H2020-FETHPC-01-2016

DEEP-EST

# DEEP Extreme Scale Technologies

Grant Agreement Number: 754304

# D3.3
# Tests for Prototype Assessment

*Final*

**Version:** 1.0

**Author(s):** H.-C. Hoppe (INTEL)

**Contributor(s):** H. Cornelius (MEGWARE), N. Eicker (JUELICH)

**Date:** 28.06.2019

## Project and Deliverable Information Sheet

| | | |
|---|---|---|
| **DEEP-EST Project** | **Project Ref. №:** 754304 | |
| | **Project Title:** DEEP Extreme Scale Technologies | |
| | **Project Web Site:** http://www.deep-projects.eu | |
| | **Deliverable ID:** D3.3 | |
| | **Deliverable Nature:** Report | |
| | **Deliverable Level:** PU* | **Contractual Date of Delivery:** 30 / June / 2019 |
| | | **Actual Date of Delivery:** 28 / June / 2019 |
| | **EC Project Officer:** Juan Pelegrín | |

* - The dissemination levels are indicated as follows: PU = Public, fully open, e.g. web; CO = Confidential, restricted under conditions set out in Model Grant Agreement; CI = Classified, information as referred to in Commission Decision 2001/844/EC.

## Document Control Sheet

| | | |
|---|---|---|
| **Document** | **Title:** Tests for Prototype **Assessment** | |
| | **ID:** D3.3 | |
| | **Version:** 1.0 | **Status:** Final |
| | **Available at:** http://www.deep-projects.eu | |
| | **Software Tool:** Microsoft Word | |
| | **File(s):** DEEP-EST_D3.3_Tests_for_Prototype_Assessment_v1.0 | |
| **Authorship** | **Written by:** | H.-C. Hoppe (INTEL) |
| | **Contributors:** | H. Cornelius (MEGWARE), N. Eicker (JUELICH) |
| | **Reviewed by:** | Z. UL Huda (JUELICH), I. Schmitz (ParTec) |
| | **Approved by:** | BoP/PMT |

## Document Status Sheet

| Version | Date | Status | Comments |
|---|---|---|---|
| 1.0 | 28/06/2019 | Final version | EC Submission |

## Document Keywords

| | |
|---|---|
| **Keywords**: | DEEP-EST, HPC, Exascale, Applications, Co-design, system architecture |

# Table of Contents

## List of Figures

## List of Tables

# Executive Summary

The DEEP-EST prototype system consists of three compute modules (Cluster Module/CM, Data Analytics Module/DAM, and Extremely Scalable Booster/ESB); in addition, a Scalable Service Module (SSSM) provides storage and login services, and the Network Federation (NF) connects the fabrics of CM, DAM, ESB and SSSM. Finally, the Network Attached Memory (NAM) and Global Communication Engine (GCE) components deliver fast access to shared, storage class memory and acceleration of collective MPI operations on the ESB. Establishing that such a complex system actually provides the functionality and performance specified is an important prerequisite for porting and optimising applications. It also gives credibility to the application benchmarking results.

This Deliverable presents a methodology for assembling a set of test and evaluation codes. It discusses a set of such codes mainly targeted at the CM, DAM, and NF, which are either already available or will become so in the next months. It also discusses an initial set of such codes targeted at the ESB and the accelerators present in the DAM, plus the NAM and GCE components. Since ESB, NAM and GCE do break new ground in terms of functionality and capabilities, this initial set will be completed over time, to be fully developed when these components will become available in working prototype form.

Since the DEEP-EST project puts its emphasis on specific innovations (the Modular Supercomputing Architecture, a “lean accelerator node” design for the ESB, and the NAM plus GCE), the proposed set of test and evaluation codes likewise focuses on these, testing their functionality and performance.

The test and evaluation codes will be integrated into the JUBE environment and run as new modules and/or system components become available. Their output will, if required, directly influence work to identify and correct functionality or performance problems. Deliverable D3.4 (Prototype Assessment) at project month 33 will document the results achieved.

# 1 Introduction and Approach

The DEEP-EST prototype system as shown in Figure 1 consists of three compute modules (Cluster Module/CM, Data Analytics Module/DAM, and Extremely Scalable Booster/ESB), a service module providing storage and login services, a Network Federation (NF). Network Attached Memory (NAM) and Global Communication Engine (GCE) components. Details can be found in the updated Deliverable D3.2 [1]. As is clear from the figure, the DEEP-EST prototype is a complex system, and care must be taken to ensure that all parts plus the complete system are working according to specifications. This will provide application developers with a solid platform to optimise their codes, and avoid, to the extent possible, the need to guess whether end-to-end performance issues are due to bottlenecks in the application code or the system.

## 1.1 What will be tested/measured

In this context, extra focus has to be on the main hardware-related innovations in the prototype compared to state-of-the-art in HPC systems, which are not usually covered by the established burn-in and check-out procedures at the system vendor (Megware), and which can materially impact delivered application performance and efficiency.

These can be summarised as

- **Modular architecture:** the system consists of several modules with different thread or node performance characteristics, and possibly different interconnects. The Network Federation (NF) enables fast communication among different modules, bridging the protocols and using the APIs needed by applications. For DEEP-EST, these are the Message Passing Interface (MPI) communication layer, which provides fast inter-process communication, and the Internet Protocol (IP), which serves as a base to implement a wide variety of communication and storage protocols and functionality.

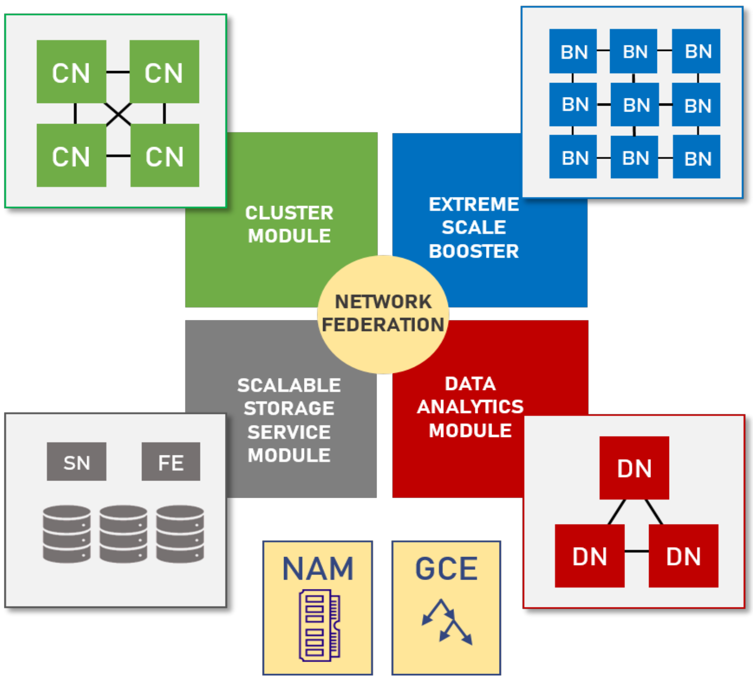


**Figure 1: DEEP-EST Prototype architecture.**

The performance of both must be measured and validated, with the objective being to meet the application requirements for cross-module communication.

- **Large-Memory node with accelerators** for the DAM: the system combines Intel® Xeon® CPUs with a significant amount of cores (24 per socket) with a large amount of byte-addressable, non-volatile memory (2 TByte of Intel® Optane™ DC Persistent Memory [2] per node) and both an NVIDIA Tesla V100 GPGPU and an Intel® Stratix® 10 FPGA with HBM-2 memory.

  DEEP-EST plans to use the persistent memory both as a fast repository for data between workflow steps (using a file system abstraction or direct memory accesses), and as memory for applications with very high memory usage, in which case the installed 384 GByte DDR4 DRAM per DAM node will act as a transparent memory cache.

  Furthermore, the data analytics and machine learning parts of applications will make use of both accelerators, depending on their respective characteristics. Both accelerators use fast HBM-2 locally, and for the application performance, the local memory bandwidth & latency and the bandwidth across PCI Express are important.

  Finally, the DAM will use both a 40 Gbit/s Ethernet and a 100 Gbit/s EXTOLL fabric. The former will be used for I/O, and the key requirements here is that the 16 DAM nodes can actually saturate the I/O bandwidth of the storage module. EXTOLL will be used for accessing the NAM component (see below), so besides assessing MPI performance, the Remote Memory Access (RMA) bandwidth to the NAM has to be measured.

- **Lean accelerator node** design for the ESB: in the update to Deliverable D3.2 [1], the DEEP-EST project has defined a “lean accelerator node” architecture for the ESB nodes; the main innovation here is the use of an efficient, entry-level CPU to host an NVIDIA Tesla V100 accelerator card. The node implements a highly efficient data path between the GPGPU accelerator via PCI Express to the EXTOLL NIC, which provides very close to a full 100 Gbit/s EXTOLL network injection bandwidth. Besides the performance characteristics of the GPGPU, the actually achieved end-to-end MPI latency and bandwidth between two ESB nodes must be measured and analysed.
- **Fast, shared memory resources (NAM)**: the Network Attached Memory (NAM) provides fast access for the DAM and ESB nodes to shared, storage-class memory across the EXTOLL fabric. The API to the NAM is patterned after the MPI one-sided routines, and provides remote memory access and synchronisation functionality as described in [3] and [4]. The most relevant performance parameters are therefore the latency and bandwidth for remote memory accesses, and the overhead of the synchronization primitives.
- **MPI collective operation engine (GCE)**: the raison d'être of the Global Communication Engine (GCE) is to accelerate the execution of MPI collective operations like broadcasts or reductions, measured as time taken from the first process starting the operation to the last process exiting from it. The actual times will be measured for different process counts and compared to an abstract model of the GCE execution which will define the theoretically achievable performance bounds.

In addition, it is important to properly establish standard performance parameters (such as CPU or GPGPU compute throughput) for each module. After all, they are the basis for application developers and users to adapt their codes or use cases. They will also provide a basis for  the decisions of automated resource management and scheduling systems.

## 1.2 Relevant Architecture/System Levels

The validation and measurements will be performed at three distinct levels:

- **Node Level**: CPU, GPGPU and FPGA performance (compute and memory) can be measured on single nodes, without using or requiring the modules or the full MSA prototype to be installed. This also applies for inter-process communication within a node.
- **Module Level**: tests of communication across nodes will of course require at least a number of nodes plus the high-performance fabric to be available and integrated. The early tests can be done on evaluators (like the planned ESB evaluator for ESB node-to-node communication), yet the full picture, in particular regarding network contention and collective operation scaling can only be assessed once the full module is available.
- **System Level**: Any cross-module tests plus all I/O tests to/from the storage module do require the participating modules and all required NF components to be available. A special case here is the DAM, which incorporates a high-speed Ethernet fabric and can therefore perform I/O without the need to have any NF components running.

As far as possible, in Tk3.3  the number of distinct validation and evaluation codes is limited by using a single code for all applicable system levels.

At the time of writing, the storage module and the CM are installed at Jülich, with work ongoing to bring the InfiniBand↔Ethernet gateways up to full bandwidth. Partner Megware is in the process of building up DAM nodes, and the complete DAM hardware is planned to be delivered until end of project month 25. The full software environment installation is planned to be finished one month later. This means that CM node and module-level tests can start after submission of this Deliverable, and initial I/O BW measurements involving the storage module will be possible. In addition, DAM node tests will also be possible at that time, with the availability of the planned accelerators defining when any tests involving these can start. It is also anticipated that first MPI tests on top of EXTOLL can be performed, using the existing EXTOLL TOURMALET NICs instead of the Fabri[3] which is still in development.

To accelerate development of the ESB nodes (in particular the fabric integration), the project will build up and connect two ESB evaluators. Each one includes an existing Intel Xeon server, one NVIDIA Tesla V100, and one EXTOLL TOURMALET PCIe NIC.

## 1.3 Developing New Evaluation Codes vs. Re-Using Existing Ones

The HPC community has created a huge number of test & benchmark codes, ranging from single synthetic benchmarks via simplified versions of actual applications (often called proxy applications) to huge integrated suites (like for instance the commercial Spec suites [5]). Due to the limited amount of effort available for Tk3.3, the significant number of different entities to be tested, and the need to adapt to innovations like the NAM, focus has been on identifying freely available, Open Source codes of manageable complexity and proven value. Another criterion was to have codes that can be run quickly (i.e. in maximum a few hours). Additions/adaptations cannot totally be avoided, like in modifying existing memory benchmarks to use the specific NAM interface developed in WP5.

One important exception here is the testing of the Intel Stratix 10 FPGA – there are currently no commonly accepted & established benchmark codes for FPGAs available, and porting a low-level code (VHDL level) from one FPGA manufacturer to a different one involves significant effort and risk. Going to a higher level of programming (OpenCL being the main candidate) alleviates some of the porting effort. Experience clearly shows that coaxing satisfactory performance out of an OpenCL application on an FPGA can easily require a significant rewrite, due to the special nature of spatial parallelism as exhibited by an FPGA compared to multi-thread or multi-process parallelism as exhibited by CPUs and GPGPUs. We therefore propose to take recourse by using vendor-supplied example codes to test functionality and basic performance.

## 1.4 Managing and Running the Evaluation Codes

In the previous projects DEEP and DEEP-ER, use of the Juelich Benchmarking Environment (JUBE) [6] has been established, and the degree of automation that this environment provides has shown significant benefit, e.g.. in catching performance regressions. WP2 in DEEP-EST is also actively integrating a set of low-level benchmarks and the application codes with JUBE.

WP3 will likewise integrate the selected test codes with JUBE, with the potential exception of the FPGA codes due to the very special handling of compilation and execution on FPGAs. By doing this, automatic invocation of the benchmarks becomes possible, and regular runs to catch any regressions that might be introduced by changes to the system SW can be arranged.

The test codes will be stored in a gitlab at JSC, which of course includes all required adaptations and tweaks for the DEEP-EST prototype system and its modules/components. Should there be any error corrections or extensions of general interests, we will work to make these available for inclusion in future versions.

# 2 Initial Set of Evaluation Codes

Following the approach detailed in section 1 above, an initial set of test and evaluation codes has been defined. This section briefly describes each of these, gives a rationale why a certain code was selected, and cross-references the parts and levels of the DEEP-EST MSA prototype it applies to.

For each test and evaluation code, the energy used to run it on the target platform will be measured and documented, using the most inclusive and precise method. For the CM and ESB, the EnergyMeter hardware component implemented by Megware will be used – this measures the total energy consumption of the complete node, including CPU, memory, accelerator and NIC. For the DAM, we will investigate what the best method would be – challenge here is that the inclusive EnergyMeter is not available.

Factoring in the energy consumed by the fabric infrastructure is an open challenge in general. Network switches (which are for InfiniBand and Ethernet) are shared resources, and it is not possible to exactly ascribe power used while a benchmark is running on part of a system to that benchmark vs. other applications running on the remaining nodes. In addition, power used by the switches does not completely correlate with actual usage of the fabric, f.i. because of keep-alive traffic between the NICs and switches.

Our proposed approach here is to measure switch power use on a fully loaded system (like f.i. running a bisectional bandwidth benchmark from section 2.3.2 below) and compute a “per port” power consumption by dividing the total power consumption by the number of active ports. For each test, we will then count the number of ports used and arrive at an estimate of the fabric power consumption.

## 2.1 Compute Performance/Throughput

Compute performance is usually measured as the number of compute instructions completed in a given time, most often for floating point (Flop/s). The usual method is to run a benchmark code which requires a certain amount of floating-point operations, and then create a Flop/s measure by dividing this number by the time taken to run the benchmark. Codes with a regular or predictable access pattern and significant arithmetic intensity are best suited to assess the peak *achievable* compute throughput.

### *2.1.1 Highly Parallel Computing LINPACK (HPL)*

Since its first publication way back in 1979, the LINPACK benchmark [7] has achieved the position of the de-facto standard to compare the compute performance of HPC systems performance. Both the Top 500 list [8] of the fastest supercomputers and the Green500 list [9] of the most energy-efficient ones are based on an evolved version of LINPACK.

LINPACK solves a dense system of linear equations using Lower-Upper (LU) decomposition with cyclic distribution of blocks and recursive panel factorization. The original versions did define fixed matrix dimensions ($100\times100$ initially), which were updated a couple of times to reflect the growth in system performance and memory capacities, and finally replaced by the Highly Parallel Computing Linpack (HPL) benchmark variant, which lets users set the (square) matrix size to any value suitable for achieving good performance on a given system. Due to the basic problem posed, LINPACK and its variants do test the computational throughput of a system for dense linear algebra, which is amenable to hardware optimisations such as caching and pre-fetching. It is not stressing the memory system capabilities (south of the caches), or the inter-node communication system.

For a given matrix size of $n\times$, the floating-point (double precision) operation count is assumed to be $2/3n^3 + 2n^2$, independent of any algorithmic tweaks.

A portable implementation of HPL is available[1], with the current version being 2.3. That implementation uses the C language, and includes an MPI version for multi-process and multi-node execution. HPL requires an implementation of the Basic Linear Algebra Subprograms (BLAS) [10], which we assume is provided as part of the compilation & SW development tools installation. Multi-threading execution of HPL is usually achieved by using a multi-threaded BLAS system (such as the Intel® Math Kernel Library or Intel® MKL). Vendors are distributing highly tuned versions of HPL, which must comply with the behaviour of LU decomposition and be close to the operation count mentioned above. It is not permissible for vendors to use lower precision (SP) or to use fast matrix algorithms like e.g. Strassen's matrix multiplication [11].

Versions of HPL for systems with accelerators are available, for instance a CUDA version developed and optimised by NVIDIA[2]. Access to this version requires registration with NVIDIA, so partner JUELICH will access and run this benchmark.

HPL is useful to assess single-thread and single-node performance (with an option to use multi-threading in the BLAS libraries or multi-processing using MPI), and of course for the

[1] http://www.netlib.org/benchmark/hpl/

[2] https://developer.nvidia.com/rdp/assets/cuda-accelerated-linpack-linux64

modules (again, offering the choice to mix multi-threading and multi-processing). For the CM and DAM, the Intel-supplied version coming with MKL can be used, and for the ESB, the CUDA version maintained by NVIDIA looks like a good choice. Time permitting, it can be interesting to also run the plain vanilla version of HPL on the CM and DAM, on top of Intel MKL.

### *2.1.2 miniGhost Finite-Element Proxy Application*

The miniGhost proxy application [12] is part of the Mantevo suite developed by Sandia National Labs [13] in 2009. The code was added after the paper on Mantevo [14] and has been used in US DoE procurements like those for the NERSC Trinity system [15].

miniGhost resembles a simple finite element application solving the heat diffusion equation with Dirichlet boundary conditions. It implements a difference stencil across a homogenous three-dimensional domain and contains kernels that compute the stencils at each grid point, handle the exchange of halo (or “ghost”) cells between threads/processes, and sums up the values in the grid to help assess convergence. Correctness checking is possible by a suitable assignment of starting values and a limitation of time steps.

The latest release of miniGhost is version 1.01, and can be found on github[3]. The code is written in Fortran, and it includes versions for a serial run (testing single-thread performance), OpenMP thread parallel runs (testing the compute performance of a complete node, and MPI (testing the performance of a module). The code documentation also mentions an OpenACC version, which might be usable to create a version running on an ESB node. This has to be investigated further.

miniGhost is also part of the SPEC/HPG hardware acceleration benchmark suite [16], which can only be obtained commercially. Should the project decide to obtain this suite, it would open up an alternative route to run miniGhost on the ESB.

miniGhost will be run on the CM and DAM, on the node and module level. In addition, we will measure the single-thread performance running on a single core. It does not seem to make sense to run the code across modules, since it is fully homogeneous, and the different performance characteristics across modules would only cause loss in efficiency.

### *2.1.3 Livermore Compiler Analysis Loop Suite*

The Livermore Compiler Analysis Loop Suite LCALS [17] is a collection of loop kernels based, in part, on historical “Livermore Loops” benchmarks [18]. Different from the older version, the new LCALS suite uses standard C++ 11, with the benchmarked loops mostly written in plain C. LCALS focuses on measuring floating-point throughput of the combination C++ compiler/execution system. For each kernel, several variants are given, which reflect different usages and execution strategies across the US DoE laboratories and the larger HPC community.

The suite contains three subsets of loops:

- Loops representative of actual C/C++ loops found in application codes.
- Very simple & basic loops which test specific compiler optimisations.
- Loops from the original “Livermore Loops coded in C” project.

For the purpose of DEEP-EST, the first subset is the most relevant.

[3] https://github.com/Mantevo/mantevo.github.io/blob/master/download_files/miniGhost_1.0.1.tar.gz

The latest release of LCALS is version 1.0.2[4]. The code contains variants for single-threaded and C++ or OpenMP multi-threaded execution and this can be used to assess both single-core and node performance. It will be run on the CM and DAM at the node level, also testing single-thread performance.

No GPU version of this code is known.

### *2.1.4 Benchmarks for the DAM FPGA*

As discussed in more detail in section 2.2.4, the situation with freely available FPGA compute performance is problematic. There are OpenCL versions of several compute benchmarks (such as HPL[5], yet by and large, these are tailored to execution on GPGPUs. They should and will run on FPGAs, yet would require substantial work to adapt the parallelisation to the capabilities of FPGAs and optimise the code. Since our plan is to use the FPGA mainly for machine learning, it seems questionable whether we should expend the effort required to optimise HPC-oriented compute benchmarks.

## 2.2 Memory Performance

The peak computational throughput (as measured f.i. by HPL) is only achieved by applications with a sufficient arithmetic intensity (number of operations per Byte read/written) and/or cache-friendly memory access patterns. For most of the real-world applications, the performance of the memory subsystem, measured as latency for random (non-cached) accesses and peak bandwidth for large streams of data (which exceed the cache sizes) is a key limiter. Also, the performance of accelerators does depend on the speed of communication with the node CPU(s); since the amount of data exchanged between the CPU and the accelerator across the PCI Express bus is the main limiter, we subsume this issue under memory performance – it's just accesses across the PCIe bus.

This section discusses two commonly accepted benchmarks for measuring memory subsystem bandwidth and random access latencies. It also discusses a benchmark application implementing a memory-bound solver, and finally discusses the situation at the time of writing regarding the FPGA accelerators for the DAM.

### *2.2.1 Sustained Memory Bandwidth – STREAM*

The STREAM benchmark [19] is the industry-standard to measure sustainable (i.e. out of cache) memory bandwidths for CPUs and GPGPUs. It measures four different loops:

- Vector copy: **`A(i) = B(i)`** for each i
- Vector scale: **`A(i) = B(i)*q`** for each i
- Vector Sum: **`A(i) = B(i) + C(i)`** for each i
- Vector triad: **`A(i) = B(i) + C(i)*q`** for each i

The operations are independent across the loop index, and there is no reuse of data in the cache; sizes of the three arrays involved are selected to be at least the maximum of 4× the size of the sum of all last-level caches used in the run, or 1 Million elements. It is possible to tweak the offset between arrays.

[4] https://computation.llnl.gov/projects/co-design/download/lcals-v1.0.2.tgz

[5] https://github.com/davidrohr/hpl-gpu

The latest CPU version[6] of STREAM is 5.10. It includes implementations in both C and Fortran, and supports OpenMP for multi-threading on a node, as well as MPI for running STREAM on multiple processes.

STREAMS will be used to establish memory bandwidths for the CM and DAM nodes; we will use the OpenMP version to scale the number of threads to the number of cores, and likewise measure the effect of having multiple MPI processes, each one running STREAM on a single node. For the DAM, we will also benchmark the non-volatile memory, both in app-direct mode (in which it can be directly addressed) and in memory mode (where the installed DRAM memory will act as a memory-side cache). For the latter, it will be interesting to run two classes of array sizes: one that exceeds the sum of last-level CPU caches, and one that exceeds the amount of DRAM memory cache. Since we do not plan to use any virtual shared memory or RMA functionality between CM or DAM nodes, it does not seem to make sense to perform any multi-node runs.

For the NVIDIA Tesla V100 GPGPUs, we can use either one of two implementations which support GPGPUs:

- CUDA-STREAM[7], derived from work by based on work by Massimiliano Fatica (NVIDIA).
- Babelstream[8] (formerly called GPU-Stream).

Both measure the local GPGPU memory subsystem, which for the V100 is using HBM-2 memory devices.

Babelstream [20] offers a wealth of versions supporting accelerator-capable interfaces like OpenCL and OpenMP 4.5. It could therefore be useful for benchmarking memory bandwidth on the FPGA devices attached to the DAM nodes, see section 2.2.4.

To measure the performance of memory transfers between CPU and GPGPU, which have to pass through the PCI Express connection, a number of small CUDA benchmark codes[9] are available from NVIDIA. They copy a buffer of configurable size between CPU and GPGPU memory, and the difference is the use of normal (i.e. pageable) CPU memory or pinned CPU memory. Since transfers to/from pageable memory behind the curtains are staged through pinned memory, the latter method should yield better results.

On the DAM nodes, we will run both variants of memory transfers with a suitable range of buffer sizes. The same will be done for the ESB nodes yet there the “lean node” concept foresees avoidance of host CPU to GPGPU transfers, with direct RMA-style transfers to GPGPU memory from other ESB nodes dominating the traditional CPU↔GPGPU offload transfers.

### *2.2.2 HPC Challenge Random Access*

Important classes of HPC workloads access memory in irregular, seemingly or indeed random patterns, with accesses typically being for small data objects. An example from DEEP-EST is the NEST code, and many graph, sparse linear algebra and algebraic multigrid codes have the same characteristics.

[6] http://www.cs.virginia.edu/str/eam/FTP/Code

[7] https://github.com/bcumming/cuda-stream.

[8] https://uob-hpc.github.io/BabelStream/

[9] https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/

The Random Access benchmark [21] of the HPC Challenge suite indexes a large table (> ½ of the system's total memory) using the most significant bits of a stream of pseudo-random 64-bit integers, and updates the corresponding entries by XORing with the random number. This requires the system to read the 64-bit tale entries, XOR a random value over it, and store the data again, with no spatial or temporal reuse of data. The number of such table updates per seconds is reported as the figure of merit (in units of GUPdates/s or GUPs).

The benchmark source code of Random Access is combined with the other HPC Challenge benchmarks[10].The HPC Challenge website[11] contains some (outdated) results and more information about the benchmark. The code can run sequentially, multi-threaded using OpenMP, or in a multi-process version which implements a distributed table and uses MPI to access foreign table entries.

We will run the Random Access benchmark on the CM and DAM nodes, using OpenMP and MPI to scale the number of threads to the natural limit. Given the nature of the DEEP-EST applications, it does not seem to make much sense to run the benchmark in MPI mode across multiple nodes, yet it would certainly be possible.

We are currently looking for a version of this benchmark which runs on a GPGPU memory. A port of the sequential version to CUDA looks feasible,

### *2.2.3 High Performance Conjugate Gradient – HPCG*

While the HPL benchmark (see section 2.1.1 above) has certainly achieved a dominating position, it continues to be criticised due to its inherent limitations: HPL does not stress the memory subsystem or the interconnect fabric, and performance depends on the dense compute throughput out of cache. The High Performance Conjugate Gradient (HPCC) benchmark [22] has been proposed to complement HPL, since its performance depends on the achievable out-of-cache memory bandwidth and the network performance.

The HPCG benchmark solves an elliptic partial differential equation on a 3D grid with a regular 27-point discretization. It uses preconditioned conjugate gradient method (CG) to solve the intermediate systems of equations and incorporates a local and symmetric Gauss-Seidel preconditioning step that requires a triangular forward solve and a backward solve. A synthetic multi-grid V-cycle is used on each preconditioning step to make the benchmark better correspond to real-world applications. HPCG implements matrix multiplication locally, with an initial halo exchange between neighbouring processes. The benchmark exhibits irregular accesses to memory and fine-grain recursive computations that play a role in many scientific HPC applications.

The source code for HPCG and binaries for NVIDIA GPGPUs is available from a web site[12]. For Intel x86 platforms, source code and binaries of an optimised version are available as part of the Intel® Math Kernel Library package [23]. The CUDA HPCG implementation is described in detail in [24].

HPCG has been picked apart and its performance analysed by a number of researchers – one representative paper is [25]. From this work, it is clear that the arithmetic intensity of the benchmark is low (0.25 operations/Byte), and therefore performance is governed by the behaviour of the memory subsystem.

[10] http://icl.cs.utk.edu/projectsfiles/hpcc/download/hpcc-1.5.0.tar.gz
[11] https://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/
[12] https://www.hpcg-benchmark.org/software/index.html

We will run HPCG on the nodes of the CM, DAM and ESB (NVIDIA version, of course) to assess the behaviour of these nodes for a memory-intensive numerical workload. In addition, we will run HPCG across multiple nodes of the CM and DAM, and if possible, on the ESB to test the fabric with a very communication heavy workload.

### *2.2.4 Benchmarks for the DAM FPGA*

The situation regarding freely available benchmarks for current FPGA architectures, such as the Intel® Stratix® 10 FPGA, which will be a part of the DAM nodes, is a bit problematic. Several OpenCL benchmark suites are available commercially, with the SPEC/HPG hardware acceleration benchmark suite [16] being the most suitable. Besides availability of benchmark sources, differences between FPGA families and generations are comparatively large (compared to CPUs, for instance), and obtaining satisfactory efficiency does regularly need significant OpenCL code adaptations, putting the relevancy of portable OpenCL benchmarks into question.

Two different free implementations of the STREAM benchmark have been created with FPGAs as execution platforms in mind:

- Babelstream[13] [20] (formerly called GPU-Stream) contains an OpenCL version.
- MP-Stream[14] [26] is an OpenCL implementation of the STREAM benchmark, with versions for Intel/Altera FPGAs (previous generations), CPUs and NVIDIA GPGPUs.

Both versions could be used for benchmarking the memory bandwidth of the Stratix 10 FPGA which will be part of the DAM nodes. At the time of writing, discussions with Intel's internal FPGA experts were ongoing on this issue.

To measure CPU↔FPGA data transfer performance, as well as random access performance, we could use the Spec suite mentioned above, if the project is willing to invest in a license. As an alternative, we could use Intel-specific low-level benchmarks – a topic which is being discussed at the time of writing.

## 2.3 Communication Performance

The basic performance metric for point-to-point communication is of course the time it takes to transmit a given number of Bytes (*N* Bytes), with this number being highly use case and application specific. The test and evaluation codes discussed in this section measure this basic metric for a variety of values of N. It is usual practice to fit a latency/bandwidth model, which assumes that the time to transmit N Bytes is given as $N/BW + l$, with $l$ being called latency, and *BW* being the asymptotic bandwidth. These two parameters are then taken to describe the point-to-point performance of a network fabric.

At least for the inter-process communication using the Message Passing Interface (MPI), it is important to also measure the duration of various collective communication schemes, such as broadcast, gather/scatter and reduction operations. For HPC applications, these collective operations can take substantial parts of a workload runtime, whereas IP and I/O multi-cast communication schemes are usually less relevant.

[13] https://uob-hpc.github.io/BabelStream/
[14] https://github.com/waqarnabi/mp-stream

In the following sections 2.3.1 and 2.3.2, we describe two benchmarks that will serve to document the MPI performance of the nodes, modules and the full system, and help to diagnose any problems.

Section 2.3.3 discusses the need for measuring Remote Memory Access (RMA) performance between ESB nodes and to the NAM component, and section 2.3.4 details the well-established IOR benchmark, which will be used to assess I/O performance to the storage module, and potentially also to persistent memory in the DAM. Finally, section 2.3.5 covers the iPerf3 benchmark which we plan to use to establish TCP and UDP protocol performance, according to the needs of applications.

### *2.3.1 MPI Linktest*

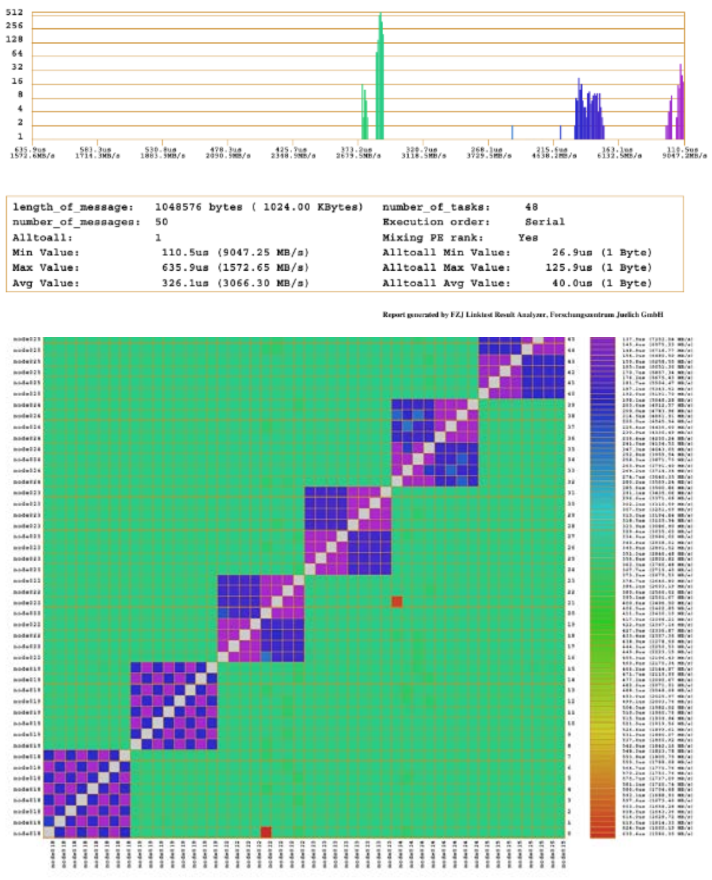
The MPI Linktest benchmark [27] has been developed by partner JUELICH and was used to good advantage in the DEEP and DEEP-ER projects. Linktest conducts parallel ping-pong send/receive tests between pairs of processes, iterating to test all possible pairs of sender and receiver process. Outputs are the latency and bandwidth measured for each pair, which can be presented as a communication matrix (bottom of Figure 2), and in various histograms showing groupings performance values (top of Figure 2). The tool can also analyse latency and bandwidth as a function of distance between the communication partners, which can come in handy for benchmarking EXTOLL fabrics.

The current version of Linktest[15] is version 1.2p1.

[15] https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/_node.html

In DEEP-EST, we will use Linktest at the module level, testing all node-to-node connections in one benchmark run. Varying the number of MPI processes per node will modify the injection rate into the network fabric and enable a very intuitive comparison of intra-node and inter-node messaging performance.

Linktest can also be used to test the communication between multiple modules. However, care



```
length_of_message:   1048576 bytes ( 1024.00 KBytes)   number_of_tasks:      48
number_of_messages:  50                                Execution order:      Serial
Alltoall:            1                                 Mixing PE rank:      Yes
Min Value:            110.5us (9047.25 MB/s)           Alltoall Min Value:      26.9us (1 Byte)
Max Value:            635.9us (1572.65 MB/s)           Alltoall Max Value:     125.9us (1 Byte)
Avg Value:            326.1us (3066.30 MB/s)           Alltoall Avg Value:      40.0us (1 Byte)
```

**Figure 2: Example output of the Linktest tool.**

has to be taken to restrict the number of communicating processes to reflect the actual communication patterns of applications; all ESB processes communicating to the much smaller DAM is not a realistic use case, for instance, and will certainly overwhelm the provided number of NF gateways.

### 2.3.2 *Intel® MPI Benchmarks*

The Intel® MPI Benchmarks (IMB) [28] were started in the early 2000s as the Pallas MPI Benchmarks. The development and distribution was taken over by Intel after the acquisition of Pallas in 2003.

IMB contains several different benchmarks – for DEEP-EST, the most relevant are:

- **IMB-MPI1**: performing point-to-point and collective communication tests using MPI-1 functionality.
- **IMB-EX**: performing tests of the one-sided communication functionality in MPI-2.

- **IMB-RMA**: remote Memory Access (RMA) benchmarks using passive target communication to measure one-sided communication (MPI-3 feature).

The first one will be used to test key collective operations (broadcast, gather/scatter, reduce and allreduce) on each of the DEEP-EST prototype modules. The benchmarking of collective performance on the ESB with and without the GCE is of particular interest here.

IMB-MPI1 will also be used to establish bisection bandwidth for each of the modules, and we will adapt it to drive cross-module tests with realistic assumptions (like only a subset of nodes on the ESB and CM communicating across the NF). Finally, IMB-MPI1 provides a very quick way to measure point-to-point communication between only a few pairs of MPI processes. Once the GCE components become available, IMB-MPI1 will be used to measure their performance.

IMB-EX and IMB-RMA are mainly useful to benchmark the NAM component, which will make its shared memory available using MPI-2 and potentially MPI-3 methods [4]. It is expected that we will need to adapt these benchmarks to work for the NAM.

The latest version of IMB[16] is the 2019 Update 3 release. The benchmarks are written in plain C and work with any standard-compliant MPI implementation. The code is made available under version 1.0 of the Common Public License, giving users the right to perform arbitrary modifications to the source code.

### *2.3.3 RMA Performance between ESB Nodes and to the NAM*

The scalability of the ESB architecture critically depends on maximizing achievable bandwidth between ESB nodes; plan of attack as documented in the updated Deliverable D3.2 [1] is to use the Remote Memory Access (RMA) features of the EXTOLL fabric, and integrate them with NVIDIA's GPUDirect RDMA technology to enable direct memory accesses by the EXTOLL NIC (either the TOURMALET card or the new Fabri$^3$ implementation). This way the ESB node CPU will not be involved in any volume data transfer between the fabric and the GPGPU memory, and any communication overhead within the node (across the PCI Express connection) will be minimized.

The effect of the above integration will be measured indirectly using the MPI test and evaluation codes described in sections 2.3.1 and 2.3.2 above, since the ParTec ParaStation MPI implementation as developed in WP5 will use this direct RMA mechanism for bulk data transfers. All measurements will, however, include the time taken by the higher-level MPI SW layers, and this time can be substantial. Direct measurements of RMA performance between ESB nodes will serve to validate the EXTOLL integration with the ESB node. The most important metrics will be the achieved bandwidth and the random-access latency.

We plan to re-purpose the memory benchmarks described in sections 2.2.1 and 2.2.2, adapting the codes where necessary to properly allocate and register memory segments within the GPGPU memory of two ESB nodes in a way that enables measuring the streaming bandwidth between them and the random access performance of one node accessing memory in the second node. The exact nature and amount of changes required will be discussed with partner EXTOLL once a first prototype version of the EXTOLL integration using the ESB evaluator is available (see the end of section 1.2).

[16] https://github.com/intel/mpi-benchmarks

The network attached memory (NAM) component will provide access to shared, storage-class memory over the EXTOLL fabric. The objective is to optimise access latencies and bandwidth to be close to the EXTOLL fabric limits, by using fast FPGA processing of data and caching in DDR4 memory. Details about the NAM design can be found in [3]. The primary means of access to the memory is the low-level *libnam* library [3], and support will also be provided to use the NAM via MPI-2/MPI-3 one-sided communication interfaces [4]. At the time of writing, the definition of *libnam* was still very much in flux, and the extent of MPI support was also under discussion.

Task Tk3.3 plans to adapt at least one of the following benchmark sets to drive tests of the RMA performance between DAM and ESB nodes[17] and the NAM:

- **STREAMS and Random Access**: these two benchmarks (detailed in sections 2.2.1 and 2.2.2) test and measure regular accesses to local memory. Should it be possible to map remote memory segments on the NAM into a DAM or ESB node memory, they could be adapted to perform the required allocations and registration, and then in effect test streaming data from local to remote memory in the NAM (and vice versa), or randomly accessing this remote memory.
- **IMB-EX/IMB-RMA**: depending on the level of one-sided functionality to be implemented in WP5, these benchmarks could serve to measure the end-to-end performance when accessing the NAM as seen from applications.

How to apportioning the energy used by the NAM to benchmark applications running on it is an open issue right now – it might be possible to measure an energy increase caused by running such stress tests, yet that leaves the question of how to treat the energy used by the NAM in idle mode open.

### 2.3.4 HPC IO Benchmark – IOR

In the DEEP-EST MSA prototype, permanent storage is provided by the “Scalable Storage and Service” Module (SSSM), which uses a 40 Gbit/s Ethernet fabric and is connected to the other modules by Network Federation (NF) gateways. Clearly, the achievable I/O performance from/to the SSSM has to be measured from the CM, DAM and ESB.

The Interleaved or Random (IOR) benchmark [29] was originally developed by Lawrence Berkeley National Laboratory to measure the performance of parallel file systems using a variety of I/O interfaces.

IOR[18] is in active development, with the latest release at the time of writing being version 3.3. This version includes additional support for the BeeGFS file system [30] by partner FHG-ITWM, which will be used for the DEEP-EST prototype. IOR supports the Posix I/O interface, requires MPI for inter-process synchronisation, and can also benchmark the MPI-IO I/O interface, if so required.

We will use IOR to systematically test the I/O performance between each of the DEEP-EST modules and the SSSM. These tests will adopt typical I/O buffer sizes as used by the applications in WP1, and scale the number of IOR processes running on the module, both by using additional module nodes and by running multiple IOR processes per node. The most

[17] ESB and DAM share a common EXTOLL fabric, which does not include the CM. Implementation of libnam heavily uses EXTOLL-specific functionality, which cannot be guaranteed to be supported by the NF connecting the CM to the DAM or ESB.
[18] https://github.com/hpc/ior

important performance objective is to demonstrate that we can indeed saturate the read and write bandwidth of the SSSM from any of the modules.

Several of the modules (such as the CM and DAM) do include local storage-class memory (NVMe SSDs in the case of CM and DAM, and byte-addressable non-volatile memory in the form of Intel® Optane™ DC Persistent Memory [2] for the DAM), which can host local BeeGFS instances. We plan to benchmark the SSDs using BeeGFS and IOR; the non-volatile memory can be used as a storage target (as for instance demonstrated by the NEXTGenIO project [31]); yet it is as of now unclear whether BeeGFS will be adapted to target it within the project timescale.

The network attached memory (NAM) will not provide an I/O-style interface – NAM benchmarks will use memory accesses as described in section 2.3.3.

### *2.3.5 IP Benchmarking – iPerf3*

The Internet Protocol (IP) is the lingua franca between computer systems and many devices; most often, applications utilize the higher-level User Datagram Protocol (UDP) or Transport Control Protocol (TCP). In DEEP-EST, I/O traffic to/from the SSSM and system software or programming models like MPI and Spark will be using UDP or TCP. Since the NF gateways for InfiniBand and EXTOLL fabrics to/from Ethernet will carry IP protocol, we have to measure their performance characteristics. In addition, it will be important to know the actual performance of the 40 Gbit/s Ethernet fabric in the DAM module.

One of the standard benchmarks for UDP and TCP is iPerf [32] [33], which implements a client/server paradigm (*n* benchmark clients communicating with one server), and supports a wide range of configuration parameters. We will match the key parameters to correspond to the actual use by DEEP-EST system SW and applications.

The iPerf3[19] development is ongoing, with version 3.6 as the latest release. iPerf3 uses a three-clause BSD license, and developed by ESnet (part of the US Energy Sciences Network) and Lawrence Berkeley National Laboratory.

To test the NF gateways, we will run pairs of iPerf3 servers and clients between the CM and ESB on one side and the SSSM on the other side. This will require installation and running iPerf3 on the SSSM nodes, which should pose no technical problem, yet will need to be coordinated with the regular use of the system, since for the benchmarks, we should keep any I/O traffic quiet. Single-pair measurements can and have been done using the gateway evaluators.

For the DAM, we plan to measure pairs of client and server, scaling up the processes per client and server node until we see saturation, and then dividing the DAM into two halves, one of them running the client, and one running the server processes.

## 2.4 Summary Table

Table 1 summarises the benchmarks identified, their use in DEEP-EST, and any open issues (such as adaptation or porting) required.

[19] https://github.com/esnet/iperf

**Table 1: Summary of benchmarks identified in this Deliverable**

| To be measured | Benchmark | Description | Modules | System | NAM/ GCE | Comments |
|---|---|---|---|---|---|---|
| **Compute Performance** | HPL | Dense LA | CM, DAM, ESB | N/A | N/A | GPGPU and FPGA ports available |
| | miniGhost | Finite-difference stencils | CM, DAM, ESB | N/A | N/A | GPGPU version part of commercial suite |
| | LCALs | Loop vectorisation | CM, DAM | N/A | N/A | |
| **Memory Performance** | STREAM | Sequential contiguous out of cache accesses | CM, DAM, ESB | N/A | NAM* | Versions for GPGPU and FPGA available; simplified benchmark for PCIe performance to GPGPU and/or FPGA |
| | Random Access | Random DRAM access | CM, DAM, ESB† | N/A | NAM* | Looking for ESB version, adaptations required for NAM |
| | HPCG | Memory bound CG+MG code | CM, DAM, ESB | N/A | N/A | No version for FPGA |
| **Communication Performance** | Linktest | Latency/BW for all node/process pairs | CM, DAM, ESB | Yes | N/A | |
| | IMB | Versatile MPI benchmark | CM, DAM, ESB | Yes | NAM, GCE | One-sided performance benchmarks for NAM, collective benchmarks for GCE |
| | IOR | I/O benchmark | CM, DAM, ESB | Yes | N/A | Benchmark can target local SCM as well as disk on SSSM |
| | IPerf3 | IP benchmarks | CM, DAM, ESB | Yes | N/A | IP BW to SSSM, selected pairs of modules as required by applications |

*: benchmark to be adapted to use NAM API

†: looking for a GPGPU version; straight port could be feasible

# 3 Summary and Outlook

This document discusses the innovative elements on the DEEP-EST MSA prototype architecture, identifies the key performance metrics that have to be tested to assess the actual prototype implementation, and discusses a number of readily available test and evaluation codes as well as their intended usage. It also identifies open questions (mainly related to testing the FPGA-related performance metrics) and discusses adaptations to the existing benchmark codes required to test some of the project's innovations (like the NAM).

During the next three months of the project, T3.3 will work with the other partners (in particular, with EXTOLL and ParTec) to define the scope of adaptations and come up with a design for these. In addition, the test codes for measuring the CPU to FPGA performance and the internal FPGA to memory speed will be defined, working closely with Intel's FPGA experts.

Since the CM is already installed and running at the time of writing, the test codes related to it will be ported and run on that platform next. The delivery of the DAM is expected at project month 25, and the related test codes will be ported and run on that platform after its installation. Once the ESB evaluator is available, and the initial integration of EXTOLL is functional, work on the RMA performance tests can progress, followed by the MPI performance tests. Testing the NAM and GCE will need to wait until the Fabri3 implementation is available, and the ESB can be tested once a first, functional version has been installed.

# 4 Bibliography

[1] H.-C. Hoppe and H. Cornelius, “Deliverable D3.2 -- Update: High-Level System Design,” DEEP-EST Project Consortium, Juelich, 2019.

[2] Intel Corporation, “Intel® Optane™ DC Persistent Memory,” 2019. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html. [Accessed 12 June 2019].

[3] M. Nuessle, A. Giese, S. Kapferer and H. Cornelius, “Deliverable D4.3: Network Federation, Fabri3, NAM and GCE designs,” DEEP-EST Project Consortium, Juelich, 2018.

[4] V. Beltran, J. Ciesko, R. Clauß, K. Keller, L. Bautista, J. Roob, P. Reh, L. Montigny and B. Steinbusch, “Deliverable D6.2: Prototype Programming Environment Implementation,” DEEP-EST Project Consortium, Juelich, 2019.

[5] Standard Performance Evaluation Corporation, “SPEC's Benchmark,” 2019. [Online]. Available: https://www.spec.org/benchmarks.html. [Accessed 21 June 2019].

[6] Juelich Supercomputing Centre, “JUBE Benchmarking Environment,” 2019. [Online]. Available: https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/JUBE/_node.html. [Accessed 12 June 2019].

[7] J. Dongarra, P. Luszczek and A. and Petitet, “The LINPACK Benchmark: past, present and future,” *Concurrency and Computation: Practice and Experience,* vol. 15, no. 08, pp. 803-820, 2003.

[8] Prometeus GmbH, “Top 500 List,” 2019. [Online]. Available: https://www.top500.org. [Accessed 12 June 2019].

[9] Prometeus GmbH, “The Green 500 List,” 2018. [Online]. Available: https://www.top500.org/green500/. [Accessed 12 June 2019].

[10] Netlib, “BLAS (Basic Linear Algebra Subprograms),” 2017. [Online]. Available: http://www.netlib.org/blas/. [Accessed 12 June 2019].

[11] V. Strassen, “Gaussian elimination is not optimal,” *Numerische Mathematik,* vol. 13, no. 4, pp. 354-356, 1969.

[12] R. F. Barrett, V. C. T. and M. A. Heroux, “MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing,” 2012. [Online]. Available: http://spec.cs.miami.edu/accel/Docs/miniGhost.v1.0.pdf. [Accessed 12 June 2019].

[13] M. A. a. D. D. W. a. C. P. S. a. W. J. M. a. E. H. C. a. W. A. Heroux, M. Rajan, E. R. Keiter, H. K. Thornquist and R. W. Numrich, “Improving Performance via Mini-applications,” Sandia National Laboratories, Albuquerque, 2009.

[14] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist and R. W. Numrich, “Improving Performance via Mini-applications,” 2009.

[15] National Energy Research Scientific Computing Center (NERSC), “NERSC-8 / Trinity Benchmarks,” 29 April 2016. [Online]. Available: https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/. [Accessed 21 June 2019].

[16] Standard Performance Evaluation Corporation, “SPEC/HPG hardware acceleration benchmark adds suite to measure OpenMP application performance,” 2017. [Online]. Available: http://spec.cs.miami.edu/accel/press/accelv12release.html. [Accessed 12 June 2019].

[17] R. D. Hornung and J. A. Keasler, “A Case for Improved C++ Compiler Support to Enable Performance Portability in Large Physics Simulation Codes,” Lawrence Livermore National Laboratory, Livermore, 2013.

[18] F. H. McMahon, “The Livermore FORTRAN Kernels: A Computer Test of the Numerical Performance Range,” Uiversity of California, Lawrence, 1986.

[19] J. D. McCalpin, “STREAM: Sustainable Memory Bandwidth in High Performance Computers,” 1995. [Online]. Available: http://www.cs.virginia.edu/stream/. [Accessed 12 June 2019].

[20] M. Martineau, S. McIntosh-Smith, J. Price and T. Deakin, “Evaluating Attainable Memory Bandwidth of Parallel Programming Models via BabelStream,” *International Journal of Computational Science and Engineering,* vol. 1, no. 1, 2017.

[21] V. Aggarwal, Y. Sabharwal, R. Garg and P. Heidelberger, “HPCC RandomAccess benchmark for next generation supercomputers,” in *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009.

[22] J. Dongarra, M. Heroux and P. Luszczek, “High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems,” *International Journal of High Performance Computing Applications,* vol. 30, no. 8, 2015.

[23] Intel Corporation, “Overview of the Intel Optimized HPCG,” 23 May 2019. [Online]. Available: https://software.intel.com/en-us/mkl-linux-developer-guide-overview-of-the-intel-optimized-hpcg. [Accessed 12 June 2019].

[24] E. Phillips and M. Fatica, “A CUDA Implementation of the High Performance Conjugate Gradient Benchmark,” in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, Springer International Publishing, 2015, pp. 68-84.

[25] V. Marjanovic, J. Gracia and C. W. Glass, “Performance Modeling of the HPCG Benchmark,” in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, Springer International Publishing, 2015, pp. 172-192.

[26] S. W. Nabi and W. Vanderbauwhede, “MP-STREAM: A Memory Performance Benchmark for Design Space Exploration on Heterogeneous HPC Devices,” in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018.

[27] Juelich Supercomputing Centre, “LinkTest: Parallel MPI PingPong Test,” 2018. [Online]. Available: https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/_node.html. [Accessed 12 June 2019].

[28] Intel Corporation, “Introducing Intel® MPI Benchmarks,” 2019. [Online]. Available: https://software.intel.com/en-us/articles/intel-mpi-benchmarks. [Accessed 12 June 2019].

[29] H. Shan, K. Antypas and J. Shalf, “Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark,” in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Austin, TX, 2008.

[30] F. Herold and S. Breuner, “An introduction to BeeGFS®,” 2018. [Online]. Available: http://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf. [Accessed 12 June 2019].

[31] The University of Edinburgh, “Next Generation I/O for the Exascale,” 2019. [Online]. Available: http://www.nextgenio.eu/. [Accessed 12 June 2019].

[32] A. Tirumala, L. Cottrell and T. Dunigan, “Measuring End-To-End Bandwidth with Iperf Using Web100,” in *Passive and Active Monitoring Workshop (PAM 2003)*, San Diego, CA, 2003.

[33] ESnet, “iperf3,” 2018. [Online]. Available: https://software.es.net/iperf/. [Accessed 24 June 2018].

[34] H. Cornelius and A. Auweter, “Deliverable D4.1: Prototype Hardware Design,” DEEP-EST Consortium, Juelich, 2018.

[35] Wikipedia, “bfloat16 floating-point format,” 19 June 2018. [Online]. Available: https://en.wikipedia.org/wiki/Bfloat16_floating-point_format. [Accessed 24 June 2018].

[36] G. Slavova, “Introducing Intel® MPI Benchmarks,” 7 February 2018. [Online]. Available: https://software.intel.com/en-us/articles/intel-mpi-benchmarks. [Accessed 24 June 2018].

[37] J. Dongarra, “Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report),” University of Tennessee, Knoxville, 2014.

[38] N. Eicker, T. Moschny and C. Clauss, “Deliverable D5.2 - Update: Software specification,” DEEP-EST Consortium, Juelich, 2019.

[39] H. Cornelius and A. Auweter, “Deliverable D4.1 - Update: Prototype Hardware Design,” DEEP-EST Consortium, Juelich, 2019.

## List of Acronyms and Abbreviations

### A

| | |
|---|---|
| **AI:** | Artificial Intelligence |
| **API:** | Application Programming Interface |
| **ASTRON:** | Netherlands Institute for Radio Astronomy, Netherlands |
| **AVX:** | Advanced Vector Extensions |

### B

| | |
|---|---|
| **BLAS:** | Basic Linear Algebra Subprograms, a specification of linear algebra routines implementing vector and matrix operations |
| **BN:** | Booster Node (functional entity) |
| **BoP:** | Board of Partners for the DEEP-EST Project |
| **BSC:** | Barcelona Supercomputing Centre, Spain |
| **BW:** | Bandwidth |

### C

| | |
|---|---|
| **CERN:** | European Organisation for Nuclear Research / Organisation Européenne pour la Recherche Nucléaire, International organisation |
| **CM:** | Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core |
| **CMS:** | Compact Muon Solenoid experiment at CERN’s LHC |
| **CMSSW:** | Physical toolset software for the CMS experiment at CERN |
| **CN:** | Cluster Node (functional entity) |
| **CNN:** | Convolutional Neural Networks |
| **CPU:** | Central Processing Unit |
| **CRC:** | Cyclic Redundancy Check |

### D

| | |
|---|---|
| **DAM:** | Data Analytics Module: with nodes (DN) based on general-purpose processors, a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications |
| **DBSCAN:** | Density-based Spatial Clustering for Applications with Noise |
| **DDG:** | Design and Developer Group of the DEEP-EST Project |
| **DDR:** | Double Data Rate |
| **DEEP:** | Dynamical Exascale Entry Platform (project FP7-ICT-287530) |
| **DEEP-ER:** | DEEP - Extended Reach (project FP7-ICT-610476) |
| **DEEP-EST:** | DEEP - Extreme Scale Technologies |

| | |
|---|---|
| **DIMM:** | Dual In-line Memory Module |
| **DLMOS**: | A Deep Learning Model of the Solar Wind to forecast the plasma conditions at the orbit of the Earth from images of the Sun developed by KU Leuven |
| **DN:** | Nodes of the DAM |
| **DNN:** | Deep neural network |
| **DRAM:** | Dynamic Random Access Memory. Typically describes any form of high capacity volatile memory attached to a CPU |

## *E*

| | |
|---|---|
| **EC:** | European Commission |
| **EDR:** | Enhanced Data Rate |
| **ESB:** | Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth |
| **EU:** | European Union |
| **Exascale:** | Computer systems or Applications, which are able to run with a performance above $10^{18}$ Floating point operations per second |

## *F*

| | |
|---|---|
| **FFT:** | Fast Fourier Transform |
| **FP7:** | European Commission 7th Framework Programme |
| **FPGA:** | Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing |

## *G*

| | |
|---|---|
| **GCE:** | Global Collective Engine, a computing device for collective operations |
| **GFLOP/S:** | Gigaflop, $10^9$ Floating point operations per second |
| **GFLOPS/W:** | Giga ($10^{\wedge 9}$) Floating-Point Operations per Second per Watt, or alternatively: Giga Floating-Point Operations per Joule |
| **GPFS:** | IBM General Parallel File System |
| **GPGPU:** | General Purpose Graphics Processing Unit |
| **GPU:** | Graphics Processing Unit |
| **GROMACS:** | A toolbox for molecular dynamics calculations providing a rich set of calculation types, preparation and analysis tools |

## *H*

| | |
|---|---|
| **H2020:** | Horizon 2020 |

| | |
|---|---|
| **HBM:** | High Bandwidth Memory |
| **HCA:** | Host Channel Adapter |
| **HPC:** | High Performance Computing |
| **HPDA:** | High Performance Data Analytics |
| **HPDBSCAN:** | Highly Parallel Density-Based Spatial Clustering for Applications with Noise |
| **HPL:** | High Performance Linpack |
| **HT:** | Hyper - Threading |
| **HW:** | Hardware |

## *I*

| | |
|---|---|
| **IMB** | Intel MPI Benchmarks |
| **Intel:** | Intel Deutschland GmbH, Neubiberg, Germany |
| **I/O:** | Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation |

## *J*

| | |
|---|---|
| **JUELICH:** | Forschungszentrum Jülich GmbH, Jülich, Germany |

## *K*

| | |
|---|---|
| **KU Leuven:** | Katholieke Universiteit Leuven, Belgium |

## *L*

| | |
|---|---|
| **LHC:** | Large Hadron Collider (LHC), the world's most powerful accelerator providing research facilities for High Energy Physics researchers across the globe |
| **LU** | Lower-Upper decomposition of a matrix into a product of lower and upper diagonal matrices; generalization of the familiar Gauss algorithm for solving linear equations. |

## *M*

| | |
|---|---|
| **MPI:** | Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages |
| **MSA:** | Modular Supercomputer Architecture |
| **MT:** | Multi-Threading |

## *N*

| | |
|---|---|
| **NAM:** | Network Attached Memory |
| **NCSA:** | National Centre for Supercomputing Applications, Bulgaria |

| | |
|---|---|
| **NEST:** | Widely-used, publically available simulation software for spiking neural network models developed by NMBU. |
| **NMBU:** | Norwegian University of Life Sciences, Norway |
| **NIC:** | Network Interface Controller |
| **NN:** | Neural Network |
| **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 |

## *O*

| | |
|---|---|
| **OmpSs:** | BSC's Superscalar (Ss) for OpenMP |
| **OMNIweb:** | NASA provided dataset collection of low resolution “OMNI” (LRO) data of near-Earth solar wind magnetic field and plasma parameter data |
| **OPA:** | Intel Omni Path |
| **OpenCL:** | Open Computing Language, framework for writing programs that execute across heterogeneous platforms |
| **OpenMP:** | Open Multi-Processing, Application programming interface that support multiplatform shared memory multiprocessing |

## *P*

| | |
|---|---|
| **ParTec:** | ParTec Cluster Competence Center GmbH, Munich, Germany. Linked third Party of JUELICH in DEEP-EST |
| **PCIe:** | Peripheral Component Interconnect Express; a bus that is often used to connect CPUs to GPUs, network devices, etc. |
| **PDUs:** | Power Distribution Units |
| **piSVM:** | Parallel classification algorithm |
| **PMT:** | Project Management Team of the DEEP-EST Project |

## *R*

| | |
|---|---|
| **RAID:** | Redundant Array of Inexpensive Disks |
| **RAM:** | Random-Access Memory |
| **RDMA** | Remote Direct Memory Access |
| **RMA** | Remote Memory Access |

## *S*

| | |
|---|---|
| **SIMD:** | Single Instruction Multiple Data |
| **SKU:** | Stock Keeping Unit |
| **SSD:** | Solid-State Drives |

| | |
|---|---|
| **SSSM:** | Scalable Storage Service Module |
| **SVM:** | Support Vector Machine |
| **SW:** | Software |

## *T*

| | |
|---|---|
| **TCP:** | Transmission Control Protocol; a reliable, stream-based network protocol |
| **TDP:** | Thermal Design Power |
| **TFLOP/S:** | Teraflop, $10^{12}$ Floating point operations per second |

## *U*

| | |
|---|---|
| **UoI:** | Háskóli Íslands – University of Iceland, Iceland |

## *V*

| | |
|---|---|
| **VGG:** | Name of a research group that developed a successful topology of a convolutional neural network for large-scale image recognition tasks used often as pre-trained networks today |

## *W*

| | |
|---|---|
| **WP:** | Work package |

## *X*

| | |
|---|---|
| **xPic** | Programming code developed by partner KU Leuven to simulate space weather |