Verifying Performance Guidelines for MPI Collectives at Scale

Sascha Hunold
sascha.hunold@tuwien.ac.at
Faculty of Informatics, TU Wien
Vienna, Austria

ABSTRACT

MPI collective communication operations are crucial for high-performance computing, making the efficient implementation of collective algorithms essential for optimal application performance. While most MPI libraries provide several algorithms for a specific collective operation, each may work better in a specific scenario. Therefore, selecting the most suitable algorithm for each use case is important. However, even the best algorithm in a given MPI library's set may deliver suboptimal performance.

Self-consistent MPI performance guidelines are general expectations that collectives must meet to be deemed performance-consistent. Specifically, a specialized collective call should not be slower than its less specialized counterparts. This paper introduces a tool for assessing the performance consistency of MPI collectives in a statistically sound manner. Through a case study, we demonstrate the current state of MPI performance consistency for three TOP500 machines.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**.

KEYWORDS

MPI, collective communication, performance analysis, benchmarking, performance guidelines

ACM Reference Format:
Sascha Hunold. 2023. Verifying Performance Guidelines for MPI Collectives at Scale. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3624062.3625532

1 INTRODUCTION

The Message Passing Interface (MPI) is fundamentally important for developing scalable, parallel applications for today's supercomputers [9]. It offers a wide array of routines to solve various communication issues in distributed systems, such as addressing individual processes and facilitating communication between them (point-to-point communication), or ensuring that transferred data items are interpretable by different processes (MPI data types). An important category of MPI are the collective communication operations, also known as MPI collectives. These operations perform data transfers among a group of processes, such as a broadcast (`MPI_Bcast`).

Collective communication operations in MPI are routines whose semantics are defined in the standard. However, their implementations are determined by the respective MPI libraries. Therefore, certain MPI collective operations can be realized using various algorithms, each of which may be advantageous in different communication scenarios [1].

From an MPI library's perspective, it is important to select the fastest algorithm for a given collective operation and set of parameters (e.g., the message size or the number of processes). The problem is that even if the library selects the best possible algorithm for a specific MPI collective from its set of algorithmic options, the selected algorithm may still be inefficient, or put differently, the collective may violate various performance guidelines. Such guidelines can be formulated from general performance expectations on MPI collectives, e.g., a broadcast of 1 kB of data should not be slower than a broadcast of 1 MB of data (surely assuming the same number of processes).

In this paper, we present a tool to automatically analyze the performance-consistency of common blocking MPI collectives. The performance tool orchestrates a number of micro-benchmark runs and evaluates the recorded measurements in a statistically sound manner. In the evaluation phase, the MPI collectives are not only tested against re-implementations using different collectives operations (mock-ups) but they can also be tested against collective algorithms that are implemented in other, external libraries. In a case study, we show that a large number of MPI collectives can be improved significantly, since they fail to be performance consistent. In this paper, we make the following contributions:

- We present the tool `MPINPC` (`MPINPC is Not a Performance Checker`), which can be used to automatically verify the performance consistency of MPI collectives.
- We showcase experimental outcomes on several larger HPC systems, showing that the set of collective algorithms available to MPI libraries should be significantly expanded. The results also suggest that MPI libraries need more performance tuning.

In the remainder, we recall the concept of performance guidelines for MPI operations in Section 2. Then, we present the specific features of the tool `MPINPC` and show experimental results for different supercomputers in Section 4. We conclude the paper in Section 5.

2 RELATED WORK AND BACKGROUND

The concept of self-consistent performance guidelines was introduced by Träff et al. [10]. Although performance guidelines can be formulated for a variety of MPI operations (e.g., data types, point-to-point), this paper focuses on guidelines for collective operations.

Performance guidelines represent the general expectations that library developers hold for MPI operations. Often, the semantics of

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3625532

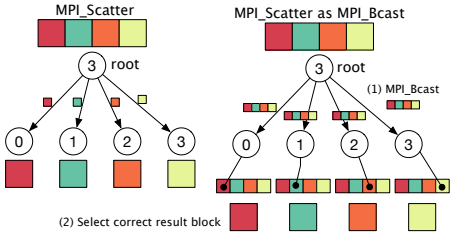


Figure 1: Illustration of guideline `MPI_Scatter`$(n) \preceq$ `MPI_Bcast`(n), where the right-hand side re-implements the functionality of `MPI_Scatter`.

certain MPI collectives can be emulated using other, less specific MPI collectives. Despite this, the expectation is that a less specialized re-implementation should not be faster than the original function, as this would make the specialized function unnecessary.

Let us consider the following guideline: `MPI_Scatter`$(n) \preceq$ `MPI_Bcast`(n). It states that scattering a communication volume of n blocks should not be slower than broadcasting the same vector of n blocks. The general idea of this guideline is depicted in Figure 1. On the left-hand side of this figure, we see the semantics of the specialized scatter function. On the right-hand side is the re-implementation of scatter using broadcast. Here, the entire data block is sent to all processes in the communicator, after which each process selects the specific block it would have received via scatter. Clearly, and also regardless of the additional memory overhead, the runtime of the scatter re-implementation using broadcast should not be faster than a library's implementation of scatter.

The verification of performance guidelines for MPI collectives was initially conducted by Hunold et al. [5], employing a multi-step approach to assess the performance-consistency of MPI libraries. In comparison, we introduce a one-shot tool for the automated and efficient analysis of performance guidelines. Additionally, we aim to assess performance guidelines at large scale, i.e., utilizing several thousands of processes. Hunold and Carpen-Amarie [4] also presented a tuning approach, wherein so-called mock-ups are used as library replacements if MPI libraries do not provide efficient algorithms for specific MPI functions. Parker et al. [6] presented a study showcasing performance-guideline violations of `MPI_Allgather` on the *Theta* supercomputer.

Due to space constraints, we can only point to a few related approaches, where the performance and scalability of MPI libraries and applications is analyzed. Shudler et al. [8] assess whether the scalability of MPI collectives measured in experiments matches their theoretical runtime complexity. A performance model is typically required to compare the performance expectation in reality. In order to obtain an empirical performance model, Copik et al. [2] have shown how taint analysis can help to improve the quality of performance models.

3 AUTOMATIC VERIFICATION OF MPI PERFORMANCE GUIDELINES

We now introduce our guideline verification tool called `MPINPC` (`MPINPC is Not a Performance Checker`). We begin by discussing the need for such a tool, despite the procedure proposed by Hunold et al. [5] for verifying performance guidelines. The primary limitation of the previous approach was its restricted applicability and feature set. Scientists were required to initiate multiple benchmark calls, gather data, identify the correct data processing method, and plot the results using their own functions. Additionally, the prior approach primarily concentrated on mock-ups when examining the performance guidelines.

To significantly enhance the previous works, our tool offers the following comprehensive set of features:

Nonparametric tests: Our tool is capable of comparing the runtime of MPI collectives in a statistically meaningful manner. Users have the option to export all raw data from the experiments. Additionally, `MPINPC` is equipped with several nonparametric tests, i.e., t-test, Wilcoxon rank-sum, or Mann-Whitney, which may be used in different situations.

Precision and Predictability: Using the ReproMPI benchmark [3] as a foundation, our tool leverages two unique features for performance-guideline verification. It includes multiple clock synchronization algorithms, eliminating measurement distortion from barrier synchronization for small message sizes [7]. It also consistently measures the runtime of collectives within a specific timeframe, ensuring predictability in performance-guideline verification. For example, each collective is measured for a maximum of 2 s, a crucial feature for seamless integration into CI/CD pipelines for performance-regression testing.

Extensibility: The tool is bundled with a large number of re-implementations of specialized MPI collectives. For example, it provides a *mock-up* for Guideline GL1.

$$\texttt{MPI\_Bcast}(n) \preceq \texttt{MPI\_Allgatherv}(n)\,, \tag{GL1}$$

$$\texttt{MPI\_Bcast}(n) \preceq \texttt{MPI\_Bcast}_{\text{lane}}(n) \tag{GL2}$$

$$\texttt{MPI\_Bcast}(n) \preceq \texttt{MPI\_Bcast}_{\text{hier}}(n) \tag{GL3}$$

In this mock-up implementation, the same functionality as in broadcast is realized with allgatherv, where only one process (the broadcast root) inserts data into the allgatherv call. More importantly, `MPINPC` is also easily extensible to include collective algorithms from external libraries. Two examples are the Guidelines GL2 and GL3, where the algorithm selected by the MPI library is tested against the algorithms devised by Träff and Hunold [11].

Before we present our experimental study, we show a typical use case of `MPINPC` in Figure 2. In this example, the performance of `MPI_Allreduce` of Open MPI is compared to the performance of the mock-up implementations and external algorithms. We ran the guideline verification with 32×32 processes on our local cluster for the following message sizes: 8, 80, 800, 8000, and 80 000 B. We show one of the possible outputs of `MPINPC`, where the runtime of the default collective is compared to possibly faster competitors. In the presented scenario, the default implementation violates the performance guideline for 8 B and 8000 B. In both cases, algorithms from external libraries are statistically found to be faster (here we used the rank-sum test). The output also includes how much faster another algorithm has been, which is listed in the *slowdown* column. We can observe that the guideline violation is more severe for 8000 B, where the default algorithm is 2.5x slower than the best algorithm from `MPINPC`.

```
MPI Collective: MPI_Allreduce
collective      count   N    ppn   n     default_median  slowdown   fastest_mockup              mockup_median
MPI_Allreduce   8       32   32    5000  0.023603        1.112355   allreduce_as_allreduce_hier 0.021219
MPI_Allreduce   80      32   32    5000  0.027895
MPI_Allreduce   800     32   32    5000  0.032663
MPI_Allreduce   8000    32   32    5000  0.135422        2.547088   allreduce_as_allreduce_lane 0.053167
MPI_Allreduce   80000   32   32    5000  0.222206
```

Figure 2: Example output of `MPINPC` for `MPI_Allreduce` and 32×32 processes.

Table 1: Overview of hardware setup and MPI libraries used in the experimental evaluation.

machine	country	compute nodes	interconnect	MPI libraries
Joliot-Curie Rome	France	1656 compute nodes (Atos) 2 × 24-core Intel Skylake@2.7GHz	Infiniband EDR (fat tree)	Open MPI 4.1.4
Theta	USA	4392 compute nodes (Cray XC40) 64-core, Intel Xeon Phi 7230@1.3GHz	Aries interconnect	Cray MPICH 7.7.14
Discoverer	Bulgaria	1128 nodes (Atos BullSequana XH2000) 2 × 64-core AMD Epyc 7H12	Infiniband HDR (Dragonfly+)	Open MPI 4.1.4

4 EXPERIMENTAL RESULTS

Now, we show the experimental results obtained with `MPINPC` on three TOP500 supercomputers.

Experimental Setup. The hardware configuration of the three parallel machines is summarized in Table 1. Although the machines have a different interconnects and processors, we expect similarities between *Joliot-Curie Rome* and *Discoverer*, as both utilize the same version of Open MPI.

In the experiments detailed in this paper, we compare the performance guidelines of blocking collectives with fixed data sizes against different mock-ups and implementations from other collective libraries, specifically hierarchical and lane collectives [11] and circulant collectives [12] (note that we do not test the $-v$ or $-w$ versions, such as `MPI_Allgatherv`). We measured the runtime of all collectives with 1, 10, 100, 1000, and 10 000 B, and for fixed-sized collectives (e.g., `MPI_Bcast`) we also tested with 100 000 and 1 000 000 B (to avoid excessive memory use with collectives like `MPI_Allgather`). These sizes were chosen to cover a wide range of potential payloads, ensuring that collectives are tested beyond just powers of two. This, however, is just a snapshot; the tool is capable of testing any other message size.

Since ReproMPI supports fixed time-interval measurements for a predictable benchmarking process, each collective call was repeatedly measured for 3 s. Then, we compute the ratio of the median runtimes of the default MPI library call and the competing guideline call for each message size. We then categorize the results by severity. We assign a "low" severity label to collective calls that are less than 10 % slower than a guideline call. The other labels are medium, medium high, high, and very high, which represent a maximum runtime slowdown of 50 %, 100 %, 400 %, or more than 400 %, respectively.

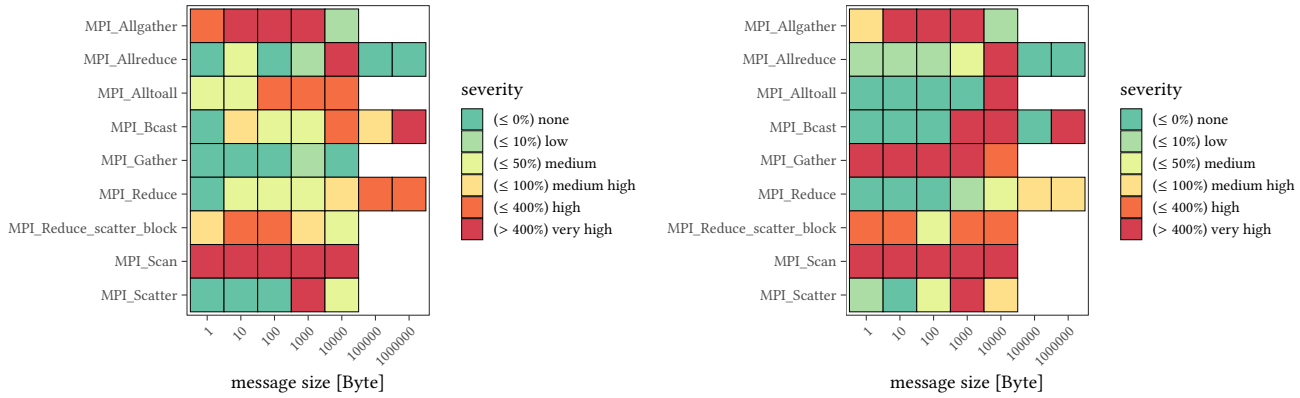
Overview of Guideline Violations. Now, we present a bird's-eye view of the results, i.e., we present an overview of the guidelines-compliance of the various collectives in Figure 3a for the supercomputer *Joliot-Curie Rome* and a total of 12 288 (256×48) processes. It can be observed that `MPI_Gather` is the only collective that is unproblematic, as all its message sizes are colored in green. Most other collectives have certain message ranges where performance lags. For example, `MPI_Reduce` with larger messages (e.g., 100 000 B) or `MPI_Bcast` with a payload of 10 000 or 1 000 000 B underperform. Additionally, `MPI_Allgather` or `MPI_Scan` severely violate the guidelines for almost all message sizes. In these cases, the MPI library must either select a different existing algorithm or enhance performance by integrating a novel algorithm to resolve this issue.

We can observe a similar trend in Figure 3b, which presents the guideline results from *Discoverer* with 64×128 processes. Since both machines employ Open MPI 4.1.4, the results for comparable process counts are unsurprising. However, differences exist, e.g., the performance of `MPI_Reduce_scatter_block` on *Discoverer*. In this case, the overall performance has worsened compared to *Joliot-Curie Rome*. A message size of 10 000 B appears to be critical on *Discoverer* and Open MPI, as most library collectives perform much worse than their guideline counterparts.

Note that these results compare the collective algorithms that a library selects to their competitors. Therefore, our tool assesses the status quo of a library. To enhance the overall quality of an MPI library, a tailored library tuning approach must work in tandem with our guideline checking tool.

An example of such a tuning effort is shown in Figure 4, where the performance guidelines for `MPI_Scan` were reassessed after manually altering the algorithm used in Open MPI to the recursive doubling algorithm. This analysis shows a completely different picture compared to Figure 3b. After adapting the internal algorithm for `MPI_Scan`, the MPI library is found to be performance consistent, showing only a small improvement potential for 10 kB.

Comparing Algorithmic Options. A high-level overview might not provide a comprehensive understanding of the performance of MPI collectives. For this reason, `MPINPC` also enables a detailed view into the performance of various algorithms for different message sizes. Figure 5 presents such a view, comparing the median runtime of algorithms for `MPI_Reduce_scatter_block` on *Theta* with



(a) 256 × 48 **processes on *Joliot-Curie Rome*; Open MPI 4.1.4**

(b) 64 × 128 **processes on *Discoverer*; Open MPI 4.1.4**

Figure 3: Overview of performance guideline analysis.

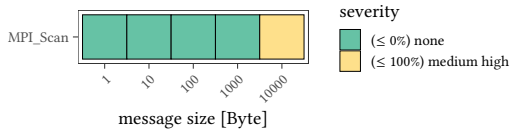


Figure 4: Performance guideline analysis for `MPI_Scan` on *Discoverer* after adapting the internal algorithm; 64 × 128 processes; Open MPI 4.1.4.

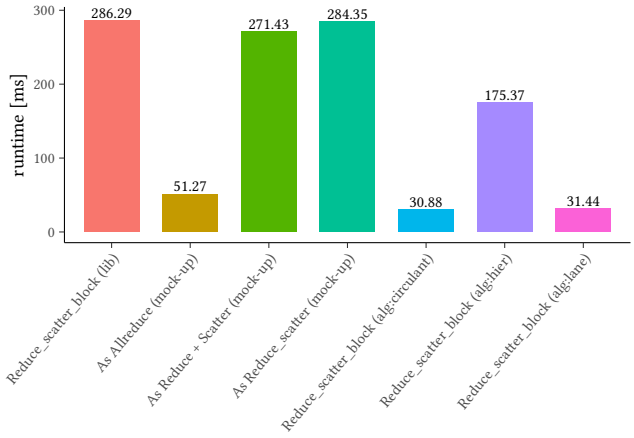


Figure 5: Median runtimes of `MPI_Reduce_scatter_block` implementations obtained on *Theta* with 128 × 64 processes and a message size of 1000 B. The "mock-up"s are re-implementations of the same functionality with other MPI calls, while "alg" denotes an external algorithm.

a configuration of 8192=128×64 processes. The results indicate that the library implementation of `MPI_Reduce_scatter_block` essentially utilizes the same algorithm as `MPI_Reduce_scatter`, which the mock-up version also employs. We can also observe that the mock-up using the highly tuned allreduce call significantly lowers the runtime from 286.29 ms to 51.27 ms. Nonetheless, the fastest algorithms in this case are the circulant algorithm and the lane implementation. These outcomes emphasize two potential directions: (1) the necessity to introduce better algorithms to MPI libraries; and (2) ensuring that newly devised algorithms are rigorously tested against a comprehensive set of possible algorithms. A comparison only to the default MPI call is insufficient to assert the efficiency of novel MPI collective algorithms.

5 CONCLUSIONS

We presented the tool `MPINPC`, which can automatically and efficiently verify the performance guidelines of MPI collectives. The performance results of `MPINPC` can be used to enhance the quality of MPI libraries in the development phase, for example, within the CI/CD pipeline or during the tuning phase, to confirm the efficiency of the decisions made.

Although the tool's original focus is on verifying performance guidelines, it also serves as an ideal program for assessing the performance of novel algorithms for MPI collectives. It easily allows the integration of new algorithms into `MPINPC`.

ACKNOWLEDGMENTS

We thank Maximilian Hagn (TU Wien) for helping us to implement the `MPINPC` tool on top of ReproMPI.

This work was partially supported by the Austrian Science Fund (FWF): project P 33884-N.

We acknowledge PRACE for awarding us access to Joliot-Curie SKL hosted by GENCI@CEA, France and to Discoverer at SofiaTech, Bulgaria.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. In particular, we acknowledge the MPICH project for providing the compute time at ALCF for conducting our research.

REFERENCES

[1] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert A. van de Geijn. 2007. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience* 19, 13 (2007), 1749–1783. https://doi.org/10.1002/cpe.1206

[2] Marcin Copik, Alexandru Calotoiu, Tobias Grosser, Nicolas Wicki, Felix Wolf, and Torsten Hoefler. 2021. Extracting clean performance models from tainted programs. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Jaejin Lee and Erez Petrank (Eds.). ACM, 403–417. https://doi.org/10.1145/3437801.3441613

[3] Sascha Hunold and Alexandra Carpen-Amarie. 2016. Reproducible MPI Benchmarking is Still Not as Easy as You Think. *IEEE Trans. Parallel Distrib. Syst.* 27, 12 (2016), 3617–3630. https://doi.org/10.1109/TPDS.2016.2539167

[4] Sascha Hunold and Alexandra Carpen-Amarie. 2018. Autotuning MPI Collectives using Performance Guidelines. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia)*. ACM, 64–74. https://doi.org/10.1145/3149457.3149461

[5] Sascha Hunold, Alexandra Carpen-Amarie, Felix Donatus Lübbe, and Jesper Larsson Träff. 2016. Automatic Verification of Self-consistent MPI Performance Guidelines. In *Euro-Par 2016: Parallel Processing (LNCS, Vol. 9833)*. Springer International Publishing, 433–446. https://doi.org/10.1007/978-3-319-43659-3_32

[6] Scott Parker, Sudheer Chunduri, Kevin Harms, and Krishna Kandalla. 2018. Performance evaluation of MPI on Cray XC40 Xeon Phi systems. In *Cray UserGroup Proceedings*.

[7] Joseph Schuchart, Sascha Hunold, and George Bosilca. 2023. MPI Process Synchronization in Space and Time. In *Proceedings of the 30th European MPI Users' Group Meeting*. ACM, 1–11. https://doi.org/10.1145/3615318.3615325

[8] Sergei Shudler, Alexandru Calotoiu, Torsten Hoefler, Alexandre Strube, and Felix Wolf. 2015. Exascaling Your Library: Will Your Implementation Meet Your Expectations?. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 165–175. https://doi.org/10.1145/2751205.2751216

[9] Nawrin Sultana, Martin Rüfenacht, Anthony Skjellum, Purushotham V. Bangalore, Ignacio Laguna, and Kathryn Mohror. 2021. Understanding the use of message passing interface in exascale proxy applications. *Concurr. Comput. Pract. Exp.* 33, 14 (2021). https://doi.org/10.1002/cpe.5901

[10] Jesper Larsson Träff, William D. Gropp, and Rajeev Thakur. 2010. Self-Consistent MPI Performance Guidelines. *IEEE Transactions on Parallel and Distributed Systems* 21, 5 (may 2010), 698–709. https://doi.org/10.1109/TPDS.2009.120

[11] Jesper Larsson Träff and Sascha Hunold. 2020. Decomposing MPI Collectives for Exploiting Multi-lane Communication. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*. 270–280. https://doi.org/10.1109/CLUSTER49012.2020.00037

[12] Jesper Larsson Träff, Sascha Hunold, Ioannis Vardas, and Nikolaus Manes Funk. 2023. Uniform Algorithms for Reduce-scatter and (most) other Collectives for MPI. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*.

APPENDIX: ARTIFACT DESCRIPTION/ARTIFACT EVALUATION

ARTIFACT DOI

10.5281/zenodo.8384541

ARTIFACT IDENTIFICATION

The paper presents a tool to automatically verify that MPI libraries fulfill self-consistent performance guidelines. We demonstrate the usefulness of the tool by presenting experimental results on three different supercomputers, which are *Joliot-Curie Rome* (France), *Discoverer* (Bulgaria), and *Theta* (USA).

The computational artifact provided contains a link to the source code of the performance guidelines tool as well as the required input data for running the experiments on each of the machines. The artifact also contains the original dataset that was gathered in our previous experiments.

REPRODUCIBILITY OF EXPERIMENTS

The experimental workflow is detailed in the artifact repository. After installing the performance-guideline verification tool on the respective machine, the experiments can be conducted. To that end, the user needs to write the appropriate batch job files.

Artifact 1 - Figure 3a. We conduct a performance guideline study on *Joliot-Curie Rome*. In particular, we use Open MPI 4.1.4 and 256×48 processes (that is, we start 48 processes and each of the allocated 256 compute nodes). The input file describing which algorithms will be tested is given in the artifact repository.

Expected Results: For each collective, all guideline implementations and the default implementation are benchmarked. There are nine collectives, and for some collectives, eight competing implementations will be tested for up to seven different message sizes. As we test each collective for every message size for a maximum of three seconds, one set of experiments takes at most 25 min. In order to reproduce Figure 3a, the ratios of the median runtime of the default implementation and the competing guideline implementations have to be computed. The expectation is that the violations illustrated in the figure will recur, i.e., the default implementation of `MPI_Allgather` severely violates the performance guidelines for messages in the range from 1 B to 1000 B.

Artifact 2 - Figure 3b. Reproducing this artifact is analogous to the reproduction of Artifact 1 but with experiments conducted on *Discoverer* using Open MPI 4.1.4 and 64×128 processes.

Artifact 3 - Figure 5. Reproducing this artifact on *Theta* requires similar steps as the two previous artifacts. However, only the collective call `MPI_Reduce_scatter_block` is verified, which reduces the expected runtime to roughly 2 min. In addition, the experiment requires Cray MPICH 7.7.14 and 128×64 processes.