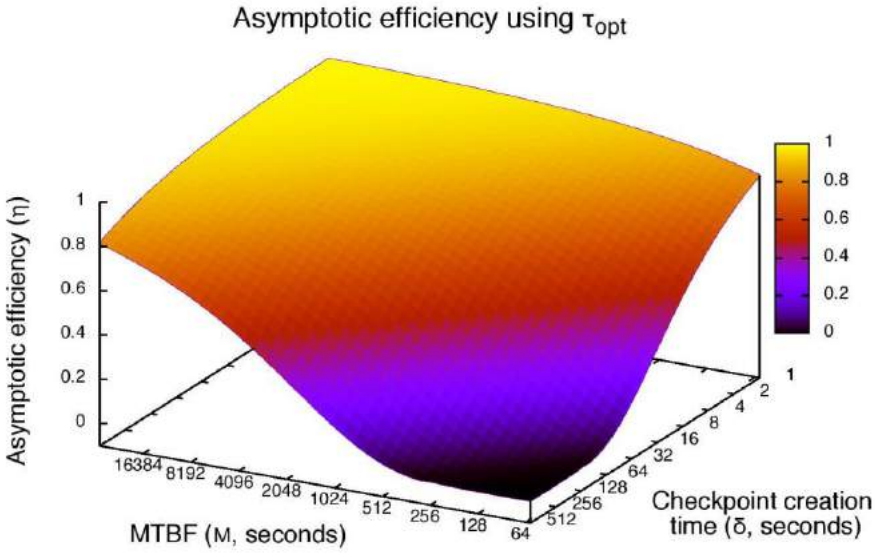


Figure 18: Asymptotic efficiency using τ_{opt}

As expected, the *asymptotic efficiency* increases when the checkpoint creation time decreases and/or the MTBF increases. Conversely, if the checkpoint creation time increases and/or the MTBF decreases the *asymptotic efficiency* decreases.

In WP4 and WP5 we have developed techniques to both decrease the observed checkpoint creation time (by first quickly saving the checkpoint data on the local NVM and then transferring the checkpoint data from the NVM to the PFS asynchronously in the

background) and increase the MTBF (by using lightweight task-based checkpoint/restart techniques to handle *soft* errors locally). Further improvements are still possible to reduce even more the observed checkpoint creation time, such as using faster NVM technologies, overlapping checkpoint creation time with application execution or performing the required global synchronization at the end of a checkpoint asynchronously.

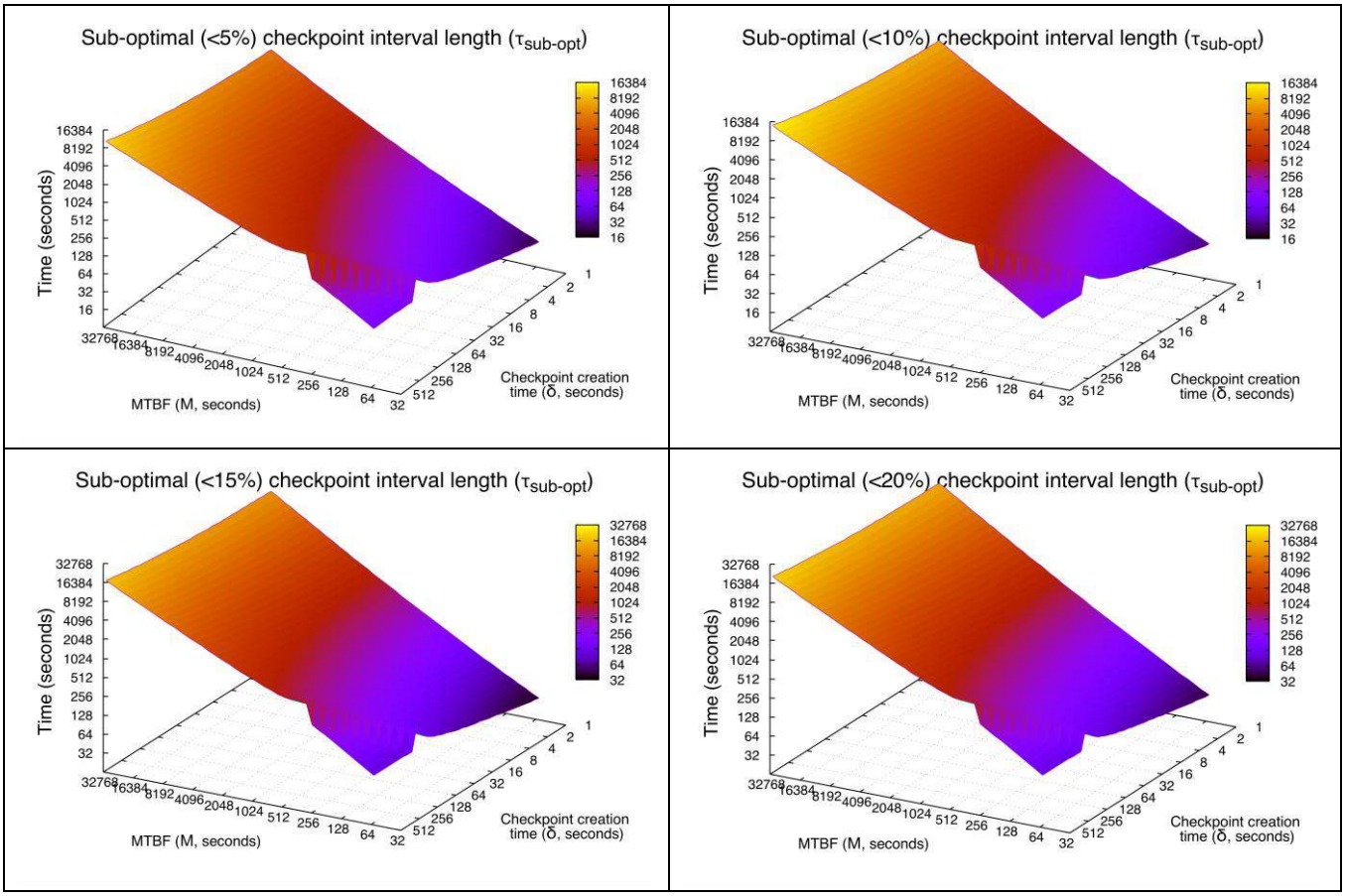


Table 1: Largest checkpoint interval (τ) length that reduces the *asymptotic efficiency* by less than 5%, 10%, 15% and 20% from left to right and top to down respectively.

The previous analysis seems to indicate that if we can further improve the observed checkpoint creation time and future Exascale systems can achieve a reasonable MTBF, current persistent checkpoint/restart technique could be enough to provide an adequate level of resiliency.

However, if we do a more detailed analysis of how persistent checkpoint/restart works, even if we can reduce the observed checkpoint creation time from the application point of view to nearly zero, we still have to eventually move checkpoint data from the local NVM to the PFS. The bandwidth required to move the checkpoint data from local nodes to the PFS can easily become a bottleneck on future Exascale systems. The lower bound (because the bandwidth of the interconnection network is shared between the checkpoint/restart library and the application running) of the bandwidth required is inversely proportional to the checkpoint interval length, as we have to move the current checkpoint data from local NVM to PFS before the next checkpoint data is completely written on the NVM. If we look at Figure 17, we see that the optimal checkpoint interval length $\tilde{\tau}_{opt}$ always increases when the MTBF increases, but the increment is faster when the observed checkpoint creation time is larger.

That means that using the $\tilde{\tau}_{opt}$ when the observed checkpoint creation time is small will put even more pressure on the required bandwidth between local NVMs and the global PFS. On Table 1 we explore the use of sub-optimal checkpoint creation times that are larger than $\tilde{\tau}_{opt}$ (and thus will require less bandwidth between local nodes and the PFS) but have a bounded decrease on the asymptotic efficiency. The four pictures on Table 1 show the sub-optimal checkpoint creation times for a bounded reduction of the asymptotic efficiency of 5%, 10%, 15% and 20% respectively. Notice that the scale of the sub-figures is different.

If we compare the graphs on Table 1 with Figure 17, we can see that by allowing a bounded degradation of the asymptotic efficiency we can largely increase the checkpoint interval length. Figure 19 shows the reduction in bandwidth achieved when we allow a decrease of 10% in the application performance. This reduction ranges from a maximum of 96,6% to a minimum of 40%-50% (we are excluding the quadrant that includes the configurations with the lowest MTBF and largest observed checkpoint creation time, as the low asymptotic efficiency of these configurations make them irrelevant for actual consideration). Thus, on future Exascale machines, it will be crucial to study the trade-of between the asymptotic efficiency of the application and the bandwidth required between individual nodes and the PFS.

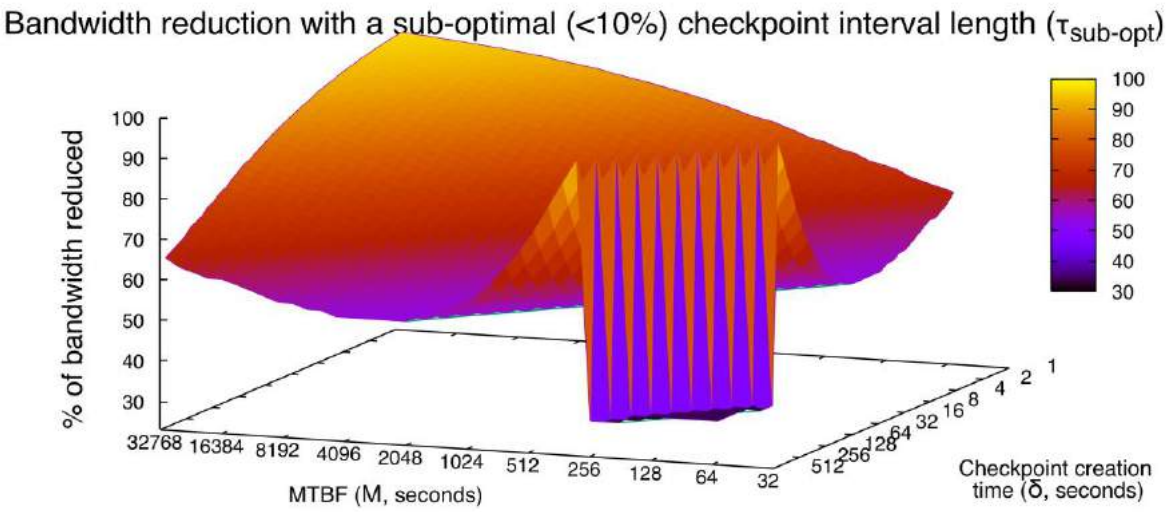


Figure 19: Percentage of bandwidth reduction from local NVM to PFS using the largest checkpoint interval length that reduced the *asymptotic efficiency* by less than 10%

To substantiate the above analysis with measured data, the xPic application by KU Leuven was run on the QPACE3 machine (which has up to 352 KNL nodes and supports BeeGFS). At the time of writing, this system was in its installation and bring-up phase, and a maximum of 64 KNL nodes could be used. This amounts to 8× the possible scale on the DEEP-ER SDV (for KNL nodes). QPACE3 does not have the local fast NVM storage installed – to at least receive best case measurements for such storage, a RAM disk was created on each node and made available via BeeOND.

Figure 20 compares the execution time of 100 iterations of xPic (32768×1×1 cells, 1024 particles per cell, 32 processes per KNL node) for three cases:

- No checkpointing, just computation
- Creating a 3 GByte checkpoint every 10 iterations on the BeeGFS file servers)

- Creating a 3 GByte checkpoint every 10 iterations on the local BeeOND RAM disk.

The data clearly shows that the checkpointing overhead is negligible up to 32 nodes – the BeeGFS file servers can easily absorb the I/O load, and local checkpointing gives no discernible advantage. For the 64 node run, the file servers apparently can no longer cope with the I/O load, and local checkpointing does win hands-down. Further measurements are required at larger scales to substantiate the advantage of local checkpointing, and more analysis is needed to understand why the local checkpointing overhead increases for 64 nodes.

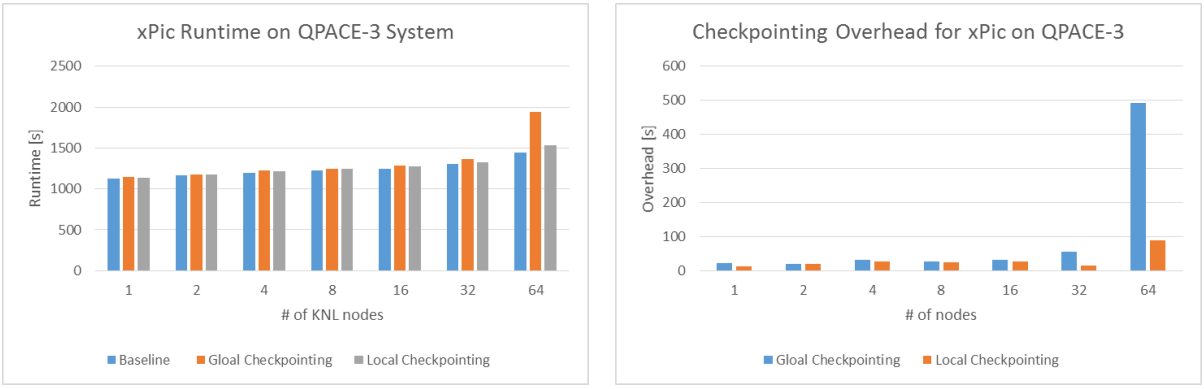


Figure 20: Initial measurements for running xPic on QPACE3 with different checkpointing targets.

Unfortunately, it was not possible to create detailed I/O traces to feed into the I/O analysis of Section 2.1.1 at the time of writing this Deliverable. For the DEEP-ER final review, it is hoped that such data will be available, in addition to runs on larger configurations than 64 nodes.

2.4 Scalability Challenges

The delays in integrating the full DEEP-ER system did, naturally, materially impact the amount and depth of scalability analysis that could be performed. The DEEP-ER SDV has been extremely useful to analyse and optimize applications, develop and validate the system software, and testing the DEEP-ER concept. With just 16 Intel® Xeon® and 8 Intel® Xeon Phi™ nodes, it does not provide a solid base for scalability extrapolations, however.

From the results of Deliverables D7.1 and D6.3, and from the preceding sections, the following, preliminary conclusions about the eventual scalability of the DEEP-ER architecture can be derived:

- The presence of fast, storage-class memory at the compute nodes does make a substantial difference for I/O performance, as long as this memory can be effectively utilized by the application for its direct I/O needs or for resiliency (buddy checkpointing). The DEEP-ER I/O stack does provide a very convenient way for applications to leverage this advantage, and thus this investment in adding local storage-class memory to the nodes pays off in two ways that help scalability:
    - Inherently collective I/O patterns (such as occurring with the use of MPI-IO and with checkpointing) are accelerated
    - Pressure is taken from the parallel I/O server infrastructure since BeeGFS and SIONlib can cache data from per-node I/O

Still, it is necessary to grow the end-to-end throughput of the I/O server infrastructure with the scale of the system; the difference to a conventional architecture is that the time available to migrate data to the backend I/O system can be overlapped with computation. For weak scaling applications, this amounts to a constant factor (depending of course on the application and use case); for strong scaling applications, the advantage will decrease while the system is being scaled up, since there is less compute time to overlap with, and while I/O data volume stays constant, the need to handle smaller and smaller I/O chunks will lead to more overhead.

For application scenarios that allow to keep the data on the node storage (for instance for passing between steps of a workflow), the pressure on the I/O server infrastructure will be further reduced (ideally to zero for temporary data), assuming a data-aware co-scheduling scheme or further I/O stack extensions that would transparently access remote node storage.

As a result, we see the DEEP-ER memory/storage architecture as providing a significant boost in I/O scalability. It does not abolish the need for growing the effective, end-to-end I/O server bandwidth altogether; pushing the evolution of applications and system middleware that support keeping data on the local storage for a longer time will be a good next step to further reduce the challenges of scaling delivered I/O performance.

Given the limitations inherent in a first prototype implementation, it is hard to assess the performance aspect of the NAM. Its role in the future can be as another layer in the memory/storage hierarchy, with performance characteristics (and cost) in between node-local storage and a conventional storage server architecture.

- As mentioned above, putting application (and task) checkpoints into local on-node storage has been proven to work, and a first step towards providing redundancy by way of the buddy checkpointing scheme. With the 2× redundancy used with DEEP-ER, restart of a system is assured for single-node failures, and for failures that affect up to half of the “right” nodes (like, for instance, only odd or even nodes). As systems scale higher, a larger absolute number of nodes can be expected to fail, and it will likely be required to guarantee higher levels of redundancy – this in turn drives storage space requirements per node, or it will require storage-class memory that stays accessible even in case of node CPU or DRAM failures.

  The NAM can be very useful as a fast storage target for checkpoints, in particular since it can also orchestrate the data transfer and handle the data processing required for creating redundant checkpoints. Of course, the number of NAMs in a system would need to scale with the number of nodes in a system (to keep data transfer performance constant), or at least with the accumulated DRAM in the system (to accommodate all checkpoints). In such a scenario, the reliability of a NAM node (compared to a compute node) becomes relevant – it might be possible that using more conservative design methods for a NAM (larger structures, lower clock speeds, lower temperatures) can achieve just that.

- The EXTOLL TOURMALET NICs have materially improved the networking performance compared to the DEEP project, and the balance between Booster compute and point-2-point communication performance has been kept roughly the same. Since all nodes that are neighbours in the EXTOLL fabric topology can directly

communicate (with no intermediate nodes/switches), such local communication is not at all affected by scaling up the system. Applications that can be mapped onto a supported topology (3D grids, 2.5D tori), and that rely on local communication (such as halo updates) are therefore expected to scale well, and the fabric itself can be linearly scaled up.

Non-local and/or collective communication is a different matter. The EXTOLL fabric has to pass packets via intermediate nodes for these, and while it does this (i) without CPU involvement and (ii) with a very small per-hop latency cost, the end-to-end latencies for collective operations increase with the system scale (basically the diameter of a 3D grid/torus), and the effects of link contention have to be taken into account, in particular if local and non-local communication schemes are combined.

The above challenges can certainly be addressed – two-phase or non-blocking collective operations, combined with application or even algorithm adaptations could hide collective operation latencies, and dynamic, smart orchestration and routing of packets between nodes can ameliorate contention in a direct-connected network.

3 Technology Development Outlook

The DEEP-ER project has performed research and development in computing architecture with the goal of improving the Cluster-Booster concept, enabling data-intensive applications to profit from the architecture's advantages. Subsection 3.1 describes the evolution and future prospects of the Cluster-Booster architecture. The remaining subsections describe the series of hardware and software technologies, which have been investigated and further developed in DEEP-ER, and how they are expected to evolve in the future.

3.1 Overall system architecture

The predecessor DEEP project showed that by segregating heterogeneous processing technologies into homogenous, interconnected cluster systems, a high flexibility in system usage and application scheduling could be gained. A 'Cluster' of general purpose Intel Xeon processors was attached to a 'Booster, which is actually a cluster of Intel Xeon Phi co-processors (KNC generation in the case of DEEP). This configuration (two homogeneous clusters connected to each other to create an overall heterogeneous system) allowed application developers to freely choose the amount of Cluster and Booster nodes that their code required.

Users execute the low-scalable parts of their codes on the Cluster (profiting from its high single-thread performance), and the high-scalable parts of their codes into the Booster (profiting from its scalability and energy efficiency). To do so, they rely on the MPI_comm_spawn() call, implemented in ParaStation MPI on top of an specifically developed Cluster-Booster protocol, which bridged between the two different network technologies of the two parts of the system (InfiniBand and EXTOLL, respectively). On top of MPI, the OmpSs programming environment provided the users with a pragma-phase interface that made the code distribution even easier.

Application developers soon realised that this new architecture and its software stack based on standards provided them with much more flexibility, enabling use-modes never possible before, while keeping the code portable and performant on standard HPC systems. System administrators on their side, noticed that the use of the resources is maximised, as each single application reserves exclusively those processors that are needed for its execution without blocking others due to their dependency from each other (as it is can well happen in host-device configurations). However, the developers also noticed that use cases with high data management – especially if the part of the code managing the data was to be executed on the Booster –, could easily run into problems due to the reduced memory capacity in the system.

Starting with this co-design input obtained from the DEEP applications, the DEEP-ER project designed a second-generation Cluster-Booster system with a multi-level memory hierarchy that provides much more capacity and memory bandwidth than its previous generation, to fulfil the increasing I/O requirements of today's HPC codes. Booster nodes with increased on-node memory capacity (96GB larger than in DEEP) where chosen. Additionally, a large pool of distributed non-volatile-memory (NVM) was added, providing each node (in both Cluster and Booster) with direct, high-speed access to a 400 GB NVM device. Such extended capacity not only increases the code's I/O performance, but enables as well new resiliency strategies such as 'buddy-checkpointing', which increase the application's tolerance failures. On top of that, the concept of adding memory devices to the network,

globally accessible to all nodes in the system, has been also explored with the development of the innovative Network Attached Memory (NAM) devices.

DEEP-ER has also upgraded the processor technology on the Cluster and Booster sides of the system. Especially important is the use of the second generation Intel Xeon Phi processors (KNL) on the Booster side. These self-bootable devices make the construction of a Booster system much more straight-forward than it had been in DEEP, where special software and hardware components had to been developed to enable the KNC operating autonomously. The matureness of the technology has also led to the decision of using the EXTOLL interconnect on both Cluster and Booster sides of the DEEP-ER system, so that no bridging protocol is required anymore.

From the architectural point of view, the DEEP-ER improved system design and software stack have ratified the conclusion reached in DEEP: Organising heterogeneous resources at the system level (instead of at the node level) provides the users the highest flexibility on how to execute their applications, and enables system administrators to exploit the hardware resources at the maximum. The goal that the team has posed itself for the near future is to generalise the Cluster-Booster concept to create a “Modular Supercomputer Architecture” (MSA).

Cluster and Booster can be seen as two homogeneous computing modules: the Cluster, tailored to applications (or parts of them) requiring high single-thread performance; and the Booster, tailored to highly scalable codes (or portions of them). Further computing modules with specific characteristics for other kinds of codes relevant in HPC can be added. For example, a module addressing the needs of High Performance Data Analytics (HPDA) codes would allow enlarging the application portfolio of HPC centres.

A most diverse application mix can be efficiently scheduled into an MSA system. Even more, application developers will be able to execute complex workflows that combine pre-processing, modelling, data analytics, verification, post-processing and visualisation into a single machine, reducing the data movements and therefore reducing time-to-solution and power consumption.

In DEEP-ER, we are convinced that the ideas first introduced in DEEP, further developed in DEEP-ER, and to be brought to matureness in DEEP-EST, will soon become reality in production environments. In fact, JUELICH is already making the first step by preparing the installation in 2017 of a Booster system for its existing JURECA cluster.

But the path is not stopping here. Thinking into the further future, the Modular Supercomputing Architecture allows integrating into a classic HPC environment more exotic computing technologies such as neuromorphic systems or quantum computers, which can boost the performance of specific kinds of codes or fields of research. These ground-breaking technologies have the potential of changing the way we understand High Performance Computing. The Modular Supercomputing Concept may become the vehicle to very soon enable their entrance in today’s HPC centres.

3.2 Node Architecture, Processors and Memory

At the end of the DEEP project, Deliverable D9.2 [6] did lay out a structure for the evolution of the HPC compute node and its processors and memory components, and reflected that by the requirements to reach sustainable Exascale-level performance. In this section, we will

follow that structure and update the analysis according to the progress in the DEEP-ER project and in HPC technology in general.

3.2.1 Process and Manufacturing Technology

Traditionally, regular advances in CMOS process and manufacturing technology were instrumental to compute technology. Following Moore's widely publicised (and often misunderstood) law, feature sizes of CMOS chips would be reduced by a factor of 1.4 every 18 months, which results amongst others in lower costs per transistor, opportunity to manufacture more complex devices, reduce operating voltages, achieve lower leakage currents (Dennard scaling until 2005). Focus in recent years (after the proverbial power wall was hit around 2005) was on architectural innovation (enabled by the opportunity to use more transistors) and on improved performance/higher energy efficiency by "non-Moore" innovation, like for instance the use of high-K dielectric materials and 3D FinFET transistors.

Currently, there is much talk about the end of Moore's law, and about "post-Moore" Computing in scientific and industry circles. This rumour of an imminent death is certainly exaggerated, as communication from industry leaders (Intel, IBM) is clear that 10 nm products are imminent, and the next step to 7 nm is being prepared. However, all exponential laws have come to an end, and published data indicates that the economic drive behind Moore's law (decreasing per-transistor cost) might be weakening already.

For the purpose of discussion, even after the end of Moore's Law, a number of process and manufacturing technology improvements are still relevant to HPC:

(1) Integration of multiple, heterogeneous dies by way of multi-chip modules, Si interposer technology and Si Embedded Multi-die Interconnect Bridges [7].
(2) Increased densities and signal rates by use of 3D stacking techniques.
(3) Integration of silicon lasers and modulators/multiplexers to support silicon photonics.
(4) Advanced CMOS design and manufacturing, and adoption of new, "exotic" materials (nano-wires, Graphene) that are CMOS-compatible.

Improvement (1) helps the basic economics of producing highly complex processors, and shows a way to combine heterogeneous technology nodes. (2) can accommodate faster and larger capacity random accessible memory, and promises to substantially increase available bandwidth. (3) is a required step to embrace optical communication between nodes or racks, and for the design of optical switches. While a being a bag of different technologies, (4) will drive the basic energy efficiency of future logic circuits.

The above is phrased in terms of additions/extensions that can leverage the well-established CMOS process and manufacturing ecosystem. This is considered important to ensure that evolved logic and memory packages can be produced in an efficient and scalable way – disruptive approaches that require a full new design and manufacturing chain pose a very high risk, and experience with CMOS shows that it will likely take a long time until dependable, efficient and affordable mass production of functioning parts is possible.

3.2.2 Processor Architecture

Traditionally, the main sources of performance gains in HPC CPUs have been the increase in both the instruction-level parallelism (ILP) and the thread-level parallelism (TLP). The former can be achieved by increasing SIMD performance with support for longer and longer vectors (the DEEP-ER Booster uses 512 bit wide vectors), and the latter can be pulled off by

adding more independent processor cores. There is a tricky balance to be struck – increasing vector lengths can greatly complicate the CPU cores, requires very efficient data management functions (gather, scatter) and a very strong, expensive memory subsystem due to the required bandwidths. A sophisticated speculative and out-of-order execution engine would also be a big advantage. Having a larger number of simpler cores can be more energy efficient and flexible, yet the overheads imposed by thread synchronization (and badly written parallel programs) have to be accounted for. One can view GPGPUs as a proof point for the performance levels that a large assembly of very small cores can provide when coupled with very high bandwidth memory – yet this level of performance is not available across applications, since it requires a highly regular behaviour across threads.

One of the key advantages of the DEEP-ER architecture is that a suitably partitioned application can take full advantage of very widely threaded CPUs and narrower, higher clocked processors at the same time. Therefore, improvements in either school of architecture thought can profit DEEP-ER immediately.

On the micro-architecture side, capabilities of CPUs to put cores into very deep sleep states quickly (and of course re-awaken them) will further increase, in addition to dynamic voltage and frequency switching (DVFS). This has a direct effect on saving energy, since given a suitably capable runtime system, energy consumption of cores or functional units that are idle can be very close to zero, and DVFS could be employed to speed up “lagging” threads or processes to avoid idle times in a larger thread/process pool. Taking full advantage of such dynamic energy and/or performance optimizations will need advances in the system software plus in the hardware (processors, but of course also I/O controllers and NICs).

The promise of further efficiency gains by integrating “accelerators” (usually referring to more specialized compute or data processing devices) remains tempting. One can do this either at the node (“accelerated nodes”) or at the system level (adding another “module” to a DEEP-ER style system).

For the accelerated node approach, an efficient integration or attachment of the accelerator with the main processor(s) and memory is key. The evolution of PCI Express to generation 4 will again provide a good boost in bandwidth (doubling the data rate compared to generation 3, with a further bandwidth increase by a factor of 1.5 in the future), and proprietary attachments such as NVLink go a step further in performance and functionality (like for instance, memory and cache coherence). Both technologies are available in first implementations. For FPGAs, Intel has announced an integrated Xeon plus Altera FPGA package that internally uses the cache-coherent Ultrapath Interconnect (UPI) link.

Another important aspect about accelerated nodes is of course the actual utilization of the accelerator by applications, and the power envelope of the full node, which either has to support operating CPU plus accelerators at the same time, or, in a “modal” approach, allow to operate one of both at the same time.

The advantage of an accelerator “module” attached to a DEEP-ER style system and available for dynamic partitioning and allocation according to workload needs is that the ratio of the quantities of resources allocated to a workload is flexible, and that given a good mix of workloads, idle times of the accelerators can be avoided, as can be situations where silicon has to be “dark” due to on-node power constraints. Thus, for DEEP-ER to further increase its efficiency and performance, development of accelerator modules with the capability to be

dynamically partitioned, composed with resources from the DEEP-ER Cluster and Booster and used to run a workload at the best operating point would be the best course.

3.2.3 Memory and Storage

DEEP-ER has clearly shown the benefit of integrating storage-class memory with compute nodes on I/O performance, in effect creating an additional layer in the system memory hierarchy. In addition, the added fast memory (MCDRAM) on-package at the KNL nodes has shown a significant improvement of compute performance. From the development of technology in the DEEP-ER timeframe, it is clear that both directions of "memory innovation" will continue, and that a DEEP-ER style system can profit from these advances.

In the storage-class memory, PCIe-attached devices that support the NVMe protocol have become mainstream, with small form factors (such as M.2) complementing PCIe add-in cards, and new iterations of flash and non-flash technology media being integrated. One example of the latter is the Micron and Intel 3D XPoint™ memory technology, which promises further significant increases in bandwidths and endurance [8]. Since DEEP-ER storage class memory is based on the PCIe and NVMe standards, new generations of NVMe PCIe ×4 add-in cards can be readily integrated, just requiring an adaptation of the Eurotech cooling solution. Taking advantage of emerging ×8 cards should be as straightforward. Finally, adopting the M.2 form factor could be interesting in order to further increase density – this will need a redesigned board and Eurotech coldplate. It is more likely that Future versions of DEEP that use KNL's successors would include the M.2 form factor.

The trend towards incorporating faster, on-package memory with high-end GPGPUs and Intel Xeon Phi CPUs will continue, since conventional DRAM is hitting its technology limit. DDR5 will not close the gap towards the bandwidth need of KNL or its successors. Advances in HMC or HBM technology should over time deliver bandwidth increases compared to MCDRAM. DEEP-ER systems would automatically profit from such advances after newer Intel Xeon Phi CPUs are integrated.

3.2.4 Node integration

In both the Intel and IBM/NVIDIA camp, there is a clear trend towards tighter node integration, with NVLink and Omni Path Architecture becoming a part of the CPU package, rather than being attached to the PCIe bus. This does enable higher bandwidth, saves energy due to the elimination of connectors and PCB traces, and provides future opportunities for tight integration of the NIC with the CPU proper, as for instance with cache line and core management and with a low-level hardware thread scheduler. At this time, no fully integrated CPU/NIC combos are available, and it remains to be seen what the impact of such integration is on end-to-end communication performance and efficiency.

Highly optimized MPI communication systems, such as Sandia Lab's Portals [9], are another relevant development. The idea behind these is to accelerate MPI message injection and reception by tapping into specialized hardware (like with Bull's BXI interconnect), or by integration closely with the CPU mechanisms. As demonstrated by BXI, such a scheme can be implemented for PCIe attached NICs; again, there is are theoretical advantages of on-package or on-die integration that have to be substantiated by actual implementations.

As alternatives for third-party fabrics and NICs such as EXTOLL, an integration on package or on an SoC might be possible, if the economics of doing so are favourable; the fall-back

would be adoption of PCIe generation 4, which should suffice for fabric speeds up to 200-300 Gbit/s per link. In this case, the benefits of tight integration with a CPU will not be available. However, as said above, these have yet to be proven.

3.3 Interconnect Fabrics

As future extreme scale systems continue to grow in size and speed the interconnection network continues to play a strong role for the overall system performance. By means raw network link parameters technology scaling remains the largest source of improvement.

We furthermore expect heterogeneous systems to evolve in the near to midterm. Such systems will comprise several highly specialized compute cluster, each encapsulated in a separate network domain and topology. Maintaining inter-cluster network performance as well as dead-lock free and adaptive routing, help to mitigating failure of network components, will require complex routing setup algorithms.

Increases in bandwidth, not only peak bandwidth reachable with large packets, but also for small and medium sized packets as used in many HPC applications will play an important role. Adoption of newer host interfaces as PCIe 4.0 as well as other interfaces will help in this direction in the near future. As an example, next generation EXTOLL devices are planned to at least double the available bandwidth per node. At some point, integration of the NIC with the processor will offer better bandwidth and especially energy metrics then going of chip. Integrating the NIC on the package is only an intermediate step on this road and still poses significant energy usage for the interface between CPU and network die. On the other hand, integrating network with COTS CPUs poses difficult economic challenges that still need to be solved.

On the latency side, improvements in the network core, i.e. switches and routers, will still be possible and employed by future HPC solutions. For example, next generation EXTOLL devices are estimated to cut latency by about 30% on the network side. On the other side, application to application latency, especially if nodes are neighbours, is already today dominated by the latencies of the host interface and the CPU/memory subsystem. Integrating the network interface directly with the memory subsystem on the CPU die may be the only solution to offer significant benefits here.

For extreme scale systems, network management, reliability and fault tolerance become an ever more important subject. Direct networks as offered in the European EXTOLL solutions inherently offer multiple paths between any endpoint and as such are a good subject, since in principal no single-point-of-failure exists, which can disconnect the complete or large parts of the network. However, further innovation in routing, management and quick and automatic fault detection, isolation and handling is needed, both on the system software side as well as on the hardware architecture.

Finally, energy consumption remains an important topic also for the network part of extreme scale systems. There is still some headroom from semiconductor technology since traditionally network ASICs are manufactured from older CMOS processes then state of the art CPUs. Integrating the network interface directly with CPU dies will offer significant savings on the CPU to network interface and incorporate the before stated improvements from process technology. Unfortunately, this even in sum, is only a small part of the complete power consumption of the network, which generally is dominated by the power needed to drive the actual amount of links between nodes, which is higher than the number of host

network interfaces. The energy needed to drive a signal over a link is determined by the physics of the link, not the actual CMOS devices used to implement the driver. As a corollary, the transistors used to driver off-chip signals have not decreased in size as the internal transistors when transitioning from one process node to a smaller process node. This also means that off-chip drivers silicon real estate usage does not directly scale when choosing a smaller process node to implement network hardware.

More aggressive link power management may provide some (albeit small) improvements. Newer SerDes architectures, though, offer significant savings due to better coding and driver/receiver architectures. At high bit rates, moving from pure electrical to optical links does offer the potential for some additional energy savings.

3.4 I/O and Storage

We next discuss some of the areas and opportunities in the evolving I/O and storage landscape that need to be looked into, for helping to scale the prototype to Exascale.

***Non Volatile Memories and "deep" I/O hierarchies*:**

The DEEP-ER prototype has explored the usage of Flash storage technology in the I/O hierarchy, which is part of the compute node, to hide the performance mismatch between the compute and storage subsystem. This opens up the question on the usage of new byte addressable non-volatile memories such as 3DXPoint[2] appearing on the horizon. The usage of these devices is still very unclear, in terms of whether they are better used to extend the memory addressing range, or used as top tiers in an I/O stack. Projects such as NextGenIO[3] are already studying the usage of such devices. There may be a play for those technologies in architectures such as DEEP-ER going forward.

The next issue is the usage of multiple types of devices in a very deep I/O hierarchy providing varying performance/capacity points as desired by the applications. The FastForward[4] project has studied the burst buffer tier placed between the compute and the storage subsystem to absorb I/O bursts and hide some of the performance gaps between compute and storage. We have extended some of those concepts in DEEP-ER using node local persistent non-volatile memories. However, it will be very interesting to study the evolution of DEEP-ER for using tall I/O hierarchies that involve Memory, Byte addressable NVRAM, Flash, fast disk tier & archival grade Disk Tiers. Indeed, the in-network non-volatile memory pools, if available can also be made part of such a deep I/O hierarchy. The SAGE Project[5] is studying the usage of such I/O hierarchies.

Usage assumptions of data storage:

Traditionally the HPC codes used the storage for limited data resilience (check pointing). However, the storage system will continue to have a bigger role in HPC workflows. Firstly, HPC will continue to be more and more data centric, especially going towards Exascale. Not only will the codes generate and consume a lot more data, but they will be part of very complex workflows that will include data gathering from instruments, pre-processing, post processing and data analytics in parallel with running simulations. There is a need to rethink storage architectures that can combine the characteristics of massive data analytics and

[2] http://www.digitaltrends.com/computing/intel-optane-ssd-2016-3dxpoint/
[3] http://www.nextgenio.eu/
[4] https://wiki.hpdd.intel.com/display/PUB/Fast+Forward+Storage+and+IO+Program+Documents
[5] http://www.sagestorage.eu

Exascale HPC as having separate architectures is only going to be prohibitively expensive in terms of cost and energy footprint. Such new workflows are evidenced by use cases such as the SKA[6]. It will be an interesting opportunity for DEEP-ER type architectures to look at such workflows and study how the I/O system can adapt and cope with them.

Figure 21 depicts the new workflow for I/O and storage in the Exascale era.

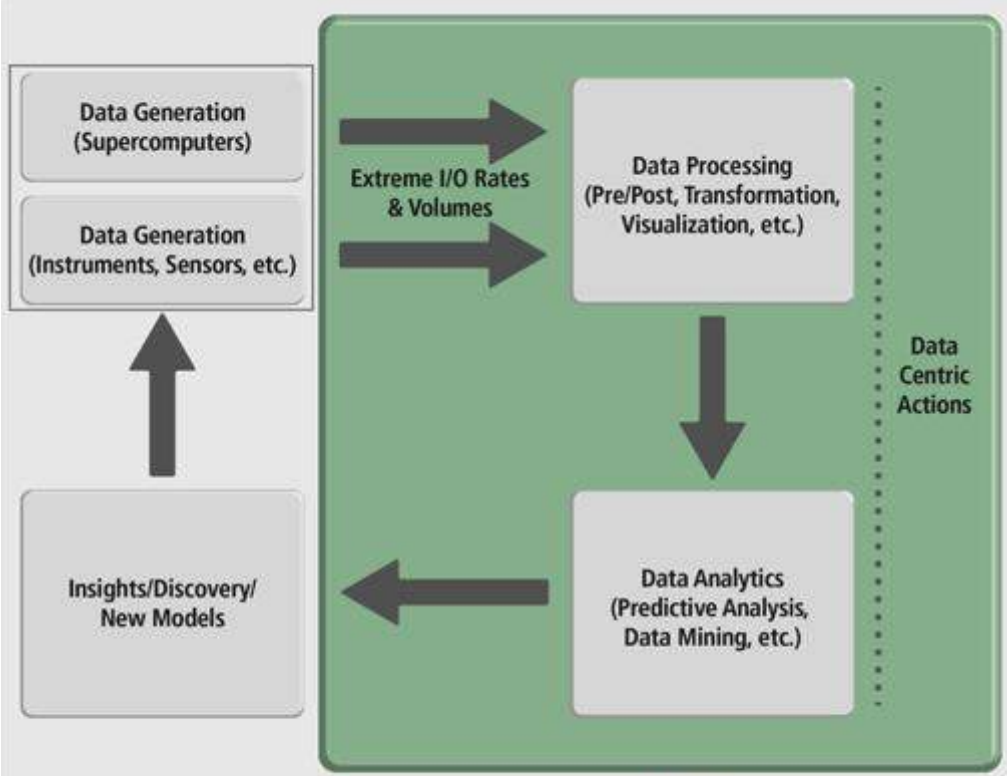


Figure 21:New workflow to be addressed for Exascale Storage and I/O

Focus on energy:

Energy is one of the biggest pain points in HPC, and the cost of energy will be quite prohibitive going by current architectural assumptions. Movement of data within the system contributes to very high-energy consumption. As a rule of thumb, if compute takes 1 picojoule of energy, movement of data to compute takes 100 picojoules of energy[7]. The storage system should be designed and improved so as to address this problem.

One approach that could be taken is the possibility of having in-storage compute, similar to what is done in the data analytics world. The two main innovations required for that within HPC are:

a. Providing more low cost compute capabilities within storage
b. Providing the semantics within the storage software and the applications & workflows to move compute to storage

[6] https://www.skatelescope.org/
[7] [**IDC 4**] Steve Conway and Chirag DeKate (IDC), High Performance Data Analysis HPC meets Big Data, March 2013

Such innovations are already happening in projects such as SAGE. It will be a great opportunity for DEEP-ER to study those innovations and include some of those elements going forward.

Storage software innovations:

Parallel file systems such as Lustre have existed for a long time and they were designed for a time when the architectural assumptions for storage and I/O (and parallelism) were very different. The arrival of extreme parallelism brings us to a point where they will be billions of threads/processes sharing the storage system. Also very strict consistency and transactional semantics such as those imposed by POSIX cannot be the way forward as this would completely stall the compute just waiting for I/O. There should be more flexible ways to access storage not just for parallel I/O access but also for sharing and loading data sets over the cloud, keeping the same storage infrastructure. Also, the storage software should be able to accommodate and manage data split across multiple tiers of storage. There is also a need to flexibly virtualize the storage infrastructure into “containers” when there is data sharing between different workflows. The software infrastructure should also be flexible enough to accommodate storage pools within the compute infrastructure. I/O Quality of Service needs to be provided for the different workflows in a very flexible way.

Object storage technologies show the promise for such flexibility especially considering that any “gateways” can be overlaid on top of them to give them any personality as required by the use cases (cloud storage, scratch storage for HPC, block device, etc).

Object storage APIs also have the promise for flexible addition of third party software plugins and extensions (such as hierarchical storage management, data integrity checks, etc) as evidenced by the SAGE project.

We expect storage hardware to evolve rapidly in the near future. To overcome scalability restrictions file systems, need to leverage the full potential of the hardware. Thus, new technology needs to be evaluated constantly to make necessary changes to the software, in order to fully support its capabilities. Furthermore, systems need to be able to work with different kinds of backend technology to optimally suit different I/O workloads.

In large-scale systems one can expect a lot of simultaneous users and processes and thus a huge amount of concurrent I/O requests. As this is always troubling for the underlying storage hardware and the network interconnects load on the global storage system needs to be minimized. Therefore, on demand burst buffer technologies and local data caching mechanisms should be extended to achieve data locality. This functionality must be supported by common cluster workload managers. Integrating burst buffer technologies into the scheduling system will make it easy to use, while still providing control mechanisms. This will greatly improve the acceptance of such a solution.

3.5 Reliability and Resiliency

SCR was extended during the project to make the most of the available technologies. As a first step, the support for node-local parallel I/O was implemented to take into account the

growing amount of cores in modern MIC architectures. All of the processes running on a single node can now write/read together in/from one single file. In conjunction with the usage of NVMes exploited during the project this gives us a significant performance impact in I/O on the whole application. As SIONlib is the first choice within the project for carrying out parallel I/O operations it was decided to include a call-back functionality into SCR shifting the ability to find missing files to third party libs. In the case of our SCR-SIONlib combo this mechanism can be used to open checkpoint files on buddy nodes or reconstructing it by using an EXTOLL NAM with SIONlib and libNAM, when data is missing during a restart. The advantage of SIONlib buddy checkpointing is to stream own checkpoints to a buddy node directly without writing it first to disk and then reading it again reducing the overhead of this approach.

Flushing and fetching between the local and global storages stays in the responsibility of SCR in this case. This leads to our next improvement, leveraging the ability of BeeGFS to do asynchronous flushes and fetches via a special BeeGFS daemon instead of utilizing the built-in daemon shipping with SCR. Therefore, calls to the BeeGFS API have been integrated into the library.

An analytical model has also been implemented into SCR to determine the optimal checkpoint frequency of any application by a given MTBF. The frequency can be calculated by taking into account the checkpointing time and the actual MTBF.

SCR was our choice for application-based checkpoint/restart because its approach is scale-independent. Checkpointing to node-local memory has the main advantage that by adding more nodes to the setup the checkpointing bandwidth increases linearly. So we will have the same scaling behaviour on an Exascale architecture as on a small cluster. The usage of Buddy-Checkpointing gives us redundancy protecting against failing of one or a few number of nodes. With the help of SIONlib implementing a multi-degree Buddy-Checkpointing solution allowing a node not only to have one but a higher number of buddies, even improves this protection. Future developments in this area will probably need to target the locality of nodes within the system in such a way, that buddy nodes have to be in a different building block so that e.g. rack failures can also be caught with this approach.

In SCR the transfer of node-local checkpoints can be done asynchronously overlapping with computation. In our approach we are using the BeeGFS-BeeOND combination to obtain that goal eliminating the need for a separate SCR daemon running on the nodes which would be otherwise responsible for the transfer.

The integration of the NAM in our storage hierarchy will also give us more flexibility in the future. It could serve as an intermediate storage target, gathering all the checkpointing data from a certain number of nodes and transfer it transparently to the global storage. This even can be done with the XOR-Sets it is able to compute in the current implementation. When starting a new XOR-checkpointing process with a NAM, the NAM could first transfer the old XOR-Set to the global storage, to be able to restart from it, even if the application fails during creation of the new checkpoint.

4 Conclusion

The DEEP-ER project has invested significant effort into analysing and modelling performance aspects of applications, focusing on the compute, I/O and resiliency aspects. The results of this work have been presented and discussed in Section 2 of this Deliverable and in Deliverable D7.1 [2]. Notable achievements are the I/O instrumentation and analysis capabilities introduced into the BSC toolchain, the methodology to assess I/.O efficiency and its impact on other aspects of application performance, and the study of checkpointing frequency and the impact of having fast, local checkpoint storage. Preliminary and unfortunately very limited measurements of actual checkpointing performance are also given in Section 2.3. Regarding power consumption/efficiency, a detailed picture of the power consumption breakdown for the DEEP-ER Booster node was obtained.

Due to the delays in making a reasonably sized DEEP-ER Booster available, it was not possible to perform the required measurements required for meaningful extrapolations to be made.

The data generated in work packages 6 and 7 did allow to identify scalability challenges and their solutions in a qualitative way, as discussed in Section 2.4. In particular, the innovative elements of the DEEP-ER system architecture (fast local storage, the new EXTOLL network and the NAM) do show a beneficial effect on absolute performance and on scalability.

With a view on the progress of relevant technology fields, section 3 discusses how implementations of the DEEP-ER architecture can evolve in the future. Substantial increases in performance and efficiency can be expected from storage-class memory, new PCI Express generations and interconnect fabrics, and of course from the next generations of CPUs. On top of that, I/O and storage subsystems are poised to provide more “intelligence” and local processing capabilities, which ties in nicely with the NAM concept pioneered in DEEP-ER. Finally, a combination of redundant local and global checkpointing/restart mechanisms promises to make this style of resiliency viable for much larger systems than expected in the past.

5 References

1	N.Eicker, H.Ch. Hoppe, J.Gimenez, E.Suarez, I.Zacharov	DEEP-ER Deliverable D7.1: Report on performance extrapolation based on design decisions	2015
2	C. Rosas, J.Giménez, J.Labarta	Scalability prediction for fundamental performance factors, Supercomputing frontiers and innovations, vol. 1, no. 2, pp. 4–19, 2014.	2014
3	J.Labarta, J.Giménez, H-Ch.Hoppe, C. Rosas	Scalability Analysis Methodology, Intel Exascale Labs Report 2015, pp. 46-51, URL: http://www.exascale-labs.eu/Intel%20European%20Exascale%20Labs%20Annual%20Report%202015.pdf	2016
4	V.Beltrán, J.Morillo, J.Bellón, M.Cintra, C.Clauss, N.Eicker, A.Galonska, M.Maroñas, S.Mateo, T.Moschny, A.Peña	DEEP-ER Deliverable D5.3: Resiliency software	2017
5	J.T.Daly	A higher order estimate of the optimum checkpoint interval for restart dumps, in Future Generation Computer Systems, 2006.	2006
6	J.Gimenez, A.Auweter, V.Beltran, U. Brüning, H.-Ch.Hoppe, T.Moschny, E.Suarez	DEEP Deliverable D9.2 - Exascale requirements report	2015
7	R.Mahajan, R. Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik	Embedded Multi-Die Interconnect Bridge (EMIB) – A High Density, High Bandwidth Packaging Interconnect, IEEE ECTC	2016
8	N/A	Intel® Optane™ SSD DC P4800X Series Product Brief, Intel document number 335696-002, URL: http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf	2017
9	B. W. Barrett, R. Brightwell, S. Hemmert, K. Pedretti, K. Wheeler, K. Underwood, R. Riesen, A. B. Maccabe, and T. Hudson	The Portals 4.0 Network Programming Interface, Sandia Report SAND2012-10087, Sandia National Labs	2012

List of Acronyms and Abbreviations

A

ADER-DG: Ader-Discontinuous-Galerkin (numerical scheme)
AoS: Array of Structs
AP: Adams Pass. Intel implementation of the KNL reference board
API: Application Programming Interface
ASIC: Application Specific Integrated Circuit, Integrated circuit customised for a particular use
ATOLL: Predecessor of EXTOLL
Aurora: The name of Eurotech's cluster systems
AVX: Extensions to the x86 instruction set architecture for microprocessors from Intel and AMD covering SIMD/vector instructions up to 128 bit length
AVX2: Intel instruction set extension introducing 256 bit SIMD support
AVX512: Intel instruction set extension introducing 512 bit SIMD support and oher improvements; implemented by second generation Intel Xeon Phi CPUs (KNL) as used in the DEEP-ER Booster

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.
BeeOND: BeeGFS-on-demand, parallel storage based on BeeGFS
BG/Q: Blue Gene Q: Third generation in the Blue Gene series from IBM
BIB: Backplane Interface Board. Electrical and signalling board to provide routing of signals between Reference board and Aurora backplane
BIOS: Basic I/O system. Boot and system initialisation code run before the OS starts
BLAS: Basic Linear Algebra Subprograms
BLN: Brick local network. Used to locally connect the Brick modules
BMC: Board management controller. Used to monitor and manage a compute blade.
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).
BNC: Booster Node Card is a physical instantiation of the BN
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

BSCW: Basic Support for Cooperative Work, Software package developed by the Fraunhofer Society used to create a collaborative workspace for collaboration over the web

C

CAE: Computer Aided Engineering
CD: Corporate Design
CE: Corrected Error
CEPBA: European Centre for Parallelism, Barcelona, Spain
Chassis: Mechanical entity mounted in a rack. A chassis typically aggregates multiple mechanical sub-units (here: Bricks) through a chassis level infrastructure (e.g. backplane, power, cooling)
CI: Corporate Identity
CINECA: Consorzio Interuniversitario, Bologna, Italy
CMC: Corrected Machine Check
CN: Cluster Node (functional entity)
CNR: National Research Council, Italy
CNRS: Centre National de la Recherche Scientifique, Paris, France
Coordinator: The contractual partner of the European Commission (EC) in the project
COM: Computer on-module: form factor for single-board computers
CPLD: Complex programmable logic device: used f.i. for controlling the power-on and pre-boot phases of computer systems.
CP/RS: Checkpoint / Restart
CPU: Central Processing Unit
CRB: Customer Reference Board. An early version of a KNL board developed by Intel
CRESTA: Collaborative Research into Exascale Systemware Tools & Applications: EU-funded Exascale project.
CROW: Eurotech interface board, part of the Aurora KNL node for DEEP-ER.
CTC: Chief Technology Coordinator

D

DAG: Directed acyclic graph.
DDG: Design and Developer Group of the DEEP-ER project
DDP: Dual Die Package
DDR-4: Interface standard to attach DRAM to a CPU
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

DESCA: Comprehensive, modular consortium agreement for the Seventh Framework Programme (FP7)

DFG: Deutsche Forschungsgemeinschaft, German research organisation

DGTD: Discontinuous Galerkin – Time Domain solver

DMA: Direct Memory Access

DoW: Description of Work

DRAM: Dynamic Random Access Memory. Typically describes any form of high capacity volatile memory attached to a CPU

E

E10: Exascale 10. Parallel I/O software developed by a consortium of partners around the EOFS community. Partner Xyratex is responsible for the development needed for the DEEP-ER project.

EATC: European Altair Technology Conference

EC: European Commission

ECC: Error correction code. Corrects errors in storage and transmission systems by added redundancy

EC-GA: EC-Grant Agreement

ECL: ExaCluster Laboratory, A collaboration of Intel, ParTec and JUELICH to develop cluster management software for Exascale computing

EDR: Enhanced Data Rate: Infiniband implementation that delivers 100Gbit/s per link; one of the candidates for the DEEP-ER inter-node intercoinnect

EEP: European Exascale Projects

EESI: European Exascale Software Initiative (FP7)

EMEA: Europe, the Middle East and Africa, Regional designation used for government, marketing and business purposes

ENI: Italian Oil and Gas Company ENI, Italy

EOFS: European Open File Systems

EPiGRAM: Exascale ProGRAmming Models

eQPACE: European project to develop global communications for the QPACE architecture

ESSL: Engineering and Scientific Subroutine Library from IBM

ETP4HPC: European Technology Platform for High Performance Computing

EU: European Union

EUROPLANET: A European Research Infrastructure for Planetary Science (FP7)

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

EXA2CT: EXascale Algorithms and Advanced Computational Techniques

Exaflop: 10^{18} Floating point operations per second

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

EXTOLL: High speed interconnect technology for cluster computers developed by University of Heidelberg

F

FDR: Full data rate: Infiniband implementation that delivers 50 Gbit/s per link; used for the DEEP system

FFT: Fast Fourier Transform

FhGFS: Fraunhofer Global File system, a high-performance parallel I/O system to be adapted to the extended DEEP Architecture and optimised for the DEEP-ER Prototype

FIO: Flexible I/O Tester: set of synthetic benchmarks to measure I/O performance

FLOP: Floating point Operation

FP7: European Commission 7th Framework Programme.

FPGA: Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing

FTI: Fault Tolerant Interface

FWI: Full Waveform Inversion: advanced technique to compute subsurface sound velocity fields from seismic experiment data.

G

GCS: Gauss Centre for Supercomputing, The alliance of the three national supercomputing centres in Germany (Garching, Jülich and Stuttgart)

GEM: Geospace Environment Modelling

GFlop/s: Gigaflop, 10^9 Floating point operations per second

GitHub: Web-based Git repository hosting service

GMRES: Generalized Minimal RESidual method

GPFS: General Parallel File System (GPFS), a high-performance clustered file system developed by IBM

GPU: Graphics Processing Unit

GREEN500 list: Provides rankings of the 500 top most energy-efficient supercomputers in the world

GridMonitor: Part of the ParaStationV5 cluster suite is a versatile system monitor for Linux-based compute cluster

GRS: German Research School for Simulation Sciences GmbH, Aachen and Juelich, Germany

H

H4H: Hybrid programming For Heterogeneous architectures (EU project)

H5hut: Library implementing several data models for particle-based simulations that encapsulates the complexity of parallel HDF5.

HDF5: Hierarchical Data Format: A set of file formats and libraries designed to store and organize large amounts of numerical data

HDR: High data rate: Infiniband implementation by Mellanox that dleivers 200 Gbit/s per link; announced in 2016.

Helmholtz Association: German research organisation

HealthChecker: Part of the ParaStationV5 cluster suite, A test suite to ensure the usability of compute and service nodes within a compute cluster

HMC: Hybrid Memory Cube

HMCC: Hybrid Memory Cube Consortium

HOPSA: HOlistic Performance System Analysis (EU-Russia FP7 project)

Host Module: Self-booting computer board with an Intel KNL CPU, DDR4 memory and a PCI Express root complex. In the Brick Architecture, multiple host modules are connected to each other and NVMe and NIC devices by a PCI Express switch. In the Node Board architecture, a host module provides PCI Express slots to plug NVMe and NIC devices in.

HPC: High Performance Computing

HSW: Haswell, codename for a processor microarchitecture by Intel

HTc: High critical temperature superconductors

HW: Hardware

Hybrid Memory Cube: Novel type of computer RAM that uses 3D packaging of multiple memory dies to increase memory capacity and number of data banks per device area. Technology is being developed by Micron Technology and backed by the Hybrid Memory Cube Consortium.

Hybrid Memory Cube Consortium: Industry association defining HMC interfaces and facilitating HMC Integration into a wide variety of systems. Includes Samsung, Micron Technology, Open-Silicon, ARM, IBM, SK-Hynix, Altera, and Xilinx.

I

I2C: Inter-Integrated Circuit bus. A low cost serial bus used to interconnect silicon devices. Typically used for status monitoring and configuration.

IB: InfiniBand

IBM: International Business Machines Corporation

ICT: Information and Communication Technologies

IEEE: Institute of Electrical and Electronics Engineers

IESP: International Exascale Software Project

Infiniband: Computer-networking communications standard mainly used in high-performnace computing; avalable implementations include FDR, EDR and HDR (announced in 2016).

INFN: Istituto Nazionale di Fisica Nucleare, Italy

Intel: Intel Germany GmbH Feldkirchen,

IP: Intellectual Property

IPC: Instructions Per Cycle

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

IRST: Institute of Scientific and Technologic Research, Trento, Italy

ISC: International Supercomputing Conference, Yearly conference on supercomputing which has been held in Europe since 1986
ITAC: Intel Trace Analyzer and Collector
ITEA-2: Strategic pan-European programme for advanced pre-competitive R&D in software for Software-intensive Systems and Services
ITWM: Institut für Techno- und Wirtschaftsförderung. An Institute of the Fraunhofer Society

J

JUBE: Jülich Benchmarking Environment
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 Phi™.
KNF: Knights Ferry, Intel first available processor based on the MIC
KNL: Knights Landing, second generation of Intel® Xeon Phi™
KPI: Key Performance Indicator
KU Leuven: Katholieke Universiteit Leuven, Belgium

L

LAPACK: Linear Algebra Package, standard software library for numerical linear algebra
LARK: Eurotech interface board, part of the Aurora KNL node for DEEP-ER.
LASA: Leuven Centre for Aero & Space Science, Technology and Applications, Brussels, Belgium
LGFMC: Lattice Green Function Monte-Carlo
libxsmm: Library for small matrix multiplications
LINPACK: Software library to perform numerical linear algebra calculations used as benchmark
LLNL: Lawrence Livermore National Laboratory
LMCC: Leuven Mathematical Modelling & Computational Science Centre, Belgium
LOFAR: Low-Frequency Array, an instrument for performing radio astronomy built by ASTRON
LPC: Low Pin Count bus
Lustre: Type of parallel distributed file system
LYNXvA: First version of the Aurora Blade Root Card.
LYNXvB: Second versiin of the Aurora Blade Root Card.
LYNXvS: First prototype of the Aurora Blade Root Card.

M

MCDRAM: Multi-Channel DRAM, a high bandwidth on package memory on KNL

MCE: Machine Check Exception
Mercurium: Source-to-source compiler of OmpSs, developed by BSC
MEW: Machine Evaluation Workshop
MFlop/s: Megaflop, 10^6 Floating point operations per second
MIC: Intel Many Integrated Core architecture
MIC-OS: Operating System of the MIC architecture
MIUR: Ministry of Education, University and Research, Italy
MKL: Math Kernel Library
MMM@HPC: Project of Multiscale materials modelling on high performance computer architectures
Mont-Blanc: European scalable and power efficient HPC platform based on low-power embedded technology
Mont-Blanc 2: Follow-up project of Mont-Blanc
MPI: Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
MPIX: Blue Gene/Q specific extensions to MPI
MQTT: Message Queue Telemetry Transport protocol

MR-IOV: Multi-root I/O virtualisation. Standard to share a PCI Express endpoint between multiple hosts
MTBF: Mean Time Between Failures

N

NAM: Network Attached Memory, nodes connected to the DEEP-ER network providing shared memory and special-purpose processing to the DEEP-ER DEEP-ER Booster and Cluster nodes.
NAND Flash: Implementation of non-volatile memory used today for solid state disks.
NASA: National Aeronautics and Space Administration, Washington, USA
NEF: Network of European Foundations: name of server where financial data is uploaded to provide it to the EC
NetCDF: Network Common Data Form. A set of software libraries and data formats that support the creation, access, and sharing of array-oriented scientific data
NIC: Network Interface Controller, Hardware component that connects a computer to a computer network
NSF: National Science Foundation, USA
NTB: Non-transparent bridge. A component required to connect PCI hierarchies
NUMA: Non-Uniform Memory Access
Numexas: NUMerical Methods and Tools for Key EXAScale Computing Challenges in Engineering and Applied Sciences
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: interface standard for attaching storage via PCie; also specifies high-level HW interfaces like queues.

O

OEM: Original Equipment Manufacturer. Term used for a company that commercialises products out of components delivered by other companies.

OGS: Institute of Oceanography and Experimental Geophysics, Italy

OmpSs: BSC's Superscalar (Ss) for OpenMP

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

OS: Operating System

OWLS: Eurotech interface board, part of the Aurora KNL node for DEEP-ER.

P

PA: Physical address space. Used on hardware level to access system components

ParaStation Consortium: Involved in research and development of solutions for high performance computing, especially for cluster computing

ParaStation MPI: Software for cluster management and control developed by ParTec

Paraver: Performance analysis tool developed by BSC

Paraview: Open Source multiple-platform application for interactive, scientific visualisation

ParTec: ParTec Cluster Competence Center GmbH, Munich, Germany

PATC: PRACE Advanced Training Centers

PC: Personal Computer

PCB: Printed circuit board.

PCH: Platform controller hub: companion device to operate Intel CPUs and attach commodity peripherals

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.

PCISIG: PCI special interest group. Industry association responsible for the development of the PCI/PCI Express standards

PCM: Phase change memory. A technology candidate for future non-volatile memories

PFlop/s: Petaflop, 10^{15} Floating point operations per second

PFS: Parallel File System

PLL: Phase-locked loop. A control system that generates an output signal whose phase is related to the phase of an input signal. Used to demodulate a signal, recover a noisy signal, frequency synthesis and distribution of precisely timed clock pulses

PLX: Provider of PCI Express system components

PM: Person Month or Project Manager of the DEEP project (depending on the context)
PMT: Project Management Team of the DEEP-ER project
POSIX: Portable Operating System Interface. A family of standards specified by the IEEE for maintaining compatibility between operating systems
PPF: Poly-Phase Filter
PR: Public Relations
PRACE: Partnership for Advanced Computing in Europe (EU project, European HPC infrastructure)
PRACE-1IP: PRACE First Implementation Phase (EU project)
PRACE-2IP: PRACE Second Implementation Phase (EU project)
PRACE-3IP: PRACE Third Implementation Phase (EU project)
Project Coordinator: Leading scientist coordinating and representing the DEEP-ER project
PROSPECT: Promotion of Supercomputing Partnerships for European Competitiveness and Technology (registered association, Germany)
PTC: Persistent Task-based Checkpoint

Q

QCD: Quantum Chromodynamics
QCDOC: Quantum ChromoDynamics On a Chip, special supercomputer developed by Universities of Edinburgh, Columbia and by IBM
QMC: Quantum Monte-Carlo
QPACE: QCD Parallel Computing Engine. Specialised supercomputer for QCD Parallel Computing
QPACE-2: Successor system to QPACE using Intel Xeon Phi co-processors.
QPACE3: Successor system to QPACE-2 using 2nd generation Intel Xeon Phi processors

R

Rack: Compartment to mechanically assemble multiple chassis to form the final computer
RAID: Redundant Array of Independent Discs
RAM: Random-Access Memory
RCM: Reverse Cuthill-McKee renumbering scheme
RDIMM: Registered Dual In line Memory Module
RDMA: Remote Direct Memory Access
RFI: Radio Frequency Interference
RMA: Remote Memory Access. Functional unit implemented in the EXTOLL NIC; used to implement highly optimized one-sided communication
RML: Risk management list used in the DEEP-ER project
ROMIO: High-performance, portable implementation of MPI I/O
RTD: Research and Technological Development
R&D: Research and Development

S

SAS: Serial attached SCSI: point-to-point serial interface for high-performance storage devices commonly used in server computers; covers speeds up to 1500 MByte/s.

SATA: Serial AT Attachment: point-to-point serial interface for storage devices commonly used in desktop and portable computers (with speeds up to 600 MByte/s); latest version covers up to 2000 Mbyte/s, competing with NVMe.

SC: International Conference for High Performance Computing, Networking, Storage, and Analysis, organised in the USA by the Association for Computing Machinery (ACM) and the IEEE Computer Society

Scalasca: Performance analysis tool developed by JUELICH and GRS

SCR: Scalable Checkpoint/Restart. A library from LLNL

SDP: Single Die Package

SDV: Software Development Vehicle: in DEEP-ER, an interim system with 16 Cluster and eight Booster nodes plus three file server nodes used for SW development and benchmarking in DEEP-ER.

SECDED ECC: Single-Error Correcting and Double-Error Detecting Error Correcting Code

SEO: Search Engine Optimisation

SERDES Serializer/Deserializer functional block converting convert data between serial and parallel interfaces; used for data transmission over a single differential line.

SIMD: Single Instruction Multiple Data

SIONlib: Parallel I/O library developed by Forschungszentrum Jülich

SISSA: International School of Advanced Studies, Trieste, Italy

SKA: Square Kilometre Array

SM-Bus: Single-ended simple two-wire bus derived from I2C for the purpose of lightweight communication often used management of computer system components.

SME: Small and Medium Enterprise

SMFU: Shared Memory Functional Unit. Used by the EXTOLL network for mapping remote memory regions.

SNB: Sandy Bridge, codename for a microarchitecture by Intel

SoA: Struct of Arrays

SOL: Serial Over Lan

SOTERIA: Project for the collection, organisation and the use of space physics data aimed at better understanding space weather (EU project)

SPR: Scientific Project Representative of the DEEP-ER Project

SRA: Strategic Research Agenda prepared by ETP4HPC

SRAO: Software Recovery: Action Optional

SRAR: Software Recovery: Action Required

SR-IOV: Single Root I/O Virtualization: enables access to a PCI Express device from multiple virtualized guest operating systems.

SSD: Solid State Disk

SSE: SIMD instruction set extension to the x86 architecture

StarSs: Generic programming environment developed by BSC

STRATOS: PRACE advisory group to foster development of HPC technologies in Europe

SVML: Intel's Short Vector Math Library
SW: Software
SWIFF: Space Weather Integrated Forecasting Framework, Leuven

T

TBR: Task-based resiliency
TCO: Total Cost of Ownership
TEXT: Towards Exaflop Applications (EU project)
TFlop/s: Teraflop, 10^{12} Floating point operations per second
Tier-0, Tier-1: Different classes of supercomputers ordered by their performance
TK: Task, Followed by a number, term to designate a task inside a work package of the DEEP-ER project
TLP: Transaction layer packet. Basic packet structure to transport transactions across a PCI Express infrastructure
ToW: Team of Work Package leaders within the DEEP-ER project
TP10: Third Party under special clause 10
TSV: Thru Silicon Via
TurboRVB: Quantum Monte Carlo Software for electronic structure calculations developed by SISSA

U

UC: Uncorrected Error
UCNA: Uncorrected Error: No Action
UHEI: University of Heidelberg, Germany
ULP: Ultra-low profile: new standard for DIMMs with a maximum height of 17.75 mm; ULP DIMMs are used on the Aurora Blade KNL nodes
UREG: University of Regensburg, Germany
UPC: Universitat Politècnica de Catalunya. Barcelona, Spain

V

VELO: Functional unit and protocol implemented in an EXTOLL NIC; used to implement highly optimized two-sided communication
VI-HPS: Virtual Institute for High Productivity Supercomputing
VF: Virtual function. A functional element of a PCI endpoint
VLP RDIMM: Very Low Profile RDIMM
VMC: Variational Monte-Carlo
VML: Intel's Vector Math Library
VTune: Intel's commercial performance analysis tool

W

WAN: Wide Area Network
WP: Work Package

X

XML: Extensible Markup Language
x86: Family of instruction set architectures based on the Intel 8086 CPU

Y

Z

ZITI Heidelberg: Institut für Technische Informatik Uni Heidelberg, Germany