International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 14, Number 1, pages 2–25, January 2024

Automated Network Performance Characterization for HPC Systems

Niklas Bartelheimer
Goethe University Frankfurt
Frankfurt am Main, Hesse, 60325, Germany

Zhaobin Zhu
Goethe University Frankfurt
Frankfurt am Main, Hesse, 60325, Germany

Sarah Neuwirth
Johannes Gutenberg University Mainz
Mainz, Rhineland-Palatinate, 55099, Germany

Received: July 29, 2023
Revised: October 19, 2023
Accepted: November 24, 2023
Communicated by Susumu Matsumae

Abstract

In this work, we propose the Network Performance Collector (NPC) workflow for automated network performance characterization. The workflow is based on the collection, processing as well as visualization of network performance metrics such as throughput and latency and can be used for analysis with different network performance models. Depending on the chosen model, benchmark tools such as iperf or sockperf as well as microbenchmarks specific to parallel programming models can be automated and orchestrated for data collection with the NPC. The data obtained can then be used by NPC, for example, to validate and characterize the performance of the underlying network or to analyze the system boundaries for a particular application. We provide a prototype implementation of the proposed workflow and demonstrate its effectiveness by automating an extended Roofline model analysis.

Keywords: Network Interconnects, Automated Performance Characterization, Extended Roofline Model, HPC, Network Benchmarks, JUBE Benchmarking Framework

1 Introduction

There are several ways to determine the actual performance characteristics of a point-to-point link for a given interconnection network. Well-known tools such as *qperf* [34] or *sockperf* [39] offer various network benchmarks to determine concrete values for network metrics such as the throughput or latency of a connection. As with most benchmarks, the reproducibility of the obtained results and the observed performance depends on the configured parameter space and the execution environment. Network benchmark tools offer a wide range of options depending on the network technology and protocol, but are commonly based on the client-server model, i.e. it is necessary to use different parameters for server and client side. This circumstance makes it difficult to perform such

benchmarks on supercomputers where workload managers such as Slurm are used. In addition, microbenchmarks for parallel programming models such as MPI can provide an estimate for the network performance observed by applications. Although it is possible to obtain an interactive shell, it would be more convenient and user-oriented to run these benchmarks as simple, automated jobs that can be scheduled by the workload manager.

To address these shortcomings, we propose the *Network Performance Collector* (NPC) workflow to ensure reproducibility, comparability, and portability of the obtained results. With NPC, it is possible to automate the benchmarking process and easily obtain performance data for different subsets of the available benchmark configurations. The modularity of its design allows NPC to be configured, applied, and extended to different measurement tools, system configurations, and network performance models, such as the Roofline model [49]. The goal is to develop a tool chain for automating the characterization of modern HPC systems, with particular regard to the increasingly heterogeneous (network) infrastructure. One example is the *Modular Supercomputer Architecture* (MSA) [27, 40], which leverages a system design that is based on heterogeneous hardware modules instead of the traditional homogeneous, fat-node approach. MSA systems consist of multiple modules, including a general-purpose CPU cluster, an accelerator-based Booster cluster (e.g. GPUs), or a quantum computing module as independent compute resources. The modularity of these components also affects the deployed network technologies, i.e. each module could use a different technology such as InfiniBand or Ethernet. In this context, a workflow for network performance modeling and automated system characterization will be even more important.

This paper extends our previous work [3] and makes the following contributions:

- We introduce the Network Performance Collector (NPC) workflow, which provides the means to configure and run benchmark and application sets in an automated manner. Its design targets the reproducibility, comparability, and portability of obtained performance results.
- We provide a proof-of-concept implementation of the proposed workflow that can be applied to applications and (micro)benchmarks. For demonstration purposes, we integrate an extended version of the Roofline model.
- We demonstrate the effectiveness and applicability of our proposed workflow by evaluating different network and micro-benchmarks as well as two selected applications on three different compute clusters: FUCHS-CSC and Cesari-MSQC at Goethe University Frankfurt, and the DEEP prototype system at the Jülich Supercomputing Centre.

The remainder of this work is organized as follows. Section 2 provides an introduction to network benchmarks and performance models and discusses related work. Section 3 outlines the proposed workflow for automated network performance characterization. In Section 4, the prototype implementation of the workflow is discussed. Preliminary performance results and a proof-of-concept are presented in Section 5 by applying the workflow to automate an extended Roofline characterization. Section 6 provides a discussion about future work. The paper concludes in Section 7.

2 Background and Motivation

In the following, we provide an overview of tools for measuring network performance and their features. We also discuss relevant work in the area of performance modeling and prediction. Special emphasis is placed on the Roofline model, since we use this model for our prototype implementation.

2.1 Network Performance Measurement

Network performance is characterized by several key factors that collectively determine how well a network functions. This includes numerous characteristics and parameters, such as bandwidth, latency, jitter, throughput, reliability and scalability, that must be analyzed together to evaluate a given network. Therefore, we define *network performance measurement* as the set of processes and tools that can be used to quantitatively and qualitatively benchmark network performance and

provide exploitable data to address network performance issues. In this work, we focus on the automation of network performance characterization based on a selection of network performance benchmarks, which generate data that can be tailored to baseline performance using pre-set routines.

qperf [34], for example, can be used to measure the throughput and latency between two nodes for the *Transmission Control Protocol / Internet Protocol* (TCP/IP) and *Remote Direct Memory Access* (RDMA). A convenient aspect of qperf is that the number of different benchmarks that are run sequentially can be defined without having to restart the client or server. Moreover, it provides a specific command line parameter that can be used to terminate the server process from the client side. *sockperf* [39] can be used to measure throughput and latency via the Socket API, which means that it is not possible to measure raw RDMA performance. RDMA communication can only be evaluated by using *Socket Direct Protocol* (SDP) or *Internet Protocol over Infiniband* (IPoIB). *hpcbench* [22] is an older project specifically designed to provide benchmarks for what were then considered high-performance networking technologies, including Myrinet, Gigabit Ethernet, and QsNet. It is noteworthy that hpcbench combined benchmarks for *User Datagram Protocol* (UDP), TCP and the *Message Passing Interface* (MPI) in one project. *perftest* [33] consists of several independent benchmarks specifically designed to evaluate RDMA-based communication using Verbs. Alternative tools that are oftentimes available on Linux machines are *iperf3*, *netperf*, and *nuttcp*, which can also be used to evaluate TCP/IP-based networks.

Besides low-level network benchmarks, a wide variety of programming model specific microbenchmarks and application benchmarks can be used to analyze the impact of software and middleware on network performance. An example is the OMB (OSU Micro-Benchmarks) [30], which provides means to test various communication routines for parallel programming models such as the Message Passing Interface (MPI) [35] and OpenSHMEM [12].

2.2 Performance Models

A recent survey [37] has summarized an overview of state-of-the-art analytical communication performance models, including LogP, LogfP and $\log_n P$. The LogP machine [13], for example, is a model for parallel computation. The name is not related to the mathematical logarithmic function. Instead, the machine is described by the four parameters: L, o, g and P. The LogP machine consists of as many processing units with distributed memory as desired. The processing units are connected via an abstract communication medium that enables point-to-point communication. This model is pairwise synchronous and overall asynchronous, and the units are measured in multiples of processor cycles. LogP is described by the four parameters:

- L, an upper bound on the latency of the communication medium.
- o, the overhead of sending and receiving a message.
- g, the gap, defined as the minimum time interval between two send or receive operations. The reciprocal value of g corresponds to the available communication bandwidth per processor.
- P, the number of processing units.

LogfP [21] is a model for small message performance over the InfiniBand network. This model provides knowledge about the inherent parallelism of a specific InfiniBand hardware and encourages developers to use this parallelism efficiently. Cameron et al. proposed $\log_n P$ [10], a general software-parameterized model of point-to-point communication for use in performance prediction and evaluation. It can be used to analyze the impact of middleware on distributed communication.

The *Bulk Synchronous Parallel* (BSP) model is a parallel computing framework introduced by Leslie Valiant [46]. Its intend is to simplify the design and analysis of parallel algorithms for distributed memory systems. In BSP, computation is structured into discrete supersteps, with processors performing local computation and synchronized communication at the end of each superstep. This model abstracts away hardware complexities and considers the network as a black box, focusing on computation, communication, and synchronization. BSP facilitates the development of parallel

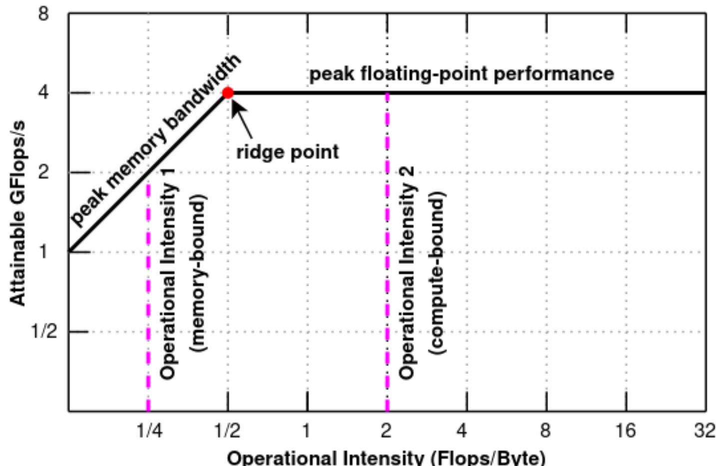


Figure 1: Example of a naïve Rooline Model [51].

algorithms by providing a clear and predictable way to balance workloads, distribute data, and analyze performance. Its simplicity makes it a valuable tool for designing and understanding parallel processing in various applications, from scientific simulations to big data processing.

$$T(h) = hg + I \tag{1}$$

To model the cost of a communication superstep, the BSP model introduces the h-relation in which every processor sends and receives at most h data words: $h = \max\{h_s, h_r\}$. Here, h_s is the maximum number of data words sent by the processor, while h_r denotes the maximum number of data words received by the processor. The cost of an h-relation can be described as shown in Equation 1. g gives the time per data word and I the global synchronization time. Basically, g describes the gap between sending successive data words, while I provides the latency.

In contrast, the *Roofline model* [49] is intuitive and can be used to estimate the performance of a given processor along with off-chip memory traffic. Essentially, it is a two dimensional graph showing the *operational intensity* (see Equation 2), also referred to as the *arithmetical intensity*, on the x-axis and the attainable floating point performance (see Equation 3) on the y-axis.

$$\textit{Operational Intensity} = \frac{\text{Floating point operations}}{\text{Memory bytes transferred}} \tag{2}$$

$$\begin{aligned} \textit{Attainable GFlops/sec} = Min(&\text{Peak Floating Point Performance}, \\ &\text{Peak Memory Bandwidth} \times \text{Operational Intensity}) \end{aligned} \tag{3}$$

As shown in Figure 1, the Roofline curve illustrates the theoretical peak performance of the system in terms of memory bandwidth and computational power. The point where the diagonal and horizontal ceilings intersect is known as the *ridge point* (RP). This point represents the optimal operational intensity for a given system, and can be used to identify potential performance bottlenecks. The main objective of the Roofline model is to set upper limits on the attainable floating point performance and DRAM bandwidth to characterize the performance of a given machine. Using these upper bounds, the performance of various *symmetric multiprocessing* (SMP) applications can be evaluated and analyzed, i.e. applications can be categorized as *memory-bound* or *compute-bound* depending on their position within the model. Optimal performance is achieved when the application can neither be classified as compute-bound nor memory-bound.

The traditional Roofline model can also be extended, for example to incorporate computing cores running on a GPU. The PCIe bandwidth can also be a useful additional upper bound, as it

defines the upper limit for the speed of data transfers between host and device memory. In addition, for distributed programs, the attainable bandwidth of interconnection networks can be used as another upper bound [15]. Recently, Wang et al. [47] have introduced the Roofline Power (RLP) management based on a latency-aware Roofline model. The RLP employs rigorously selected but generally available hardware performance events to construct rooflines. In particular, RLP extends the naïve Roofline model to include the memory access latency metric. The diverse applicability of the Roofline model proves that it provides a good starting point for a wide variety of performance studies. We intend to automate an extension of the traditional Roofline model for network performance characterization as part of this work.

2.3 Related Work

Previous work has mainly focused on three types of methods to characterize and predict performance: analytical modeling, replay-based modeling, and statistical modeling. Another viable option can be simulation. These approaches can be used to model the behavior of HPC applications and workloads.

An *analytical modeling* method has arithmetic formulas to describe performance and can quickly provide a prediction on metrics such as execution time or bandwidth. However, this method needs extensive efforts of human experts with in-depth understanding of network technologies. Today, communication performance modeling faces the challenge of the increasing complexity of the computational nodes of current networks. Rico-Gallego et al. [37] have put together an in-depth survey of existing communication performance models, which analytically represents the cost of network communication based on a limited number of platform-specific parameters. Many models have appeared over time, including LogP [13], LogfP [21], and $\log_n P$ [10].

A *replay-based model* is derived from historical run traces that contain detailed information about the computation and communication of an HPC program or workload. By analyzing traces, a synthetic program can be auto-generated to reproduce the behavior of the original program and predict its performance. However, replay-based modeling typically requires a large amount of storage to preserve traces (from hundreds of megabytes to dozens of gigabytes for each run). In addition, a synthetic application can only represent one particular execution path of the original application, which also limits the applicability of playback-based modeling. Especially for network performance, only limited options are available. The simulator Dimemas [19,41] can be used to analyze the performance of message-passing applications. Its traces can be further analyzed with the visualization tool Paraver [26]. Another replay-based tool is called ScalaTrace [29]. It relies on intra- and inter-node compression techniques of MPI events to extract an application's communication structure. These traces can then be used to replay and further analyze an application's MPI communication behavior. Recently, CODES [8, 25] has become a popular choice to simulate large-scale storage systems, I/O workloads, HPC network fabrics, distributed science systems, and data-intensive computation environments. the CODES simulation framework is built atop the Rensselaer Optimistic Simulation System (ROSS) [11], a discrete event simulation framework. Tools such as TRACER [1] can be used for predicting network performance and understanding network behavior by simulating messaging on interconnection networks with CODES.

A *statistical model* exploits machine learning techniques to determine the mapping function between performance metrics and specific characteristics. With enough training data, statistical models are able to make relatively accurate predictions about performance without the need for expertise and human input. It is natural and convenient to use the input parameters of an HPC program as characteristics. However, important performance factors may not be explicitly covered by input parameters. This is an important observation especially for the characterization of network performance. In recent years, studies have focused on how to apply machine learning methods to analyze and predict network performance. Wang et al. [48], for example, explain how to apply machine learning technology in the networking domain and also provide a brief survey of latest representative advances. Boutaba et al. [7] provide a comprehensive discussion about how different machine learning paradigms can be applied to fundamental problems in networking, including traffic prediction, routing and classification, congestion control, resource and fault management, Quality-of-Service management, and network security. Another survey [50] performed in 2019 fo-

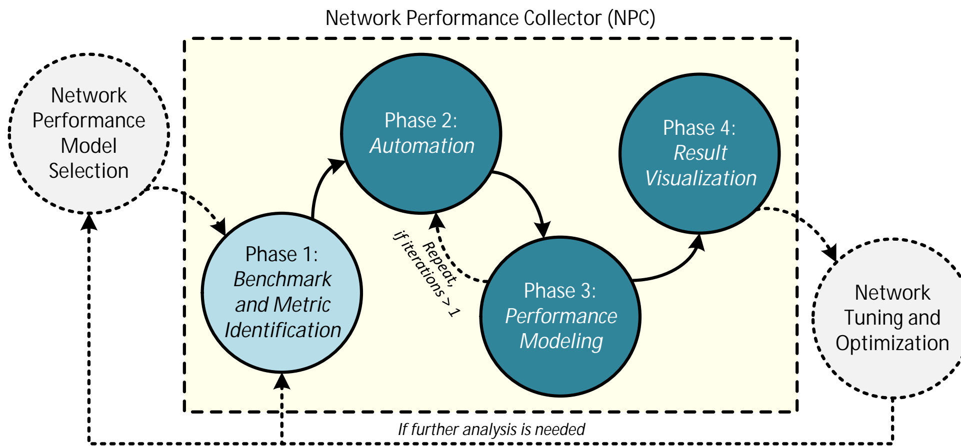


Figure 2: Overview of the NPC workflow.

cuses on machine learning techniques applied to software-defined networking (SDN). In 2020, Sun et al. [42] introduced automated performance modeling and performance prediction of parallel MPI applications based on applied machine learning with a focus on execution time prediction.

All of the discussed performance prediction and analysis methods are complementary to our work. With our proposed workflow, one of our goals is to provide an intuitive, low-overhead method for collecting network performance data during application execution or system maintenance. Its modular design allows the inclusion of additional performance models. Users, code developers, and system administrators can easily gain insight into the application and system performance of a given HPC platform. Hence, our main goal is to provide a performance engineering tooling infrastructure that can easily incorporate and automate various performance models and visualization methods.

3 Network Performance Collector (NPC) Workflow

Our paper proposes an automated and modular workflow design for network performance benchmarking, characterization, and metrics evaluation, as shown in Figure 2. The design philosophy of the *Network Performance Collector* (NPC) workflow follows three key principles:

1. *Reproducibility* is crucial for benchmarking because it ensures that performance measurements can be consistently validated by different parties. Reliable benchmarking helps assess hardware and software performance accurately, aiding in fair evaluations and informed decision-making.

2. *Comparability* is vital for benchmarking as it allows for fair and meaningful performance comparisons between different systems or configurations. It ensures that benchmark results accurately reflect the relative strengths and weaknesses of various technologies.

3. *Portability* is essential for benchmarks because it enables consistent testing across diverse computing environments. Benchmarks that can run on various platforms and architectures ensure that performance assessments can be conducted universally, enhancing the practicality and relevance of benchmarking results.

Automating benchmarks is important for reproducibility and hence comparability which is the major intent when performing benchmarks. Furthermore, managing different combinations of parameters is error-prone and often results in significant amounts of work especially if the parameter space gets large. To mitigate these issues, NPC helps perform and analyze network performance in a systematic way. It allows user-defined benchmark sets that can be adapted to new architectures and applications. In addition, the workflow provides means for pre- and post-processing to further analyze the results with different performance models.

The *Network Performance Collector* (NPC) workflow is divided into multiple phases. Prior to Phase 1 *Benchmark and Metric Identification*, a suitable *Network Performance Model* needs to be selected, depending on the performance aspect or system behavior (e.g. speed, bandwidth, latency) to be analyzed. It is important to note that this step defines at least a subset of the information needed for Phase 1. In the future, a network performance model selector module will be included as Phase 0, which will provide a predefined set of performance models and their test configurations. Depending on the chosen model, the metrics of interest need to be defined as well as the benchmark and parameter set. This decision must be made with respect to the given network technology. In addition, further microbenchmarks for parallel programming models such as MPI or Partitioned Global Address Space (PGAS) [2] can be included. These microbenchmarks reflect the observed communication performance at application level and can help with the identification of bottlenecks and overheads in the software stack.

The next step in the workflow is Phase 2 *Automation*, i.e. the orchestration and execution of the chosen benchmark set. Benchmarking a computer system usually involves numerous tasks, involving several runs of different applications. Configuring, compiling, and running a benchmark suite on several platforms with the accompanied tasks of result verification and analysis needs a lot of administrative and configuration work and produces a lot of data, which has to be analyzed and collected in a central database. Without a benchmarking environment all these steps have to be performed by hand. Phase 2 delivers such a benchmarking environment by providing a scripted framework that ensures interoperability and portability with little configuration overhead.

Subsequently, in Phase 3 *Performance Modeling*, the extraction of the corresponding benchmark results from the generated output is performed and prepared for further processing. This phase includes parsing the results of the benchmark runs and the characterization of the system network performance by applying an appropriate performance model. In Sections 5.3 and 5.4, we demonstrate how the extended Roofline model can be adopted to evaluate the network performance.

In Phase 4 *Result Visualization*, all generated benchmark results are converted into a suitable presentation format, e.g. a chart, a table or a csv file. Depending on the user-defined settings, Phases 2 and 3 can be run multiple times in an iterative manner, for example to analyze whether the results of different benchmarks are time-variant or -invariant. Including time as an additional dimension, Phase 4 then can provide insight into an application's performance over time, enabling the identification and understanding of performance anomalies.

Finally, the findings and insights of the NPC workflow can be used to apply *Network Tuning and Optimization*, for example to change the network settings or communication behavior of a certain application. While this phase is currently not implemented yet, automated tuning, bottleneck identification, and optimization will be provided as Phase 5 through the integration with MAWA-HPC, as discussed in Section 6.2. If further analysis is necessary or requested, the NPC workflow can be performed again. This can be the case, for example, after optimizations have been applied or if a new system is to be analyzed with the same benchmark set.

4 NPC Prototype Implementation

In the following, we describe the prototype implementation of our proposed workflow, which utilizes the JUBE environment and serves as a proof-of-concept. Furthermore, we present the Network Performance Benchmark (NPB), which demonstrates how the workflow can be used to automate low-level network microbenchmarks.

4.1 Workflow Automation

The NPC workflow implementation utilizes the *JUBE* benchmarking environment [36]. JUBE provides a script-based framework specifically to create and automate benchmark sets, and to provide reproducible and comparable results across runs on different compute platforms, i.e. JUBE ensures portability. JUBE-based workflows can either be written in YAML or XML. Our prototype implementation currently includes Phases 2 and 3, as shown in Figure 2. We exploit JUBE's ability to execute shell commands directly, which makes it easy to submit jobs with different parameters and

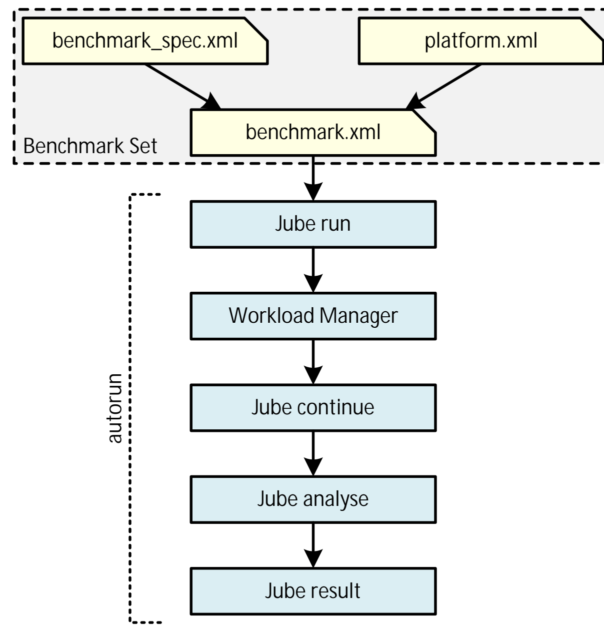


Figure 3: Overview of the JUBE execution flow.

configurations to the system's workload manager. All steps defined in JUBE run inside their own work directory. To collect results from different directories, the NPC implementation uses JUBE's *Analysis Module*, which provides a custom set of benchmark-specific regular expressions to extract the desired values from the result files. After the extraction, the collected results can be displayed and saved to a file inside the work directory.

One of the key motivations of the NPC workflow is to create a toolset which is able to orchestrate the deployment of benchmarks without the overhead of manual configuration. Therefore, a key design aspect is to maintain platform independence. A modular design is able to abstract from the entanglement between benchmark and platform to create the necessary level of abstraction. As discussed previously, portability is a key driver when creating a flexible and user friendly benchmarking environment. For this purpose, JUBE provides an *include* feature in order to incorporate heterogeneous system and benchmark specific architecture details. The light yellow part of Figure 3 schematically shows the structure of the workflow implementation for maintaining a platform independent design. A benchmark set for a particular benchmark is composed of three configuration files. For each embedded benchmark code, the `benchmark_spec.xml` file contains all the necessary information to run the code with different option sets. For example, each command line parameter that is accepted is given its own entry with a corresponding value or list of values for which a result must be determined. Furthermore, the specification file contains the required regular expressions to process and extract the desired information from the benchmark output. In addition to these benchmark-related specifications, the implementation requires the `platform.xml` file. This file consists mainly of the required variables and options used to interact with the workload manager. It may also contain cluster-specific default values for submitting and executing jobs. The final benchmark orchestration and configuration is done in the top-level file named `benchmark.xml`. Here each benchmark is orchestrated through one of the available options. It also contains the job submission configuration, e.g. for MPI applications `mpirun` must be defined as the startup program with all required startup parameters. Depending on the benchmark, several steps with internal dependencies are defined here, e.g. a benchmark run could consist of a `build`, `configuration` and `execution` step. A major advantage of maintaining this modular design philosophy is the applicability of the NPC workflow on a cross-platform basis. In practice, the switch between different platforms can be achieved by simply exchanging the `platform.xml` files containing information specific to the current architecture of the system in question.

After defining all of the benchmark-related information, the workflow starts by executing `jube run` (light blue boxes in Figure 3). The JUBE runtime executes all defined steps and interacts with the workload manager to submit the job scripts to the appropriate queues waiting for execution. JUBE does not track the progress of individual jobs. Therefore, `jube continue` must be used to signal to the runtime that job execution is complete and the results are now ready for processing. At the end of the workflow, `jube analyse` and `jube result` are used to extract individual values from the result files and put them into the defined result schema, e.g. `csv`, to convert them. Since this process can be lengthy, there is an option to use `jube-autorun` which goes through all stages without user interaction. An alternative would be to create a bash script that checks the status of the benchmark with `jube continue` and continues with the workflow once the job execution is complete. For example, the bash script can be run every 30 seconds with a `CronJob`, which should be available on any UNIX-like operating system. Examples are provided in Section 5.

4.2 Network Performance Benchmark (NPB)

Complementing our proposed NPC workflow, we also design the Python-based *Network Performance Benchmark* (NPB) for Phase 2, which addresses the shortcomings introduced by popular network benchmarks such as qperf or sockperf in terms of automation. Instead of real applications, NPB can be used as a synthetic workload in the NPC workflow. NPB accepts a number of command line arguments to properly orchestrate the selected benchmark. Processes are created with the help of *mpi4py*. The use of MPI facilitates the creation of separate processes for server and client on remote compute nodes. NPB checks at startup if the correct number of processes is requested, otherwise the program is terminated. Furthermore, it identifies the hostname of the server automatically, which makes it easy to use in conjunction with workload managers on HPC systems such as Slurm.

Listing 1: Example NPB call for sockperf.

```
mpirun -np 2 ./npb
    --servercmd="sockperf server"
    --clientcmd="sockperf throughput -i HOSTNAME"
    --killserver
```

Listing 1 shows an example NPB call for the sockperf throughput benchmark with default parameters. NPB is instrumented with both the command for the server and the command for the client part of the benchmark. One implementation detail is the keyword `HOSTNAME`, which is used as a wildcard for the IP address of the server. This keyword is essential when NPB is used in conjunction with a resource or workload manager. In this case, the nodes on which NPB is executed are usually not known in advance. By doing this, we establish a setup phase where the host names of the machines are exchanged and used to replace the keyword `HOSTNAME` in the client command. Since sockperf has no method to shut down the server side without explicitly terminating it, NPB is equipped with a *killserver* parameter. This parameter generates a message that is forwarded to the MPI process that has created the server process, which in turn triggers a termination signal to the server process via the subprocess API.

Benchmarks such as qperf provide the ability to obtain measurements for a range of parameters, such as message sizes from 1 to 4096 bytes. In addition, NPB implements the *repeat* keyword, which can be used to specify a range of values including a step size for benchmarks. This function is typically not supported natively by network benchmarks, but is mandatory to realize meaningful measurement series. The syntax of the *repeat* feature is similar to the one used by qperf (e.g. `--repeat msg_size:16:32768:*2` would result in the execution of the benchmark for 12 times with values from 16 to 32768 doubled in each iteration). It should be noted that the number of hyphens before the keyword *repeat* actually matters, since it is replicated for the command to be used for the benchmark (i.e. `--repeat` results in `--msg_size` whereas `-repeat` produces `-msg_size`).

It may be necessary to manipulate parts of a hostname determined by the NPB. This might be the case if the nodes are equipped with additional network interface controllers (NICs) or if different subnets are to be used. An example of such a use case is the presence of an IPoIB configuration. To solve this, NPB accepts two additional arguments to find and replace certain parts of a hostname.

Any benchmark that cannot output results in the data formats csv or json must be used in conjunction with a custom parser, which are provided by NPB. The parser has to implement a function with a specific signature, i.e. it takes the input string, the header for the csv file and the output format as commands. The latter are optional if only a specific format is of interest.

5 Experimental Evaluation and Proof of Concept

In this section, we evaluate the applicability of our workflow implementation. First, we use NPB from Phase 2 to conduct a bandwidth analysis with different network benchmarks and MPI implementations. Then, we employ the NPC to apply the Roofline model for selected applications. Both examples can be seen as preliminary work in automated network performance characterization.

5.1 Test Environment

The primary system employed for our experiments is the *FUCHS-CSC Cluster* at Goethe University Frankfurt, while the *Cesari-MSQC* and the *DEEP Prototype System* are used complementarily for the roofline analysis in Section 5.3.

FUCHS-CSC: The general-purpose computing platform is based on Intel CPU architectures running Scientific Linux 7.9 and SLURM as a resource manager. The cluster comprises water-cooled racks and features a total of 198 identical nodes. Each node is equipped with two Intel Xeon E5-2670 v2 (Ivy Bridge) processors, 128 GB of RAM and FDR InfiniBand connectivity. Additionally, each node provides up to 1.4 TB of local scratch space. FUCHS-CSC is backed by a parallel scratch file system based on BeeGFS with an aggregated bandwidth of 27 GB/s and a capacity of 2.4 PB.

Cesari-MSQC: This small general-purpose cluster is located at Goethe University Frankfurt and managed by the *Modular Supercomputing and Quantum Computing* (MSQC) group. The system consists of 16 nodes each powered by two Intel Xeon E5-2660 v2 (Ivy Bridge) in a dual socket configuration, with 32 GB of RAM and FDR InfiniBand. The cluste leverages Rocky Linux 8.7 (Green Obsidian) as the Operating System and Slurm as the resource manager.

DEEP Prototype: The DEEP system is an MSA prototype comprising three distinct compute modules. The *Cluster Module* (CM) utilizes 50 nodes, each equipped with two Intel Xeon Gold 6146 (Skylake) processors, 192 GB of RAM and EDR InfiniBand connectivity. In addition, each CM node incorporates a 400 GB NVMe SSD. The second module is the *Extreme Scale Booster* (ESB), consisting of 75 nodes powered by single Intel Xeon Silver 4215 (Cascade Lake) processors and Nvidia V100 Tesla GPUs (32 GB HMB2). Each ESB node is equipped with 48 GB of RAM and EDR InfiniBand. The third module is the *Data Analytics Module* (DAM) providing 16 nodes with two Intel Xeon Platinum 8260M (Cascade Lake) processors, 384 GB of RAM and EDR InfiniBand connectivity. Eight DAM nodes are equipped with a single Nvidia V100 Tesla GPU, while four nodes feature two GPUs of the same type. The last four DAM nodes are configured with two Intel STRATIX10 FPGAs (32 GB DDR4) each All nodes in the DAM module are equipped with two 1.5 TB Intel Optane SSDs [23].

5.2 Applicability of the Network Performance Benchmark

To test the applicability of our NPB implementation, we have defined an experimental environment in which we want to study the performance of different benchmarks and compare them with each other. We chose qperf and sockperf as low-level network benchmarks because both are capable of providing results for the network technologies Gigabit Ethernet (GigE) and InfiniBand. In addition, we used the OSU Micro-Benchmarks (OMB) [30] to provide a comparison with the performance results of MPI-based point-to-point communication. Here, we also compare the performance of two

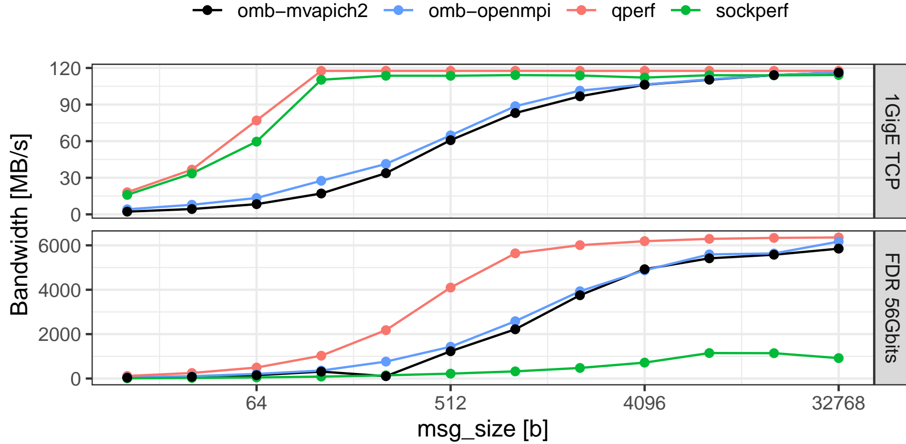


Figure 4: Comparison between the benchmarks qperf and sockperf and the MPI implementations OpenMPI and MVAPICH2.

Table 1: Benchmark configuration.

Benchmark	#Nodes	#Process	#OMPThreads	Precision
HimenoXL	2	8	-	FP32
HimenoXL	4	64	-	FP32
HPCG	2	4	40	FP64
HPCG	4	8	80	FP64

different MPI implementations with each other, namely *MVAPICH2* [31] and *OpenMPI* [17]. The benchmark runs were conducted for message sizes ranging from 16 to 32768 bytes.

Figure 4 depicts the results for GigE (upper plot) and FDR InfiniBand (lower plot). As can be seen, the performance achieved with qperf and sockperf is quite similar for GigE. The throughput delivered by the MPI benchmarks also reaches peak performance, but only for message sizes of 4096 bytes and larger. Both qperf and sockperf perform measurements for a fixed period of time, e.g. 1 second. During this time, the client sends as many messages as possible, so the network reaches its saturation point much earlier. The OMB does not run for a specific time, but sends a defined number of messages. This number is called the window size and has a default value of 64. The observation is the same for both network technologies and can be explained by the large difference in message rates. A closer look at the FDR InfiniBand results reveals the poor performance of sockperf. As mentioned in Section 2.1, sockperf can only be used with the Socket API. Thus, the results shown were obtained with IPoIB, which is likely to perform the worst in this case.

5.3 Roofline Model Analysis

5.3.1 Machine Characterization

We used qperf to determine an upper ceiling for the attainable network bandwidth. The measurement was performed for a reliable connected queue pair. For the floating point ceiling of the used processor, the *Likwid Performance Tools* [45] were applied. The peak *Floating Point Operations per Second* (FLOPS) were measured with `likwid-bench -t peakflops_avx -W N:400kB:40` for FUCHS-CSC and Cesari-MSQC benchmark configuration for single and double precision, respectively. For the CM of the DEEP system, `likwid-bench -t peakflops_avx512_fma -W N:384kB:48` was used.

5.3.2 Application Benchmarks and Experimental Setup

We used the well-known *Himeno* [20] and *High Performance Conjugate Gradients* (HPCG) [14] benchmarks to test the proposed workflow and evaluate the application of the Roofline model. Both

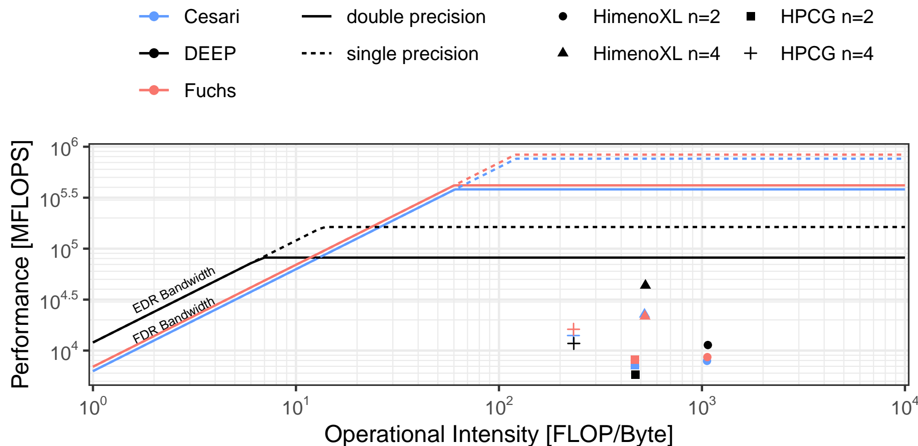


Figure 5: Roofline model with Himeno and HPCG results.

benchmarks use MPI to leverage the single program multiple data (SPMD) paradigm and report the FLOPS achieved as benchmark results. Since we are interested in the network performance, we have to measure the number of received and transmitted bytes. For this task, both benchmarks were modified to support the *Performance Application Programming Interface* (PAPI) [43]. PAPI provides access to hardware performance counters for the CPU as well as for the NIC. Using PAPI's low-level API, it is possible to track the amount of bytes transferred per node. The roofline can then be defined as shown in Equation 4:

$$P = min(P_{peak}, I \cdot b_s), \tag{4}$$

where P_{peak} is the attainable floating point performance in *FLOPS*. The slope is defined in Eq. 5:

$$I \cdot b_s = \frac{FLOP}{network\ byte} \cdot \frac{byte}{s}, \tag{5}$$

where I corresponds to the operational intensity and b_s is the attainable network bandwidth. Figure 5 shows the Roofline model for a single node of the system used, i.e., the results were downsampled to reflect the performance results at the node level regardless of the number of processes. The benchmark configuration can be seen in Table 1. For the Himeno benchmark, we used the "static allocation, MPI" implementation without employing additional threads and did not provide any hints for process binding. HPCG has been configured to use MPI for inter-node communication and OpenMP for multi-threading at the node-level. Here, we used the `-map-by ppr:1:socket` option provided by OpenMPI's `mpirun` implementation to launch exactly one MPI process per available socket. For the OpenMP threads, we used the `OMP_PLACES=cores` environment variable to control the location where threads are launched. The FUCHS-CSC and Cesari-MSQC clusters each offer ten cores per socket per machine. Consequently, we chose to create 10 threads per socket and map them individually. In addition, although a node in the CM module of the DEEP system provides up to 12 cores per socket, for the sake of better comparison, we still chose to spawn 10 threads per socket. Since we used a dual socket system, we made sure, that MPI processes were spawned in a *Non-Uniform Memory Access* (NUMA) aware fashion, employing the `-map-by socket` option provided by OpenMPI's `mpirun` implementation. HPCG has been configured to use MPI for inter-node communication and OpenMP for multi-threading at the node level. For the OpenMP threads, we used the `OMP_PLACES=cores` environment variable to control the location where threads are launched. In this case, we chose to spawn 10 threads per socket and bind them respectively to a single core.

The Himeno benchmark needs to be configured with a fixed problem size, i.e. running with an increasing number of processes shows strong scaling behavior. HPCG adjusts the total problem size depending on the number of processes started, while the local problem size remains the same for each process, leading to weak scaling results.

Table 2: Benchmark configuration.

Benchmark	#Nodes	#Processes	#OMPThreads	Precision
HimenoXL	2	20	-	FP32
HimenoXL	4	40	-	FP32
HimenoXL	8	80	-	FP32
HPCG	2	4	40	FP64
HPCG	4	8	80	FP64
HPCG	8	16	160	FP64

Figure 5 depicts the resulting Roofline model including the benchmark results. The results show that both problems are *computation bound* and are not affected by the available network performance. As expected, the operational intensity decreases by a factor of two when the number of nodes is doubled. The increase in processes and nodes also leads to more network communication, which pushes the point further to the upper boundary of network bandwidth when the number of nodes is four. Furthermore, it is evident that overall performance improves when utilizing more nodes and processes, irrespective of the system in use. In the case of the Himeno benchmark, an increased number of nodes leads to a proportional reduction in the problem size per node, directly impacting the data that can be accommodated in cache. Given the presence of *AVX 512* vector instructions and *Fused Multiply Add* (FMA) units, the Skylake architecture is better suited for executing the Himeno benchmark, as indicated by the performance results. On the other hand, for the HPCG benchmark, the variations in the vector architecture of the employed machines do not significantly influence the overall performance.

5.4 Analyzing Variance in Performance Data over Time

To exploit the advantages and automation capabilities of our proposed NPC workflow, we want to investigate whether the performance results of different benchmarks are time-variant or invariant. We use the same benchmark set as in Section 5.3, including network performance data for point-to-point links over the available InfiniBand network. Table 2 shows the used benchmark configurations. Compared to the previous measurements, we decided to run the benchmarks in three different configurations, including tests on up to eight nodes. In addition, the number of processes used for the Himeno benchmark was increased to make better use of the available computing capacity, since it does not offer multi-threading. The HPCG results were run with a configuration very similar to the one used for the previous experiments.

The duration of the experiment was 15 days with one measurement at 12:00 AM and another at 12:00 PM, resulting in a sample size of $n = 30$ for each observed performance metric. We focused on the floating point performance achieved, but also tracked the bytes sent and received through the hardware counter of the Mellanox NIC. The automated execution of the workflow is accomplished by setting up a CronJob that takes care of the on-time execution and thus the submission of the job to the Slurm Workload Manager. We use the theoretical limits of the used hardware, both for floating point performance and network bandwidth, to evaluate the obtained performance results.

5.4.1 Determination of Theoretical Performance Limits

Calculating the theoretical floating point performance of a given CPU requires knowledge of the microarchitecture specification. As can be seen in Equation 7, the attainable FLOPS of a given CPU are defined as the product of the number of cores, the frequency at which each core is operated, and the number of floating-point values processed in a single cycle. The latter is a crucial aspect of performance, since SIMD (Single Instruction Multiple Data) enables the CPU to execute a single instruction for as many floating-point values as will fit in the vector register, e.g. 8 values with double precision for a vector register width of 512 bits. In addition, modern CPUs can be equipped with specialized SIMD units that perform commonly used operations such as a multiplication immediately

Table 3: Fuchs-CSC CPU specification.

CPU	#Cores	Frequency	#SIMD	#SIMD Units	FMA
Intel Xeon E5-2670 v2	10	2.5 GHz	AVX (256 bit)	2	-

Table 4: Theoretical floating point performance limits for the used experimental setup.

#Nodes	#Sockets	Precision	GFLOPS
1	2	FP32	800
1	2	FP64	400
2	4	FP32	1600
2	4	FP64	800
4	8	FP32	3200
4	8	FP64	1600
8	16	FP32	6400
8	16	FP64	3200

followed by an addition in a single cycle. This type of computing unit is called a Fused Multiply Add (FMA). Consequently, Equation 6 defines the number of executable floating point operations (FLOPs) per cycle as the product of the number of values processed by the SIMD and FMA units and the number of units as the scaling factor.

$$\frac{\text{FLOPs}}{\text{cycle}} = \frac{\text{SIMD Width}}{\text{FP64/32}} \cdot \frac{\text{FMA Width}}{\text{FP64/32}} \cdot \text{Scaling factor} \tag{6}$$

$$\text{FLOPS} = \text{cores} \cdot \frac{\text{cycles}}{\text{second}} \cdot \frac{\text{FLOPs}}{\text{cycle}} \tag{7}$$

Table 3 contains the necessary technical details to calculate the theoretical FLOPS for the CPU type used in the FUCHS-CSC system. For a single CPU and single precision, this results in FLOPS = $10 \cdot 2.5GHz \cdot 8 \cdot 2 = 400$ GFLOPS. The CPU used provides AVX vector instructions operating on 256-bit vector registers, i.e., it is possible to process eight single precision values (FP32) in one cycle; there are two SIMD units and zero FMA units. In Table 4 the calculated values for different numbers of sockets, nodes, single and double precision can be found.

5.4.2 Performance Results over Time

In this part, we analyze the results of the long-term benchmark experiment at the system level. Furthermore, the distribution of the allocated compute nodes is analyzed and discussed. Figure 6 displays the achieved floating point performance of the Himeno benchmark in GFLOPS. The x-axis refers to the measurements performed, with adjacent values, starting on the left, scheduled with a time difference of 12 hours. From a performance perspective, it can be observed that doubling the number of processes and thus the number of compute nodes used roughly leads to a doubling of the achieved GFLOPS. It can also be seen that the fluctuations in performance between the individual measurements are almost negligible for 20 and 40 processes. Larger fluctuations are observed for the series of measurements with 80 processes. A special case is the last point (Measurement ID = 30) for both series with 20 and 80 processes, where the performance drops drastically. Looking at the results for the HPCG benchmark in Figure 7, we can observe a similar pattern. As with the Himeno benchmark, the floating point performance achieved increases by more than a factor of two when the number of processes is doubled, i.e., when the number of compute nodes is doubled.

Similar to the previous observation, the performance suffers significantly for the last measurement of the test series for eight and sixteen processes. Since the hostnames of the compute nodes used were recorded for each measurement point, we can identify the nodes used by the Himeno and HPCG benchmarks at the point of low performance. The data suggests that the results were obtained on the same set of compute nodes. In fact, the four nodes used for the Himeno benchmark with 40

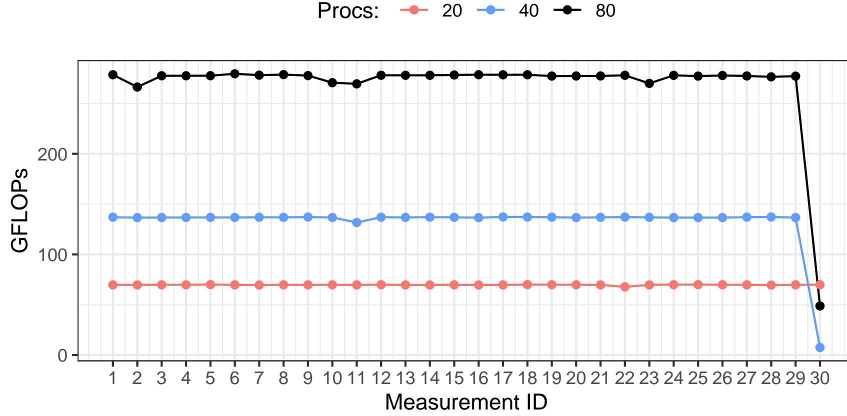


Figure 6: Floating point performance of the Himeno benchmark over a period of 15 days with two measurements per day.

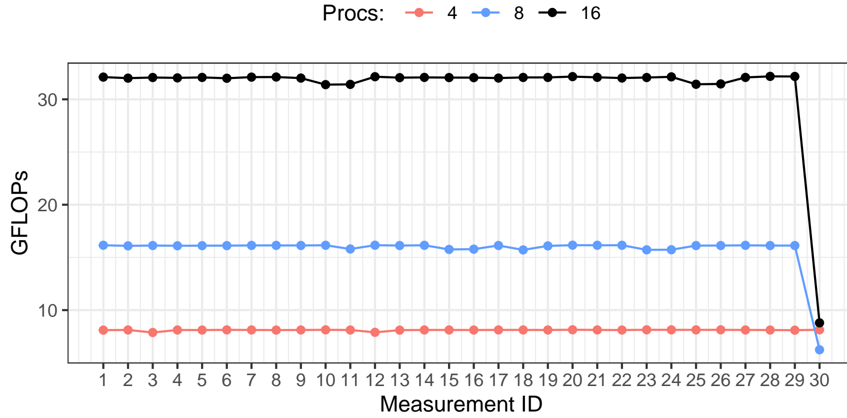


Figure 7: Floating point performance of the HPCG benchmark over a period of 15 days with two measurements per day.

processes and the HPCG benchmark with 8 processes were in the next largest set of nodes. This is a clear indication of a problem with one or all of the four nodes and could be further investigated by the cluster's maintenance staff.

A performance rating for all benchmark configurations can be found in the Table 5. The table shows the results as a percentage of the corresponding theoretical peak performance. For better comparability and accuracy, the mean value of the GFLOPS of all measurements was used to calculate the final percentage. It can be observed that the percentages for both the Himeno and HPCG benchmarks do not change significantly with the different number of compute nodes used. This is directly related to looking at the system-level results. In the previous figures, we observed that doubling the number of nodes also roughly doubles the performance. In fact, however, doubling the number of nodes also doubles the theoretical peak floating point performance.

In addition to the Himeno and HPCG benchmark, qperf was used to obtain bandwidth measurements for the available FDR InfiniBand link. The benchmark is configured to use a reliable connected queue pair and repeatedly send messages with a size of 4 MB over a point-to-point link. Figure 8 illustrates the pattern of bandwidth results over the 15-day period. Unlike the floating-point performance results of the Himneo and HPCG benchmarks, the achieved bandwidth is not as constant. The maximum value is 6330 MB/s, while the minimum value is below 6100 MB/s. The mean value is 6160 MB/s with a standard deviation of 43.3 MB/s, which is about 88% of the theoretical limit of 7000 MB/s. The results for network performance are highly dependent on the

Table 5: Achieved percentage of the theoretical floating point performance.

Benchmark	#Nodes	%
Himeno	2	4.356
Himeno	4	4.125
Himeno	8	4.203
HPCG	2	1.013
HPCG	4	0.981
HPCG	8	0.975

network topology used and also on the current amount of traffic, since network resources are shared across the cluster. Workload managers like Slurm allocate compute resources that are not far from each other, i.e. the distance in terms of hops (switches) should be small to allow nearest neighbor communication. For the qperf bandwidth benchmark, this means that we can assume that the number of two nodes required for the measurements are connected to the same switch. The presence of switching delays caused by a high level of network traffic can be an explanation for the observed bandwidth fluctuations.

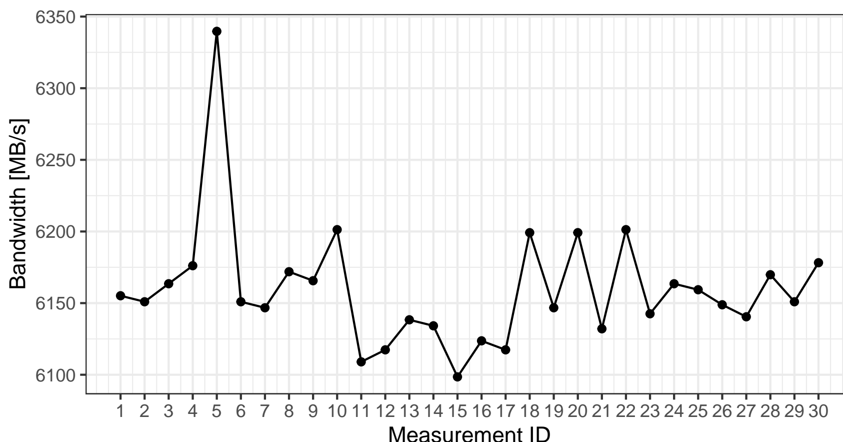
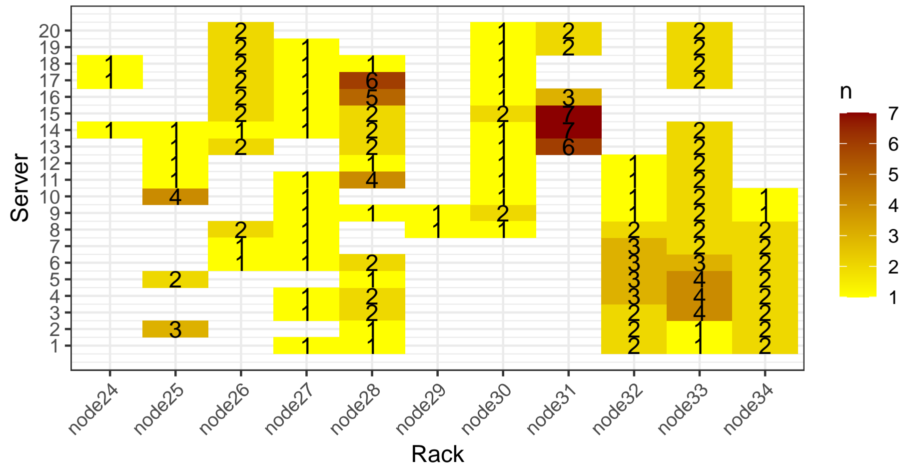


Figure 8: Bandwidth results for RDMA point-to-point connections (reliable connected) using qperf over a period of 15 days with two measurements per day.

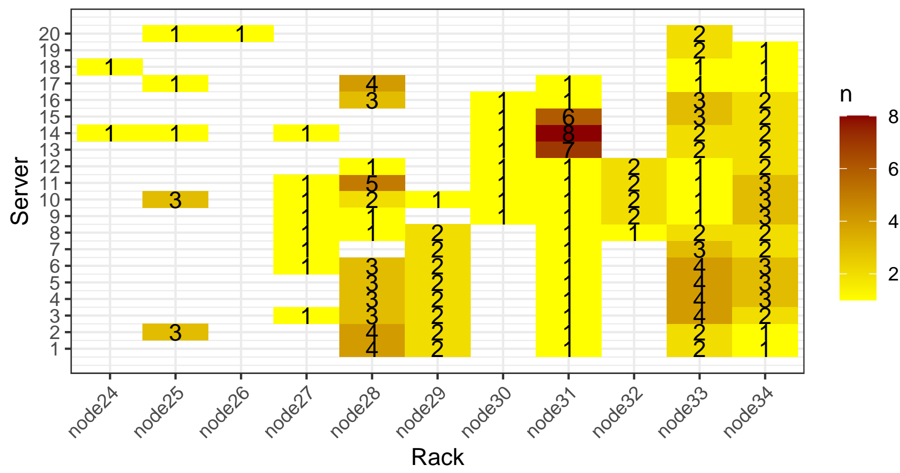
Since the FUCHS-CSC cluster at Goethe University is shared by many users, the measurements cannot be performed with the same computing resources at each run. Figure 9 consists of three heat maps, one for each benchmark, which provide information about which nodes of the cluster were used for the measurements and how often a particular node was used. Rack numbers are shown on the x-axis, while node numbers are shown on the y-axis. For example, the naming convention used is `node24-001`, referring to the 24th rack and the first server in the rack itself. In addition, the term `node24` defines the position of the top-of-the-rack switch within the fat tree network topology used. During the 15 days of the experiment, 178 out of 198 nodes were online and available. Of these 178 nodes, the Himeno benchmark was run on 112 different compute nodes. The HPCG benchmark ran on 107 different nodes. Qperf used 38 different nodes out of 178 available nodes.

6 Discussion and Future Work

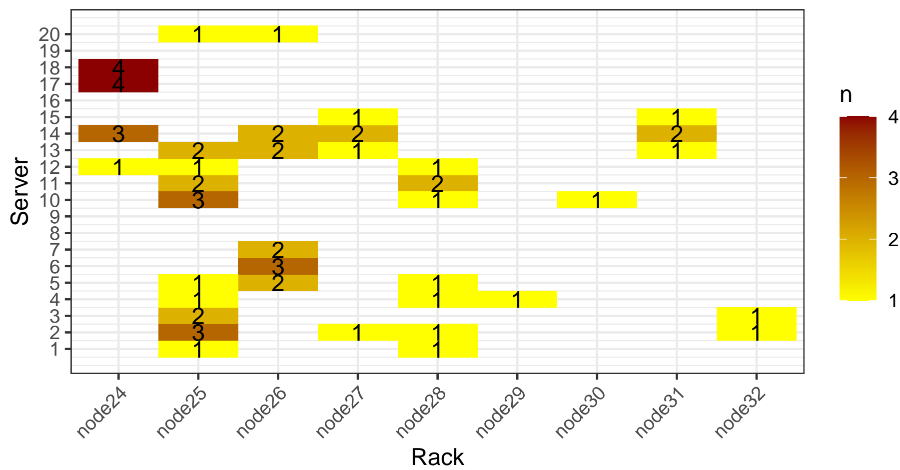
This work can be considered as a proof-of-concept analysis for our proposed NPC workflow. The current implementation limits the user to predefined options for performance analysis, including bandwidth performance, roofline analysis, and time variance, while its modular design allows for easy integration of new benchmarks, performance models, and visualization strategies. By using the



(a) Himeno



(b) HPCG



(c) Qperf

Figure 9: Heat maps of the allocated nodes for the individual benchmarks; **n** denotes the number of runs for which a specific node has been allocated.

JUBE benchmarking framework, our workflow ensures the reproducibility and comparability of the collected results. At the same time, JUBE offers the possibility to easily create benchmark sets, run them on different computer systems and evaluate the obtained results. In the following, we discuss the potential of the workflow and outline how we intend to extend and use it in the future.

6.1 Modeling Latency and Limitations of the Roofline Analysis

As discussed in Section 2.2, the Roofline model serves to identify performance limitations primarily associated with throughput, such as memory bandwidth and computational capacity. However, it is not suitable for detecting constraints arising from factors unrelated to throughput, like cache capacities and the latency introduced by the memory hierarchy. Integrating latency considerations is notably complex, as it necessitates a deep understanding of the underlying microarchitecture of the processor and the memory subsystem. In [9], a roofline extension is proposed, utilizing a *Directed Acyclic Graph* (DAG) based performance analysis that operates on a per-cycle basis.

While there are extensions for network throughput within the Roofline model [3], the progress towards extending it for network latency is relatively limited. This is primarily due to the various environmental factors that impact network latency, including network load at a given point in time.

6.2 Incorporating Additional Performance Aspects and Models

The prototype implementation of the NPC workflow currently focuses in particular on the traditional Roofline model extended for network performance. Our future work will focus on integrating additional performance models and network metrics into the workflow for the user to choose from in Phase 0. Sections 2.2 and 2.3 introduced multiple different methods to characterize and predict network performance. For our work, potential candidates are LogfP [21] and τ-Lop [38]. LogfP specifically targets the performance modeling for RDMA network fabrics, which is of particular interest in HPC systems and also considers small message transfers. τ-Lop [38], on the other hand, considers middleware costs, contention, synchronization, non-peer-to-peer collectives, and segments.

In addition, we also want to extend the potential of the Roofline model to include the examination of MPI operations. The well-known Roofline model is designed to provide a two-dimensional view of floating point performance, operational intensity, and memory bandwidth. For applications that use a distributed memory programming paradigm such as MPI, it may not be sufficient to consider only the local information of the nodes. An extension focusing on the MPI aspect of an application could use the ratio of $\frac{\text{MPI Ops}}{\text{second}}$ on the y-axis, i.e., MPI performance, and communication intensity as $\frac{\text{MPI Ops}}{\text{byte}}$ on the x-axis. In this view, the slope of the roofline would still be a function of bandwidth over operational intensity. This extension will also address network-related latency issues in MPI-based applications. The required application data can be collected using profiling tools such as Scalasca [18]. Typically, a profiler provides detailed information about the time spent in certain functions and also about the memory requirements. An impeding challenge is determining appropriate ceilings for the performance of MPI operations, since applications typically use different classes of MPI calls such as send-receive and collective operations, e.g., reductions. Collective communication is particularly difficult to analyze, since the amount of operations performed per second also depends on the number of processes involved in the operation. Therefore, it would be necessary to use flexible microbenchmarks for the essential operations. To model the mixed use of MPI calls in applications, the output of the profiler can be analyzed to calculate a weighted ceiling based on the distribution of MPI calls used, which is mapped to the results of the microbenchmarks.

6.3 Integration into the MAWA-HPC Framework

Given the complexity of modern HPC systems, achieving theoretical peak performance depends on a myriad of parameters and system configurations. In order to optimize the system performance and efficiently use the underlying resources, various methods can be applied, including simulation, benchmarking, and monitoring. However, these methods and the tools are not compatible with each other and only consider a selection of performance factors such network, I/O, resource allocation, or

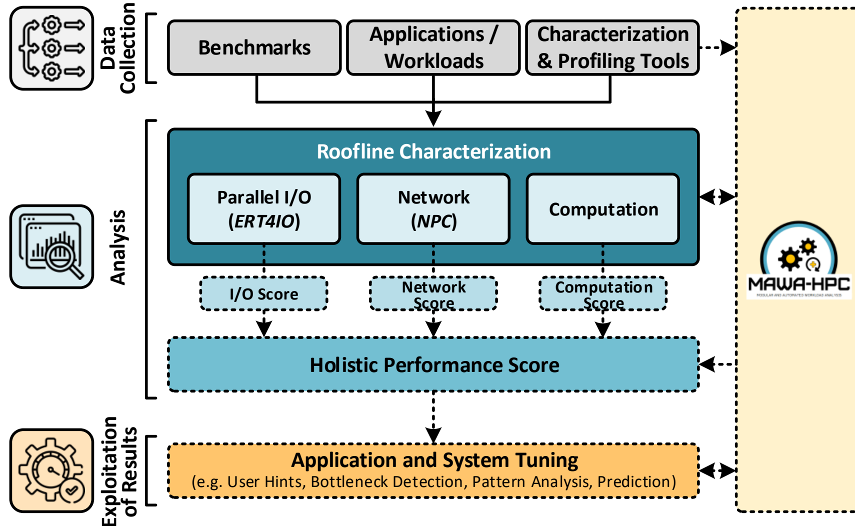


Figure 10: Integration into the MAWA-HPC framework [51].

parallel execution. Yet, each of these approaches generates knowledge that can be applied to similar problems or system configurations. To avoid that such knowledge is collected only for one-time purposes, and to also support other users, this knowledge must be easily accessible and available to the community. The MAWA-HPC (Modular and Automated Workload Analysis for HPC Systems) framework [52, 54] aims to develop a generic workflow and tool infrastructure that can be applied to different use cases and workloads from various science domains. Through its modular architecture, the workflow enables the support of various community tools, which increases the compatibility of the individual tools and covers new use cases.

By utilizing the support of monitoring and profiling tools, node-level performance engineering tools, network benchmarks, and microbenchmarks for different parallel programming models, we plan to integrate a multi-dimensional Roofline model and holistic performance scoring system into MAWA-HPC. The holistic performance score, as illustrated in Figure 10, takes into account multiple performance factors, including network, compute, and parallel I/O. Our previous work has introduced the I/O Roofline model [51, 53], which is tool and platform agnostic. The I/O Roofline model can easily be combined with NPC via the JUBE framework. As showcased in Section 5.4, including time as a third dimension, the Roofline model can provide insight into an application's performance over time, enabling the identification and understanding of performance anomalies. This will provide a comprehensive and interactive analysis for various use cases [6, 28]. For example, user hints such as Drishti [5] as well as bottleneck detection, pattern analysis, and performance prediction can be provided, leading to valuable insights for understanding the performance of emerging workloads.

6.4 Performance Analysis of the Modular Supercomputing Architecture

The *Modular Supercomputer Architecture* (MSA) [27, 40] breaks with traditional HPC system architectures by orchestrating heterogeneous computing resources at system-level, organizing them in compute modules with different hardware and performance characteristics. Modules with disruptive technologies, such as quantum devices, will also be included in a modular supercomputer to satisfy the needs of specific user communities. The goal is to provide cost-effective computing at extreme performance scales fitting the needs of a wide range of computational sciences. This approach brings substantial benefits for heterogeneous applications and workflows.

In a modular supercomputer, each application can dynamically decide which kinds and how many nodes to use, mapping its intrinsic requirements and concurrency patterns onto the hardware. Codes that perform multi-physics or multi-scale simulations can run across compute modules due to a global system-software and programming environment. Application workflows that execute different

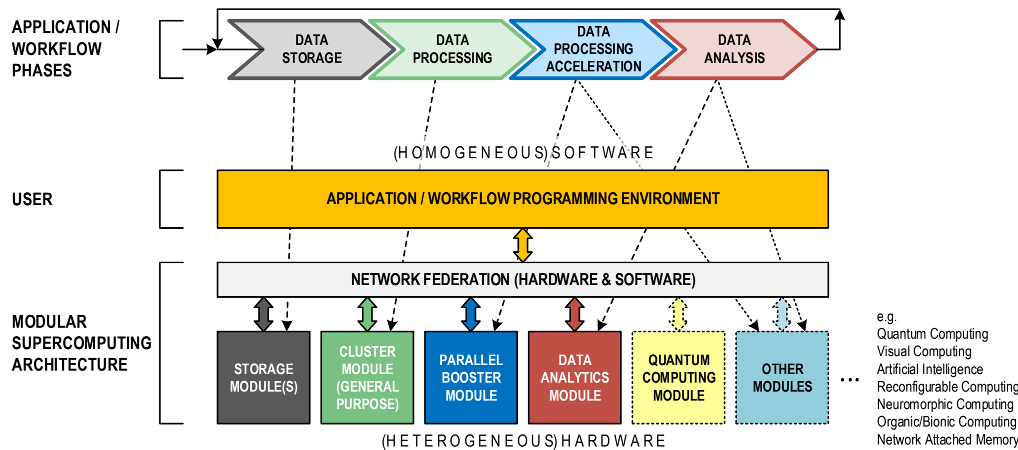


Figure 11: Modular Supercomputing Architecture [27].

actions after (or in parallel) to each other can also be distributed in order to run each workflow-component on the best suited hardware, and exchange data either directly (via message-passing communication) or via the filesystem. A modular supercomputer can provide any combination or ratio of resources between modules and is not bound to fixed allocations between, for example, CPUs and accelerators, as is the case with clusters with heterogeneous nodes. It is therefore ideal for running a heterogeneous mix of applications (higher throughput and energy efficiency).

The complex hardware environment is complemented by a software architecture [4, 44] designed to carefully address the requirements of co-designing and mapping applications for MSA systems. It focuses on the most relevant parts of the software stack required to run a supercomputer while also exploiting the modular supercomputer architecture. For example, the ParaStation MPI communication library [32] is designed to leverage distributed memory systems and the OmpSs-2 programming model [16] is used to exploit many-core processors, deep memory hierarchies, and accelerators. Furthermore, the software stack provides support for resiliency (i.e. checkpointing), parallel I/O, file system and storage. To support the complex I/O and storage system, the DEEP-EST project has focused on MSA-aware extensions [4] for BeeGFS and SIONlib.

The design of the network architecture for a modular supercomputer employs a modular approach throughout the whole system as well. In other words, the design of each module is driven by the requirements of the applications, which even influence the choice of the underlying network technology. In the DEEP-EST prototype [24], for example, the network federation comprises three different network technologies: 40 Gigabit Ethernet, InfiniBand, and Extoll. Given this complexity, modeling and understanding the performance of the different technologies is not a trivial task, as the interaction and impact of the various network fabrics must also be characterized. The NPC workflow holds great potential for automating the characterization and analysis of such complex system and network architectures. Instead of running individual benchmark sets on every single subsystem, NPC can be used to orchestrate the intra- and inter-module performance analysis for different run configurations, e.g. different combinations of compute modules. Since NPC is script-based, the workflow can exploit MSA-awareness for MPI and Slurm developed specifically for modular supercomputing systems. In future work, we plan to analyze the interplay of different interconnection networks when running code on compute modules with different network characteristics. For this purpose, we will first port existing network benchmarks for modular supercomputing systems.

7 Conclusion

In conclusion, the design of the NPC workflow provides an initial framework for the automation of performance analysis and characterization. The design follows three key principles: reproducibility, comparability, and portability. The main objective is to provide a tooling infrastructure that orches-

trates the use of benchmarks without the effort of manual configuration and enables the automatic evaluation and presentation of measurement results. Our prototype implementation highlights that the NPC can be implemented efficiently. In the experimental evaluation, we used the NPC workflow to perform roofline modeling for different hardware platforms and benchmarks. In addition, we also considered aspects such as time variance and system coverage. The modular design of NPC makes it easy to incorporate additional network performance characteristics, such as network efficiency or latency, and performance models. As discussed, the Roofline model is not applicable to analyze latency aspects. Therefore, we plan to investigate different techniques for modeling latency, for which we will also use the NPC to collect performance data. We also plan to extend our approach with existing work in the areas of simulation, machine learning, and analytical methods for performance estimation. In addition, our work is part of the MAWA-HPC project to evaluate and combine network performance, machine utilization and performance metrics, and I/O-related metrics into a unified performance model. Future work will also focus on performance characterization of modular supercomputing systems. Further information, including the prototype implementation of NPC and the NPB tool, is available on our GitHub companion page: https://github.com/msqc-goethe/npc.

Acknowledgment

The authors would like to thank Alexander Schifrin from Goethe University Frankfurt for his valuable comments and contributions. This research is supported by EUPEX, which has received funding from the European High-Performance Computing Joint Undertaking (JU) under GA No 101033975. The JU receives support from the European Union's Horizon 2020 research and innovation programme, France, Germany, Italy, Greece, United Kingdom, Czech Republic, and Croatia.

References

[1] Bilge Acun, Nikhil Jain, Abhinav Bhatele, Misbah Mubarak, Christopher D Carothers, and Laxmikant V Kale. Preliminary evaluation of a parallel trace replay tool for hpc network simulations. In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*, pages 417–429. Springer, 2015.

[2] Niklas Bartelheimer and Sarah Neuwirth. Toward Reproducible Benchmarking of PGAS and MPI Communication Schemes. In *29th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2023)*, 2023.

[3] Niklas Bartelheimer, Zhaobin Zhu, and Sarah Neuwirth. Toward a Modular Workflow for Network Performance Characterization. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 331–334, 2023.

[4] V. Beltran and P. Martinez. D6.3 Complete Programming Environment Implementation. Technical report, DEEP Extreme Scale Technologies, Grant Agreement Number: 754304, 2021.

[5] Jean Luca Bez, Hammad Ather, and Suren Byna. Drishti: Guiding End-Users in the I/O Optimization Journey. In *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*, pages 1–6. IEEE, 2022.

[6] Debasmita Biswas, Sarah Neuwirth, Arnab K. Paul, and Ali R. Butt. Bridging Network and Parallel I/O Research for Improving Data-Intensive Distributed Applications. In *2021 IEEE Workshop on Innovating the Network for Data-Intensive Science (INDIS)*, pages 50–56, 2021.

[7] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99, 2018.

[8] Kevin A. Brown, Nikhil Jain, Satoshi Matsuoka, Martin Schulz, and Abhinav Bhatele. Interference between I/O and MPI Traffic on Fat-Tree Networks. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Victoria Caparrós Cabezas and Markus Püschel. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 222–231, 2014.

[10] Kirk W. Cameron, Rong Ge, and Xian-he Sun. $\log_n$ P and $\log_3$ P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems. *IEEE Transactions on Computers*, 56(3):314–327, 2007.

[11] Christopher D Carothers, David Bauer, and Shawn Pearce. ROSS: A high-performance, low-memory, modular Time Warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.

[12] OpenSHMEM Community. OpenSHMEM Specification 1.5. http://openshmem.org/site/Specification. accessed 07-27-2023.

[13] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 1–12, 1993.

[14] Jack Dongarra, Michael A Heroux, and Piotr Luszczek. High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems. *The International Journal of High Performance Computing Applications*, 30(1):3–10, 2016.

[15] Amauda S. Dufek, Jack R. Deslippe, Paul T. Lin, Charlene J. Yang, Brandon G. Cook, and Jonathan Madsen. An Extended Roofline Performance Model with PCI-E and Network Ceilings. In *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, St. Louis, MO, USA, November 2021. IEEE.

[16] Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Process. Lett.*, 21:173–193, 2011.

[17] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[18] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurr. Comput.: Pract. Exper.*, 22(6):702–719, apr 2010.

[19] Sergi Girona, Jesús Labarta, and Rosa M. Badia. Validation of Dimemas Communication Model for MPI Collective Operations. In Jack Dongarra, Peter Kacsuk, and Norbert Podhorszki, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 39–46, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[20] Ryutaro Himeno. Himeno Benchmark. https://i.riken.jp/en/supercom/documents/himenobmt/. accessed 01-27-2023.

[21] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm. LogfP - a model for small messages in InfiniBand. In *Proceedings 20th IEEE International Parallel and Distributed Processing Symposium*, pages 6 pp.–, 2006.

[22] B. Huang, M. Bauer, and M. Katchabaw. Hpcbench - a linux-based network benchmark for high performance networks. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 65–71, 2005.

[23] FZ Juelich. DEEP Testcluster – System Overview. https://deeptrac.zam.kfa-juelich.de:8443/trac/wiki/Public/User_Guide/System_overview. accessed 10-18-2023.

[24] A. Kreuzer, E. Suarez, N. Eicker, and Th. Lippert, editors. *Porting applications to a Modular Supercomputer - Experiences from the DEEP-EST project*, volume 48 of *Schriften des Forschungszentrums Jülich IAS Series*. Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag, Jülich, 2021.

[25] Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Philip Carns. Enabling parallel simulation of large-scale HPC network systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):87–100, 2016.

[26] Adrian Munera, Sara Royuela, Germán Llort, Estanislao Mercadal, Franck Wartel, and Eduardo Quiñones. Experiences on the characterization of parallel applications in embedded systems with extrae/paraver. In *Proceedings of the 49th International Conference on Parallel Processing*, pages 1–11, 2020.

[27] Sarah Neuwirth. Modular Supercomputing and its Role in Europe's Exascale Computing Strategy. *PoS*, LATTICE2022:245, 2023.

[28] Sarah Neuwirth and Arnab K Paul. Parallel I/O Evaluation Techniques and Emerging HPC Workloads: A Perspective. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 671–679, 2021.

[29] Michael Noeth, Prasun Ratn, Frank Mueller, Martin Schulz, and Bronis R De Supinski. ScalaTrace: Scalable compression and replay of communication traces for high-performance computing. *Journal of Parallel and Distributed Computing*, 69(8):696–710, 2009.

[30] Dhabaleswar K. Panda. OSU Micro-Benchmarks. https://mvapich.cse.ohio-state.edu/benchmarks. accessed 01-27-2023.

[31] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. The MVAPICH project: Transforming research into high-performance MPI library for HPC community. *Journal of Computational Science*, 52:101208, 2021. Case Studies in Translational Computer Science.

[32] ParTec. ParaStation MPI. https://github.com/ParaStation/psmpi. Accessed: 2023-10-11.

[33] perftest. https://github.com/linux-rdma/perftest. accessed 01-27-2023.

[34] qperf. https://github.com/linux-rdma/qperf. accessed 01-27-2023.

[35] MPI Forum. MPI Documents. https://www.mpi-forum.org/docs/. accessed 07-27-2023.

[36] JUBE Benchmarking Environment. https://apps.fz-juelich.de/jsc/jube/jube2/docu/index.html#. accessed 03-14-2023.

[37] Juan A. Rico-Gallego, Juan C. Díaz-Martín, Ravi Reddy Manumachu, and Alexey L. Lastovetsky. A Survey of Communication Performance Models for High-Performance Computing. *ACM Comput. Surv.*, 51(6), Jan 2019.

[38] Juan-Antonio Rico-Gallego, Juan-Carlos Díaz-Martín, and Alexey L Lastovetsky. Extending τ-lop to model concurrent MPI communications in multicore clusters. *Future Generation Computer Systems*, 61:66–82, 2016.

[39] sockperf. https://github.com/Mellanox/sockperf. accessed 01-27-2023.

[40] Estela Suarez, Norbert Eicker, and Thomas Lippert. *Modular Supercomputing Architecture: from Idea to Production; 3rd*, volume 3, pages 223–251. CRC Press, 2019.

[41] Vladimir Subotic, Jose Carlos Sancho, Jesus Labarta, and Mateo Valero. A simulation framework to automatically analyze the communication-computation overlap in scientific applications. In *2010 IEEE International Conference on Cluster Computing*. IEEE, 2010.

[42] Jingwei Sun, Guangzhong Sun, Shiyan Zhan, Jiepeng Zhang, and Yong Chen. Automated Performance Modeling of HPC Applications Using Machine Learning. *IEEE Transactions on Computers*, 69(5):749–763, 2020.

[43] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting Performance Data with PAPI-C. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[44] K. Thust. D4.4 I/O software packages. Technical report, DEEP Extended Reach, Grant Agreement Number: 610476, 2017.

[45] J. Treibig, G. Hager, and G. Wellein. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.

[46] Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, aug 1990.

[47] Bo Wang, Anara Kozhokanova, Christian Terboven, and Matthias Mueller. RLP: Power Management Based on a Latency-Aware Roofline Model. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 446–456, 2023.

[48] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network*, 32(2):92–99, 2018.

[49] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM*, 52(4):65–76, April 2009.

[50] Junfeng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Communications Surveys & Tutorials*, 21(1), 2019.

[51] Zhaobin Zhu, Niklas Bartelheimer, and Sarah Neuwirth. An Empirical Roofline Model for Extreme-Scale I/O Workload Analysis. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 622–627, 2023.

[52] Zhaobin Zhu, Niklas Bartelheimer, and Sarah Neuwirth. MAWA-HPC: Modular and Automated Workload Analysis for HPC Systems. Research Poster, ISC High Performance Conference 2023 (ISC23), 2023.

[53] Zhaobin Zhu and Sarah Neuwirth. Characterization of Large-Scale HPC Workloads With Non-Naïve I/O Roofline Modeling and Scoring. In *29th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2023)*, 2023.

[54] Zhaobin Zhu, Sarah Neuwirth, and Thomas Lippert. A Comprehensive I/O Knowledge Cycle for Modular and Automated HPC Workload Analysis. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 581–588. IEEE, 2022.