



Electronic Communications of the EASST
Volume 86 (1) Year 2026

deRSE26 - Selected Contributions of the 6th Conference for Research Software Engineering in Germany (Part 1)

Edited by Bernd Flemisch, René Caspart, Bernadette Fritzsch, Florian Goth, Jean-Noël Grad, Florian Thiery

Easy, Collaborative Benchmarking with the Extended Benchmarking Automation Tool xbat

Nico Tippmann, Ziwen Su, Willi Leinen, Ruben Horn, Amartya Das Sharma, Shashini Wanniarachchi, Andreas Fink, Axel Auweter, Philipp Neumann

DOI: 10.14279/eceasst.v86.1.2791

License: This article is licensed under a CC-BY 4.0 License.

Electronic Communications of the EASST (https://eceasst.org).
Published by **Berlin Universities Publishing**
(https://www.berlin-universities-publishing.de/)

Easy, Collaborative Benchmarking with the Extended Benchmarking Automation Tool xbat

Nico Tippmann[1], Ziwen Su[1], Willi Leinen[2], Ruben Horn[2], Amartya Das Sharma[2], Shashini Wanniarachchi[2], Andreas Fink[2], Axel Auweter[1], Philipp Neumann[3 4]

[1]MEGWARE Computer Vertrieb und Service GmbH, Chemnitz, Germany

[2]Helmut Schmidt University, Hamburg, Germany

[3]Universität Hamburg, Hamburg, Germany

[4]Deutsches Elektronen-Synchrotron, Hamburg, Germany

Abstract: Research software development for compute- and data-intensive problems increasingly requires balancing scientific objectives with performance optimisation, performance reproducibility, and resource efficiency. However, many research software engineers (RSEs) face barriers in accessing the hardware expertise and tools necessary for performance analysis, especially in high-performance computing (HPC) environments where understanding low-level metrics is essential for sustainable software development. Moreover, collecting and interpreting performance metrics can be challenging and error-prone, particularly for RSEs without in-depth hardware expertise. To address these challenges, we present the open-source software extended benchmarking automation tool (xbat) as an easy-to-use, software-agnostic tool to automate (continuous) benchmarking and to simplify performance analysis. xbat unifies the benchmarking lifecycle from job configuration to analysis in a single, accessible platform and enables collaborative benchmarking by sharing results and configurations to facilitate replication within projects. We focus on the architecture, capabilities, and features of xbat, demonstrating its ability to simplify (continuous) benchmarking and assist RSEs in improving the performance of their research software. In addition, we present the xbat command line interface (CLI) which allows easy integration into non-interactive workflows. We showcase four use cases, namely the closed-source Gurobi Optimizer software, a co-scheduling example of the memory-bound application Stream with the compute-bound application Linpack, the open-source molecular dynamics research software ls1 mardyn using multiple nodes, and lbmpy for fluid simulation on a graphics processing unit (GPU).

Keywords: Performance Analysis, Benchmarking, High-Performance Computing, Collaboration, Command Line Tool

1 Introduction

High-performance computing (HPC) requires in-depth understanding of hardware resources and their utilisation by application software. To facilitate this, there are many benchmarking and profiling tools available such as LIKWID [THW11], perf [Lin24], MAQAO [DBC+05], JUBE [BWS+24], ReFrame [KMR+20], NVIDIA Nsight Systems/Compute [NNS26, NNC26], HPCToolkit [ABF+10], TAU [SM06], Intel VTune Profiler [IVT26] and ClusterCockpit [EGA+19]. Some of these tools require complex setups, have a narrow scope, or have a demanding initial learning phase.

Tool	**Primary Focus**	**License**	**Collaborative Benchmarking**	**GUI**
LIKWID	Hardware counter analysis	MIT	No	No
perf	Kernel and runtime tracing	GPL-2.0	No	No
MAQAO	Static and dynamic analysis of executable binaries	LGPL-3.0	No	HTML report
JUBE	Generic benchmark orchestration and analysis	GPL-3.0	Reproducible benchmarking	No
ReFrame	Regression testing framework	BSD-3-Clause	Reproducible benchmarking	No
NVIDIA Nsight	CUDA GPU and system profiling	Proprietary	Manual export/import	Desktop
HPCToolkit	Scalable parallel profiling	Apache-2.0	Portable database	Desktop
TAU	Scalable parallel profiling	BSD-style	Database-backed (TAUdb)	Desktop
Intel VTune	Microarchitectural and threading analysis	Proprietary	Hierarchical (VTune Profiler Server)	Desktop, Web
ClusterCockpit	Automatic job profiling	MIT	Project- and role-based collaboration	Web
xbat	Automated, continuous HPC performance benchmarking platform	MIT	Project-based collaboration	Web

Table 1: Comparison of benchmarking and profiling tools. (The column *Collaborative Benchmarking* summarizes built-in features for managing, reproducing, and sharing benchmark experiments or results across multiple users.)

Developed by MEGWARE Computer Vertrieb und Service GmbH, the extended benchmarking automation tool (xbat) [TAD+24, LDF+25, TSL+26b] aims to offer easy, comprehensive, collaborative, repeatable benchmarking. It has been fully open source since April 2025 and remains under active development, with extensive online documentation, guides, and a public demo instance. A comparison between xbat and other tools is given in Table 1. On the one hand, low-level profiling tools are "easy to pick up", but do not offer higher-level features that facilitate easy replication or visual analysis of experiments. Extracting multiple metrics often leads to a complex, bespoke experiment setup. Unlike benchmark orchestration frameworks such as JUBE and ReFrame, performance analysis tools such as LIKWID, perf, and MAQAO primarily provide measurement and analysis rather than portable, shareable benchmark workflows. Meanwhile, NVIDIA Nsight sessions can be stored as project files for reproducible profiling configurations or report files for analysis on different systems. On the other hand, high-level tools that themselves offer sharing of benchmark configurations or results are inherently more complex in terms of setup or usage. For pure orchestration solutions like JUBE or ReFrame, users must provide code to extract metrics from the application output, while xbat collects various metrics over time automatically. In the context of HPC, a desktop-based graphical user interface (GUI) exacerbates this by often requiring X11-forwarding, remote desktop or file transfer for analysing benchmark results. While such "traditional" tools therefore have a place as expert-driven analysis utilities, xbat is a universal performance management platform offering frictionless adoption by novice cluster users through a shared web portal. This avoids the need to support different workstation operating systems, has significantly better performance compared to X11-forwarding or remote desktop when used over the internet, and ensures development flexibility. The most similar solution is arguably ClusterCockpit, which also offers automatic monitoring of user jobs and a comprehensive web GUI. However, unlike xbat, it does not offer advanced support for parameter studies through a job submission interface. Instead, ClusterCockpit is designed primarily as a general cluster monitoring tool rather than a dedicated job-monitoring solution: while it attributes data to jobs to enable a job view, its core purpose is cluster-wide observability rather than in-depth, per-job analysis. Both tools consequently gather considerably fewer metrics and typically at a much lower sampling frequency than xbat, which is tailored for detailed, high-resolution benchmarking of individual experiment runs and their comparisons. Additionally, xbat supports collaborative benchmarking, consisting of two parts: First, sharing benchmark configurations and setups for reproducible results, and second, sharing benchmark results for comparison and analysis.

This paper is structured as follows: Section 2 discusses the xbat software. In Section 3, we showcase several xbat features for various applications. Our conclusions as well as an outlook on future and ongoing work are presented in Section 4.

2 xbat

The xbat software is presented in this section. In particular, Section 2.1 discusses the xbat architecture, Section 2.2 focuses on the capabilities and features of xbat, whereas Section 2.3 introduces the xbat CLI and Section 2.4 describes generating performance plots using the roofline model [WWP09].

2.1 Architecture

The current xbat architecture is shown in Figure 1 and mainly consists of two services that separate the central management stack from the on-demand metric collection on compute nodes. The large group on the left side of the figure represents the main `xbat.service` stack, which is typically deployed via podman-compose on a management or login node. It hosts the user-facing frontend, application programming interface (API), cache, databases, and management daemon, although the databases can optionally be deployed separately. Users access xbat through the Nuxt frontend, which communicates with the Flask-based REST-API. Valkey is used as an API cache, MongoDB stores user, job, and configuration data, and ClickHouse stores time-series metric data. The management daemon `xbatctld` coordinates benchmark jobs and their state through the REST-API and integrates xbat with site-provided cluster infrastructure, in particular Slurm and the configured authentication provider. The `xbatd.service`, in contrast, runs on compute nodes on demand. It is launched via Slurm Prolog and Epilog scripts only when an xbat-related job is scheduled on the node. After startup, the collector daemon retrieves its instructions through the REST-API and forwards the collected performance metrics to the ClickHouse metric database. Compared to prior publications on xbat [TAD+24], the architecture has been updated, replacing QuestDB with the more scalable [KKD+23, SMK26] ClickHouse as the metric (time-series) database. QuestDB's built-in compression capabilities are limited, requiring reliance on external solutions such as the ZFS filesystem to manage storage efficiently. ClickHouse, when properly configured, provides significantly better compression ratios, simplifies deployment, and offers quality-of-life features absent in QuestDB – most notably native `DELETE` support, which enables fine-grained management of user data across large-scale instances. These drastically reduced storage requirements[1], paired with a revised and streamlined installation and setup process, finally make the software viable for production environments. Its containerised deployment model and integration with existing user management tools ensure a seamless integration of xbat into existing HPC infrastructures. Building on these improvements, the system can scale to accommodate large and heterogeneous workloads. Using this architecture, the number of nodes (belonging to the same or different jobs) is limited only by available storage space for the metrics and the write rate of the metric database (or its network). As the data volume is directly proportional to the number of nodes, the post-processing and visualisation may require more resources and time.

2.2 Capabilities

xbat is designed with a strong focus on automation to reduce both effort and susceptibility to error in the benchmarking process. Job scripts can be configured directly in the web-based GUI (Figure 2), which not only lowers the barrier for novice users but also unlocks extended capabilities such as generating multiple job variants by iterating over a set of user-defined variable values, enabling systematic parameter studies and direct performance comparisons between configurations.

Job submission is possible via the GUI or, without any required changes to existing work-

[1] Around 20 MB/hour per node (equals 150 MB/hour uncompressed) were measured on a dual-socket node with Intel Xeon Platinum 8360Y processors. The amount of data collected may vary depending on the hardware.

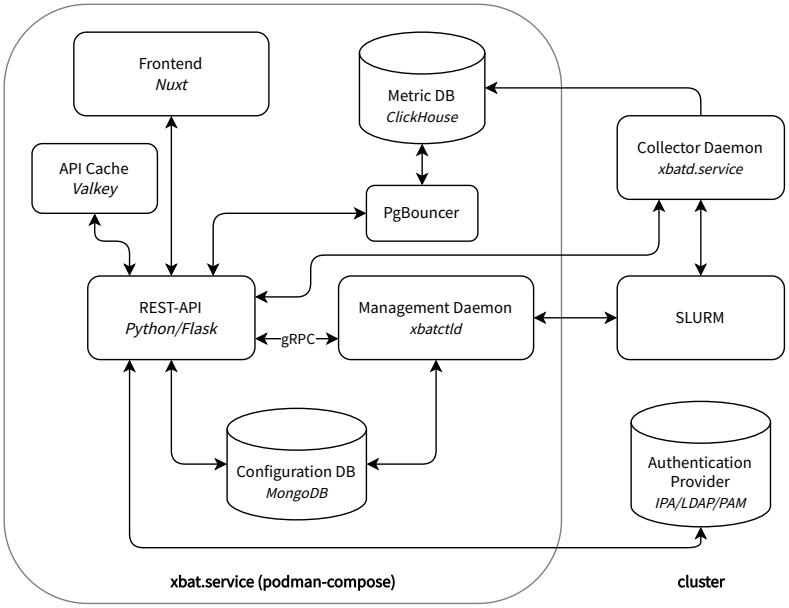

Figure 1: The current xbat architecture (**left:** The main `xbat.service` stack, typically deployed via podman-compose on a management or login node, **right:** Cluster-side components and services outside the main `xbat.service` stack). Arrows indicate the main control flow and metric-data paths.

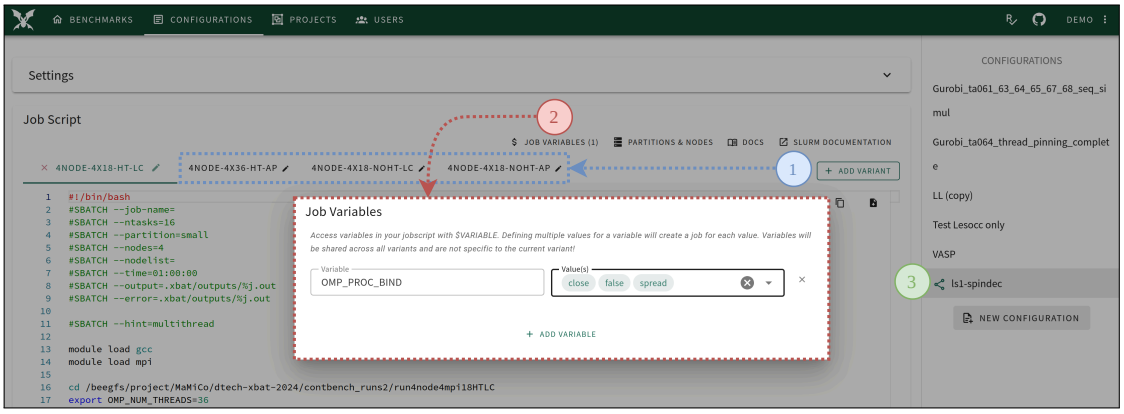

Figure 2: Screenshot of xbat benchmark configuration page showing: **(1)** multiple job script variants for one benchmark configuration, **(2)** job variables, which are expanded into different job variants, and **(3)** configurations shared between multiple users within projects.

flows as an *ad hoc* benchmark, via the Slurm command line interface (CLI) by adding the `--constraint=xbat` flag to the `sbatch` command. Regardless of the submission method, each submission creates a *benchmarking run* in xbat, which contains one or more jobs – the latter being the case when a job script with multiple variants is submitted. The high degree of automation makes xbat suitable for use in continuous integration (CI) pipelines, allowing performance regressions to be directly related to code changes. For example, a CI workflow may build a new version of a software and provide it through a shared file system or artefact repository. Using the REST-API of xbat, a prepared benchmark is started and optionally receives additional variables. The resulting job output and recorded metrics help relating code alterations to performance changes.

The collected measurements are displayed in an extensive, customisable, and interactive graph system. Graphs can be exported as PNG or SVG files, and the underlying data can be downloaded in JSON and CSV format. Additionally, xbat provides feedback on the power consumption of a job and performs micro-benchmarks via likwid-bench [THW12] to measure empirical peak values of the bandwidth of caches and dynamic random-access memory (DRAM) as well as floating point operations per second (FLOPS) of the hardware used. Jobs can be compared against each other and against the empirical peak values. Integrated statistics are supported, as well as automatic roofline modelling based on the empirical peak values.

A collaboration feature, organised into a project system, allows users to share measurements and job configurations with others for review and replication. To effectively manage configurations, they can be organised into nested folders with the option for manual backup/restoration to/from the local file system. In addition, extensive documentation, guides, and a public demo instance are available online[2].

The xbat daemon (`xbatd`) collects up to 140 performance metrics on the compute nodes, depending on hardware availability. The bulk of these measurements is provided by LIKWID via hardware performance counters. Other metrics are derived from the `/proc` filesystem, Intelligent Platform Management Interface (IPMI) or vendor-specific tools like nvml or rocm. With this large set of data sources, xbat provides a diverse set of metrics covering central processing unit (CPU), memory, I/O, interconnect, graphics processing unit (GPU), field-programmable gate array (FPGA) and energy consumption. To provide users with the highest possible and most in-depth insight into the behaviour of their application, all measurements are performed at the lowest available hardware level, which in most cases is an individual hardware thread, but depends on the concrete metric or device. xbat then aggregates these metrics across a hierarchy of levels, from fine-grained to coarse-grained as listed in Table 2. This multi-level view allows users to pinpoint performance bottlenecks at exactly the granularity they need. The overhead introduced by these measurements is below 1 %. Additionally, monitoring can be fully disabled (e.g., when using xbat solely for parameter studies via job variables and variants), restricted to high-level metrics, or complemented with fine-grained measurements including hardware performance counters.

[2] xbat project website: https://xbat.dev/

Level	Measurement Scope	Metric Type
Thread	Individual hardware threads	CPU, Cache, Memory, Energy, I/O, Inter-connect
Core	Physical CPU cores	
NUMA domain	Memory locality domains	
Socket	Physical CPU packages	
Device	E.g., GPU, FPGA or storage devices	
Node	The full compute node	
Job	Across all nodes of a job	

Table 2: xbat measurement levels for different metric types. (Some metrics may support different scopes compared to the general range of their type.)

2.3 Command Line Tool

xbat provides a user-friendly web-based GUI for defining and executing benchmarks as well as inspecting the results. However, many HPC users are accustomed to interacting with computing resources primarily through CLIs on the terminal. Regular Slurm jobs can be set up for instrumentation through xbat by including a single argument in the `sbatch` command, as described above. However, analysing and processing the results for many jobs through the GUI can be cumbersome. Furthermore, benchmark data should be downloadable for use with post-processing and plotting scripts for consistent and comparable analysis. A Typer-based [Ram25] companion command line tool has been developed, which leverages the xbat API to provide essential features from the GUI directly on the terminal and for easier integration into non-interactive workflows. This CLI replaces manual port forwarding, HTTP requests and authentication token handling, thereby greatly simplifying the headless interaction with xbat features. By replacing verbose `curl` commands with short, semantic xbat CLI commands, automation scripts also become more readable and easier to maintain. The implemented commands and corresponding xbat API features are listed in Table 3.

Command	Functionality
`login`	Authenticate against the xbat API.
`ls`	List benchmark runs or constituent Slurm jobs.
`rm`	Delete benchmark runs.
`log`	Print the output and error logs of a Slurm job.
`pull`	Download measurements for a finished Slurm job.
`roofline`	Print the roofline model data of one or more finished Slurm jobs.
`start`	Start a new benchmark run from an xbat config or a Slurm job script.
`stop`	Stop benchmark runs or Slurm jobs.
`export`	Create a backup of benchmark runs.
`gui`	Open the GUI in the browser. (Optionally for a benchmark run or Slurm job.)
`plot`	Create rudimentary visualisations of downloaded benchmark results.

Table 3: xbat CLI commands.

This CLI may be used on the cluster running xbat, but also on a local workstation or in a CI workflow container (replacing the direct REST-API usage mentioned above) for external benchmark result processing as shown in Figure 3. For network isolated xbat installations, secure shell (SSH) port forwarding is transparently applied through the Fabric library [For26]. The target xbat instance and SSH proxy are specified through environment variables. While the web GUI remains the core interface of xbat, the CLI is a complementary method for console-based and non-interactive workflows, which provides the most essential features of xbat.

```
1  $ xbat login                                  # --ci to use environment

2a # Start benchmark config
   $ xbat start config $BENCHMARK_CONFIG_ID
2b # Start Slurm job
   $ xbat start job job.sh                       # Prints $JOB_ID
3  $ xbat ls -j | grep $JOB_ID                   # Check job state
   $ xbat gui $JOB_ID                            # View job in web GUI
```

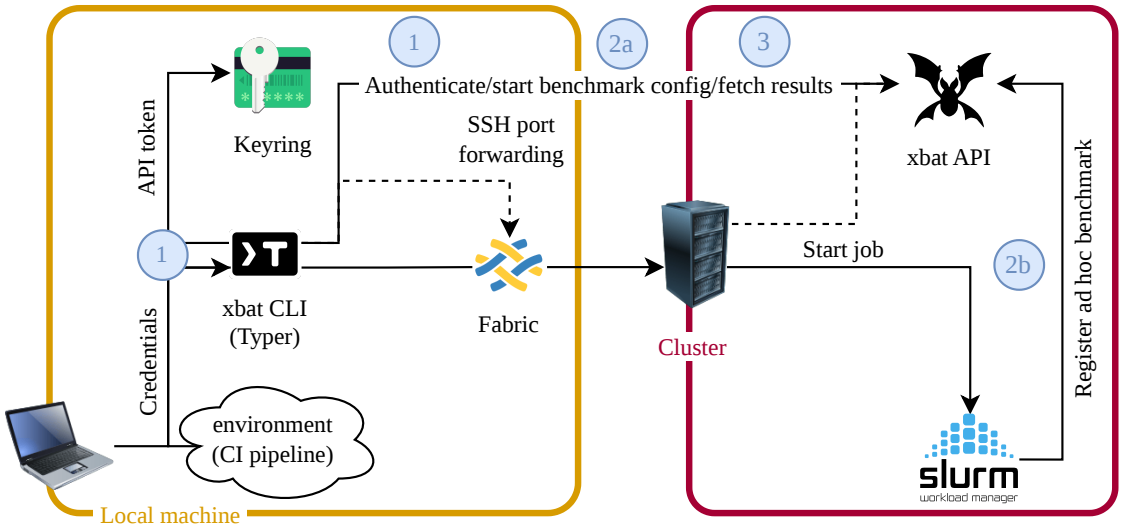

Figure 3: Visualisation of the key features of the CLI for xbat and the components involved: 1. authenticate against xbat API, 2. start a benchmark run, and 3. obtain the results for further processing.

2.4 Roofline Model

The xbat web-based GUI offers automatic roofline plots to assess and compare benchmark runs. For more customisability and generating figures for publications, the roofline model data can be obtained through the CLI in Section 2.3. Rudimentary plotting is also provided with different aggregation methods of performance and operational intensity.

The empirical peak performance of the CPU and the empirical peak memory bandwidth are

used in xbat to form the ceiling of the roofline model. Depending on the level of vectorisation (scalar, SSE, AVX, AVX512), the usage of fused multiply-add (FMA) (only in combination with AVX and AVX512) or data type (single-precision (SP) or double-precision (DP)), a different empirical peak value is obtained. Hence, xbat allows plotting multiple empirical peak performance values of the CPU. For each instrumented job, the measurements are aggregated to a single point below the roofline. This data is then visualised as a scatter plot.

Let there be n measurement intervals and xbat measures for each interval X_j with $j \in \{1,\ldots,n\}$ and length (i.e., time elapsed) t_j (with $t_j = t_k$ for all $k \in \{1,\ldots,n-1\}$), the FLOPS rate f_j, and the amount of written and read bytes via DRAM b_j, i.e., no bytes related to caches are counted. We define

$$\mathcal{V} := \{\, j \in \{1,\ldots,n\} \mid f_j > 0 \text{ and } b_j > 0 \,\}$$

as the set of all valid measurement intervals. The pairs $X := (I,P)$ with operational intensity I (measured in floating point operations (FLOPs) per byte) and performance P (measured in gigaFLOPS (GFLOPS)) that are used in different types of roofline models are then computed with

$$I_j = \frac{f_j t_j}{b_j}, \qquad P_j = \frac{f_j}{10^9} \tag{1}$$

for all valid measurement intervals $\mathcal{V}$.

The *maximum* aggregation roofline plot uses $X_{\max}$ with the maximum point $j = j_{\max}$ in Equation (1) which is computed with

$$j_{\max} := \arg\max_{j\in\mathcal{V}} P_j, \qquad X_{\max} := X_{j_{\max}} \tag{2}$$

and represents the actual interval point with maximal performance for the instrumented job.

The *median* aggregation roofline plot uses the median point $j = j_{\text{med}}$ in Equation (1) which is the actual interval point whose performance is closest to the median performance:

$$j_{\text{med}} := \arg\min_{j\in\mathcal{V}} \left| P_j - \text{median}(\{P_k \mid k \in \mathcal{V}\}) \right|, \qquad X_{\text{med}} := X_{j_{\text{med}}}.$$

The *average* aggregation roofline plot uses a component-wise average over all measurement intervals in Equation (1):

$$X_{\text{avg}} := \left(\frac{1}{|\mathcal{V}|} \sum_{j\in\mathcal{V}} I_j, \frac{1}{|\mathcal{V}|} \sum_{j\in\mathcal{V}} P_j \right). \tag{3}$$

The *total* aggregation roofline plot uses aggregated FLOPS, bytes, and time values:

$$X_{\text{tot}} := \left(\frac{\sum_{j\in\mathcal{V}} f_j t_j}{\sum_{j\in\mathcal{V}} b_j}, \frac{\sum_{j\in\mathcal{V}} f_j t_j}{10^9 \sum_{j\in\mathcal{V}} t_j} \right).$$

Unlike the maximum, the three other aggregation methods median, average, and total summarise the overall performance of the benchmarked software with different sensitivity to outliers. The choice of aggregation method may also depend on the specific application being benchmarked. Figure 12 shows an illustrative example for visualising multiple benchmark runs as well as the difference between roofline plots using maximum and average aggregation. Since this headless analysis can be scaled to an arbitrary number of benchmark runs, the overall performance of HPC applications can be better understood.

2.5 Limitations

xbat can only be used on clusters running the Slurm workload manager; however, Slurm is very widely adopted [BEA+18]. Furthermore, it requires root access to handle data collection and job submission. At the moment, only x86-architectures and RPM-based distributions are supported. The majority of metrics are provided by the LIKWID library, which means that the full range of monitoring can only be achieved if the underlying hardware is also supported by LIKWID. Additionally, xbat cannot be used in conjunction with other tools which rely on hardware counters, such as ClusterCockpit and some LIKWID features.[3] The minimum measurement interval is five seconds, which allows obtaining multiple samples per metric over the entire runtime of the job as well as observing different phases of the application, but is not sufficient for investigation of high frequency simulation loops. While heterogeneous clusters are supported, xbat assumes that the node hardware is homogeneous within a single multi-node job, for now. Moreover, the empirical peak values (FLOPS and memory/cache bandwidth) are only measured at the node level and then scaled down to the lower levels of the hardware hierarchy, assuming linear scaling.

xbat has been deployed on multiple production HPC clusters for CI workflows or user benchmarks as well as internal MEGWARE testing infrastructure. The concrete capabilities depend on the available cluster hardware, in particular on whether and to what extent it is supported by LIKWID.

3 Use Cases

In our four use cases, we demonstrate various xbat features and illustrate its versatility and practicality. Our experiments are performed on the HPC cluster HSUper within the dtec.bw[4] projects hpc.bw [NDK+22] and MaST [NDV+22]. The closed-source Gurobi Optimizer software is used in Section 3.1 to show the GUI of xbat, its ability to show metrics at different levels and the ability to compare measurements for selected metrics with empirical peak values of the hardware used. The co-scheduling use case in Section 3.2 uses the memory-bound application Stream and the compute-bound application Linpack and illustrates the compare and statistics features of xbat. Various types of roofline models and multi-node jobs are shown in Section 3.3 for the open-source molecular dynamics research software ls1 mardyn. Section 3.4 demonstrates the comparison of different methods for the same GPU-based simulation using lbmpy.

3.1 Gurobi Optimizer

The Gurobi Optimizer software is used in mathematical optimisation, for example in mixed integer programming (MIP). It is a closed-source commercial solver that is usually used on machines (or one node of a cluster) with a moderate number of cores (i.e., with the default parameters it has a soft limit of 32 threads). Our use case is to study the usage of the Gurobi Optimizer on HPC hardware [PDD+24]. The experiment in Figures 4 and 5 shows the solution of the MIP instance ta064 [FV03]. Gurobi Optimizer version 13.0.0 with non-default parameter

[3] likwid-perfctr, likwid-bench or likwid-perfscope, see https://xbat.dev/docs/user/likwid, unless hardware performance monitoring in xbat is turned off to avoid interference when accessing hardware counters.
[4] dtec.bw – Digitalization and Technology Research Centre of the Bundeswehr project website: https://dtecbw.de

Threads=-1 is used to allow the Gurobi Optimizer to utilize all available hardware resources instead of only up to 32 threads.[5]

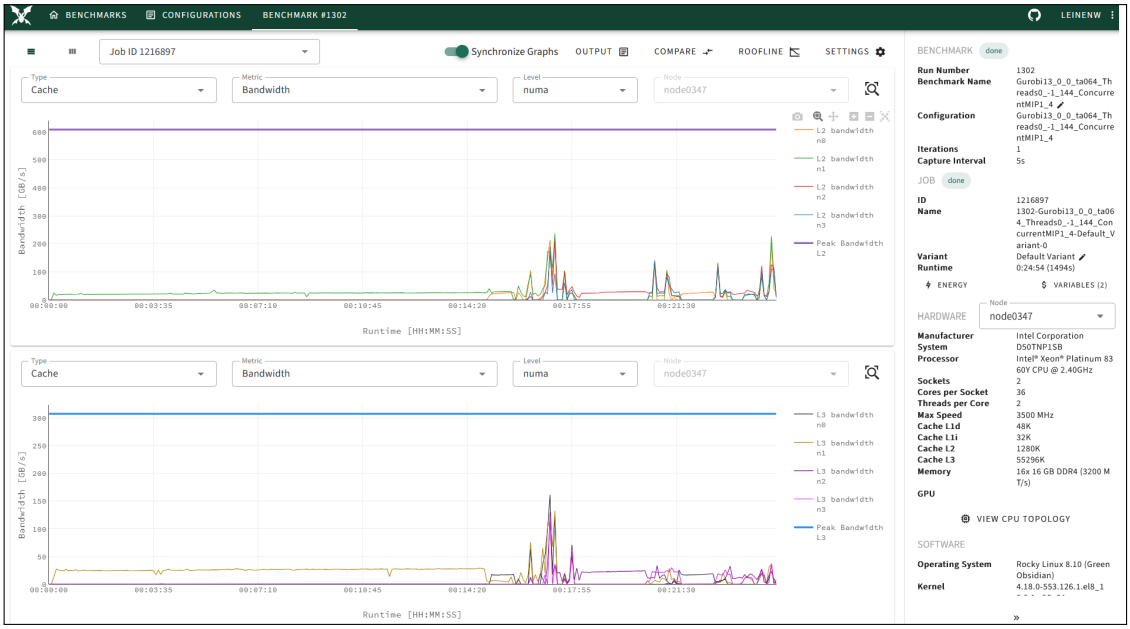

Figure 4: xbat screenshot of a Gurobi Optimizer run.

The HSUper HPC cluster consists of dual-socket nodes with Intel Xeon Platinum 8360Y processors. In the default configuration each socket has 36 cores and simultaneous multi-threading (SMT) is enabled, i.e., 144 hardware threads are available per node. xbat helped us to understand the hardware usage of the Gurobi Optimizer software on an HSUper node [LFN25].

A screenshot of the xbat GUI for a Gurobi Optimizer run is given in Figure 4. The top plot shows the L2 cache bandwidth at NUMA domain level and a comparison with the empirical peak value. The bottom plot is similar to the top plot except that the L3 cache bandwidth is shown. Both plots demonstrate that the measurements for the Gurobi Optimizer are rather irregular and far below the empirical peak values. During several phases, one NUMA domain accounts for most of the cache bandwidth usage.

Figure 5 was exported via the xbat GUI and reports measurements for the L2 cache bandwidth at core level and a comparison with the empirical peak value. The graph shows that, at the core level, individual cores can achieve L2 cache bandwidth values close to or even slightly higher than the empirical peak value (in contrast to the measurements in Figure 4 and in [LFN25]). However, the imbalance is even more pronounced at the core level than at the NUMA domain level. During several phases, a single core accounts for most of the cache bandwidth.

Based on our analyses with xbat, we conclude that oversubscription of resources and co-scheduling of multiple Gurobi Optimizer instances on one compute node are viable options and could demonstrate that this can result in significant speedups and reductions in energy consump-

[5] Parameter documentation at: https://docs.gurobi.com/projects/optimizer/en/current/reference/parameters.html

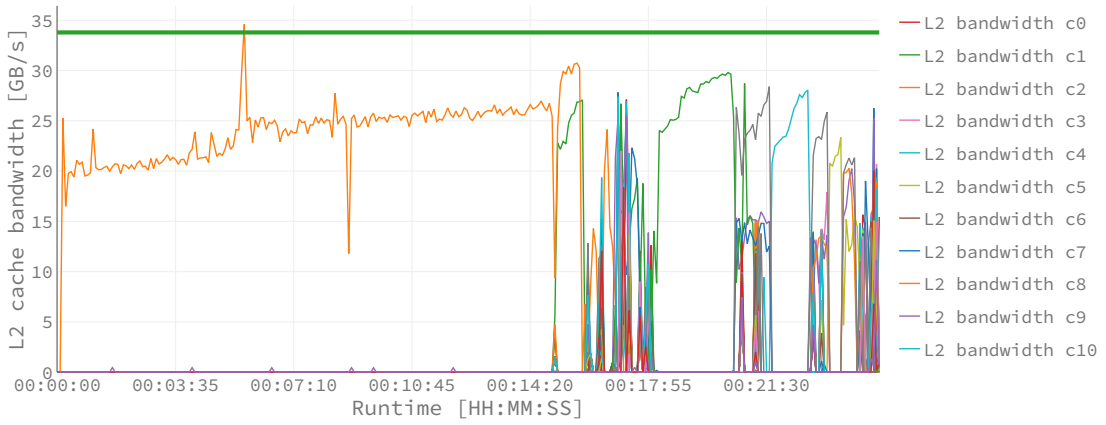

Figure 5: xbat visualisation of the L2 cache bandwidth at core level of a Gurobi Optimizer run.

tion compared to solving one MIP instance after another when calculating ensembles of optimisation problems [LFN26, LFN].

3.2 Co-Scheduling

Improving energy efficiency in HPC systems has become an increasingly important research topic, and co-scheduling has emerged as a promising solution. Modern HPC nodes consist of many cores with shared cache hierarchies and memory channels. However, HPC applications often exhibit significantly different resource utilisation characteristics. Memory-bound applications typically exhibit high cache miss rates and saturate memory bandwidth, while compute-bound applications mainly stress the processor's compute units with relatively low memory usage. As a result, memory-bound jobs spend more energy on data movement, whereas compute-bound jobs consume more energy on computation. These contrasting characteristics create an opportunity to co-schedule complementary workloads on the same node, leading to improved resource utilisation and potentially better energy efficiency.

However, evaluating such co-scheduling scenarios is challenging because multiple performance and energy metrics must be considered together. Although such information can be collected using hardware performance counters, configuring and interpreting these counters often requires significant expertise. xbat simplifies this performance analysis and enables users to easily collect, visualise, and compare metrics across multiple workloads and execution scenarios. As an example, Stream [McC95], a memory-bound benchmark, together with Linpack [DL11], which is compute-bound, is considered. The two applications are executed simultaneously occupying a single node on HSUper. Figures 6 to 8 depict the result plots obtained from xbat for the individual runs of Stream and Linpack applications in comparison to the simultaneous run.

In terms of energy-related analysis, which is the key focus in co-scheduling, Slurm Workload Manager in general provides aggregate CPU and memory energy consumption values in job outputs as an average value. However, understanding the individual contribution of these components is important for gaining deeper insights into workload behaviour. Figures 6 and 7 illustrate the variability in power consumption for the CPU and DRAM individually, as well as

for the overall system.

The comparison in Figure 6 shows that Stream, being a memory-bound application, exhibits higher DRAM power consumption, whereas Linpack, as a compute-bound application, consumes more CPU power. Furthermore, Figure 7 depicts that the co-scheduled execution draws slightly higher instantaneous system power. However, the energy consumption and runtime of the individual executions must be accumulated when comparing against the co-scheduled run. Compared with the total runtime of individual executions, a runtime reduction is observed in the co-scheduled scenario. This outweighs the increase in power, resulting in lower overall energy consumption for the co-scheduled scenario than for the sum of the individual executions. In addition to the comparison plots for power consumption, xbat also reports the total energy consumption in kWh. In this example, the co-scheduled run recorded 0.51 kWh while the sum of energy consumption from the individual runs of Stream and Linpack was 0.53 kWh representing an energy consumption reduction of approximately 4 % when using co-scheduling.

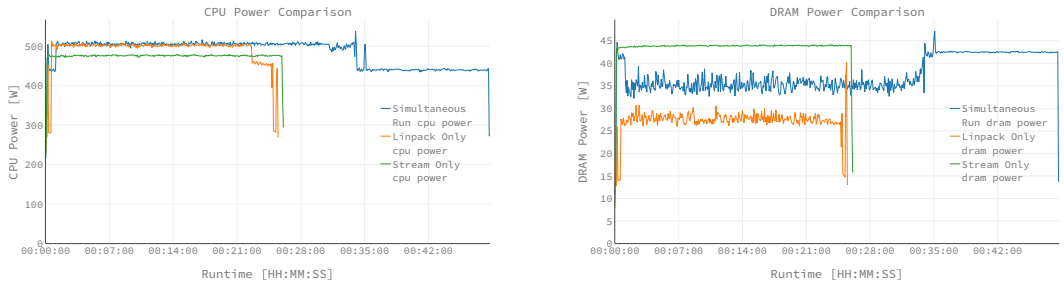

Figure 6: xbat visualisation of the comparison of job-level CPU power (left) and DRAM power (right) for the simultaneous and individual runs.

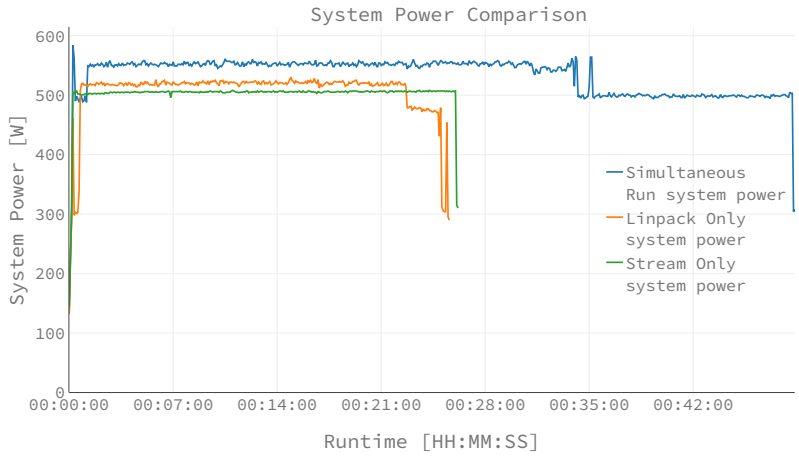

Figure 7: xbat visualisation of the comparison of job-level system power for the simultaneous and individual runs.

Two other important metrics in this context are the memory bandwidth and L3 cache behaviour to evaluate the effectiveness of co-scheduling. Although the job output reports the best

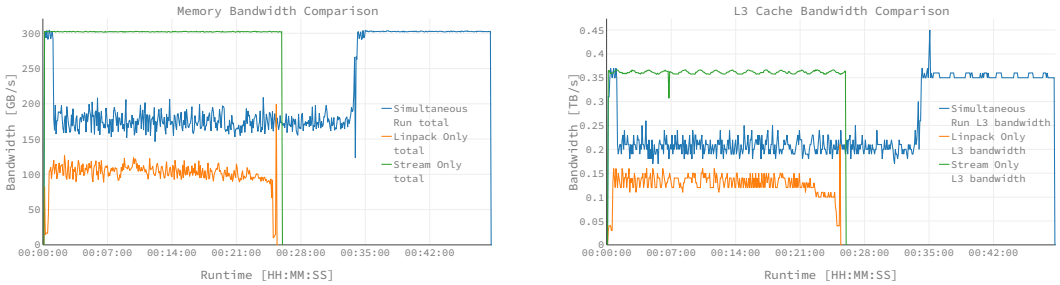

Figure 8: xbat visualisation of the comparison of job-level memory bandwidth (left) and L3 cache bandwidth (right) for the simultaneous and individual runs.

GRAPH STATISTICS EXPORT

Metric (GB/s)	Trace	Min	Max	Avg	Sum	Median	StdDev	Variance
› write (3)								
⌄ total (3)								
	Co-Scheduled *36-36 [0]*	0.01	304.69	214.48	125684.78	182.62	62.5	3905.7
	Linpck *36 Linpack [0]*	0.01	199.5	99.64	30688.89	102.8	21.89	479.16
	stream *36 Stream [0]*	0.01	303.01	298.39	93694.61	302.23	33.89	1148.42

Figure 9: xbat screenshot showing descriptive statistics for memory bandwidth.

bandwidth achieved across all iterations of the application, this value alone is insufficient for analysing the actual variation in memory bandwidth and the resulting resource contention and interference effects between workloads. To this end, Figure 8 presents a detailed analysis obtained using xbat, comparing the total memory bandwidth (read + write) and L3 bandwidth utilisation of the individual executions against the co-scheduled scenario. Furthermore, xbat reports descriptive statistics for each metric per job and Figure 9 shows a screenshot from xbat presenting the memory bandwidth statistics for all three scenarios.

Accordingly, the average memory bandwidth for the simultaneous scenario is approximately 214 GB/s. However, the time-series graph shows that, for the first 36 minutes, the memory bandwidth remains around 175 GB/s. After this point, the bandwidth increases and reaches approximately the same level observed in the Stream-only execution. This behaviour occurs due to Linpack completing its execution around the 35th minute, leaving only Stream running thereafter. Thus, the first 35 minutes demonstrate the effect of resource contention caused by co-scheduling. As both applications simultaneously access shared memory resources, Stream is unable to utilise the memory channels as aggressively as running standalone, leading to a lower observed memory bandwidth. This observation is supported by the higher standard deviation recorded (Figure 9) for the co-scheduled execution, indicating greater temporal fluctuation in memory bandwidth as both applications are competing for shared memory resources. This behaviour is further reflected in the L3 bandwidth utilisation, where the total L3 bandwidth as a sum of the corresponding load

and evict bandwidths is reported.

Overall, the compare and statistics features from xbat facilitate a more intuitive analysis of co-scheduling effects by clearly exposing differences in performance and energy metrics across the standalone and simultaneous execution scenarios.

3.3 ls1 mardyn

ls1 mardyn is a highly scalable [NBB+14] simulation software for molecular dynamics. It uses the auto-tuning molecule container AutoPas [GST+19] at the node level. Its features include efficient vectorised implementations of the Lennard-Jones force, parallelisation at intra- and inter-node level via MPI and OpenMP, and dynamic *load balancing* across nodes. When load balancing (LB) is enabled, ls1 mardyn periodically checks for load imbalances in the simulation and attempts to rectify them. In the *kd-tree* LB scheme, all particles in the domain are counted and the simulation volume is repartitioned in a binary tree-like fashion to ensure equal particle counts in each partition. In the *diffusive* LB scheme, the performance of each subdomain is monitored, and slower subdomains are contracted while faster subdomains are expanded in an attempt to balance the runtimes. In this work, we take this purely geometric approach to subdomain partitioning, and examine how this is reflected in the hardware behaviour.

For this experiment, a $120 \times 90 \times 90$ simulation volume is used and filled with liquid and gaseous argon such that the two phases coexist at the given temperature. This results in a load imbalance, as the liquid volume has $\approx 350,000$ particles, while the gaseous volume has $\approx 10,000$ particles. This is simulated for $100,000$ timesteps, with no LB, kd-tree LB and diffusive LB. Since the load imbalance is static, the kd-tree balancer is run once at the start of the simulation. The diffusive balancer, however, starts with a default grid, and is allowed to reach a balanced state by rebalancing every 500 timesteps. The simulations were run on one node of HSUper, with 72 MPI ranks for saturation.

Figure 10 shows the measured DP FLOPS in the left plot and the DRAM data volume used in the right plot. First, we can see that kd-tree based LB outperforms no LB, which is in turn outperformed by diffusive LB. The left graph in Figure 10 shows constant FLOPS for kd-tree and no LB, while the FLOPS for diffusive LB increases before plateauing, verifying that the rebalancings by the diffusive balancer do lead towards a more balanced scenario. This rebalancing effect can also be seen in Figure 11, showing how the FLOPS per non-uniform memory access (NUMA) domain approach similar values as more rebalancings are performed. The right graph in Figure 10 depicts higher memory usage for the kd-tree balancer. This is caused by the non-cubical partitions created by the balancer, leading to a suboptimal surface-area-to-volume ratio, and as particles are exchanged across the surface area, more memory is used. The usage for the diffusive case is identical to the no LB case initially, as both start from the same subdomain layout.

The executed benchmarks were shared with the project *MaST*. The subsequent performance analysis using the roofline model in Figure 12 was done by a different project user, highlighting the collaborative aspect of xbat. This figure shows a comparison of roofline model plots using different aggregation methods, maximum using Equation (2) vs. average using Equation (3). The roofline is defined by the DRAM bandwidth (BW) limits and scalar DP peak FLOPS. The lack of LB leads to some ranks handling very few particles. Since the data volume of the cache

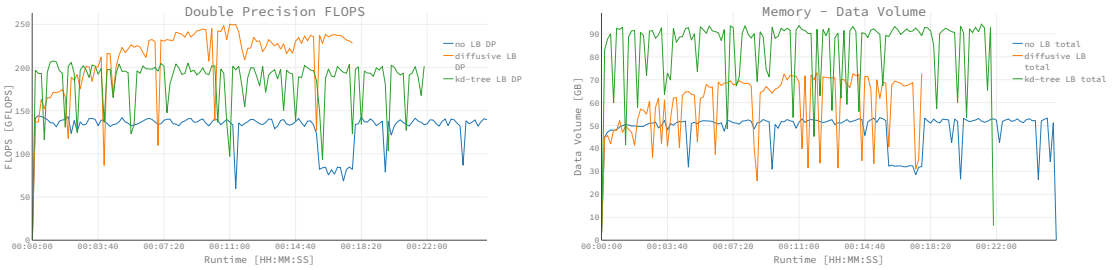

Figure 10: xbat visualisation showing the double precision FLOPS (left) and memory data volume over time (right), for each of the LB schemes, at the job level.

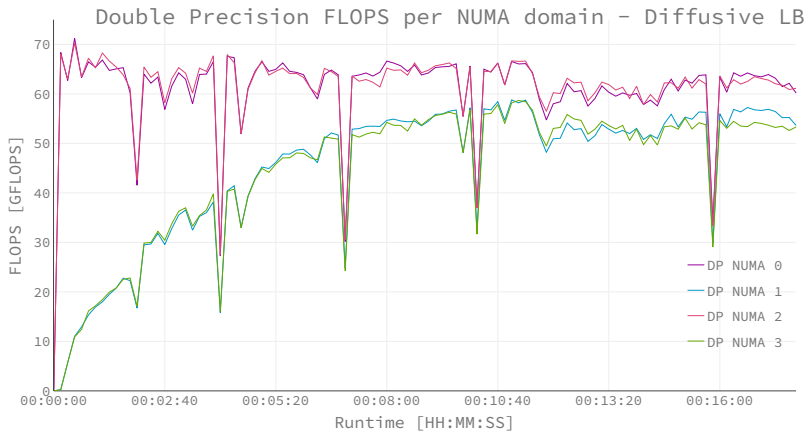

Figure 11: xbat visualisation of the evolution of the FLOPS over time for the diffusive LB per NUMA domain, correlating improved balancing with higher FLOPS.

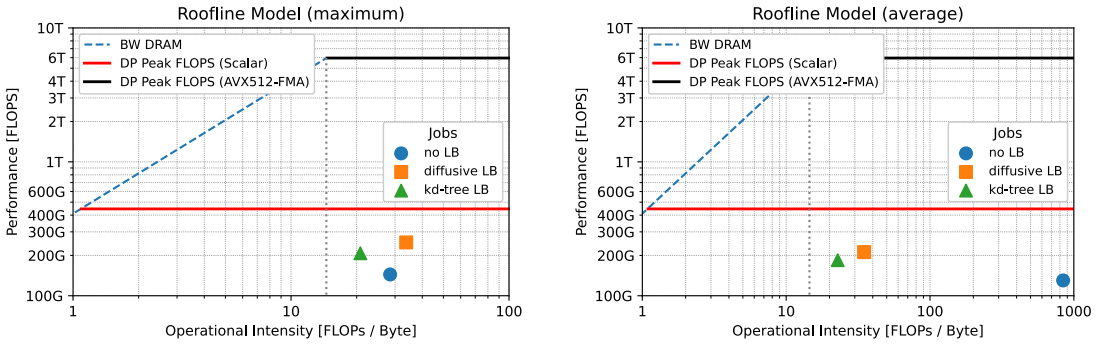

Figure 12: Roofline model plots using different aggregation methods; maximum (left) vs. average (right).

is not considered, that of the memory may therefore be exceptionally low for a few samples as most particle data resides in the cache. This leads to extreme outliers for the computation of the average operational intensity. Furthermore, the performance is also higher when using LB.

Having explored the effect of LB in the single-node case, we now test our findings on a multi-node molecular dynamics simulation. Keeping all other simulation parameters constant, a new simulation volume is defined with size $1,200 \times 900 \times 900$ and is run for $10,000$ timesteps. This results in a total of $363,219,300$ particles, run on 16 nodes of HSUper, with two MPI ranks per node and 36 OpenMP threads per rank, for saturation.

The results of the multi-node jobs can be seen in Figure 13, with the left plot showing the number of DP FLOPS performed and the right plot showing the inter-node communication. We see a reversal of the two fastest methods, with kd-tree based LB performing faster than diffusive LB. This behaviour is explained by the reduced influence of the aforementioned surface-area-to-volume ratio effect with fewer MPI ranks dividing the simulation volume. As shown in the right plot, initially the diffusive load balancer only transmits as much data as the default balancer, and slowly converges to the transfer rates of the kd-tree balancer.

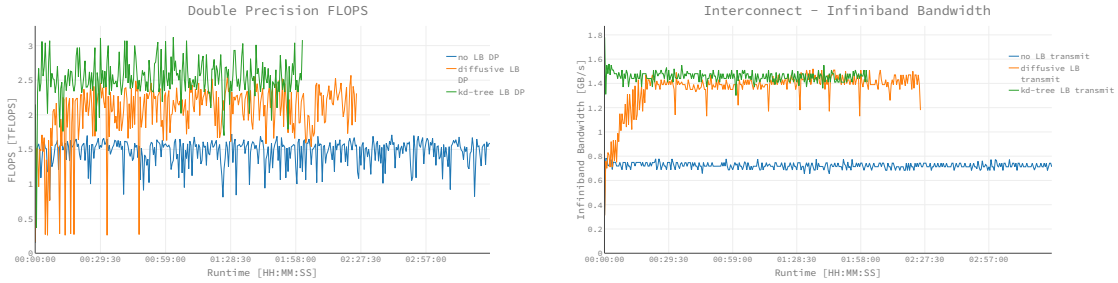

Figure 13: xbat visualisation showing the double precision FLOPS (left) and interconnect over time (right), for each of the LB schemes, for the multi-node experiment, at the job level.

The results confirm our assumptions about the behaviour of the simulation and reveal the effects of the load-balancing approaches on the hardware. Thus, profiling with xbat can provide valuable insights into program behaviour, especially when comparing new application features with existing ones, as demonstrated here.

3.4 lbmpy

Using mesoscopic Lattice Boltzmann Methods (LBM), fluids can be simulated in a discretised domain using two steps: particle interaction (collision step) and particle motion (streaming step). Software packages like lbmpy [HHR23] can leverage parallel architectures like multicore CPUs and GPUs to efficiently accelerate these steps. Two stencils are commonly used for the aforementioned collision step in three-dimensional simulations: D3Q19 and D3Q27, which use 18 and 26 neighbouring cells, respectively. The streaming step must avoid data races when parallelising over the grid cells and can be implemented using a temporary buffer (A-B pattern) or as an in-place operation with alternating steps (A-A pattern) for the Particle Distribution Functions (PDFs) [BMW+09]. We evaluate the four possible combinations using a compressible lid driven cavity flow simulation with the cumulant collision operator [GSPK15], and a lid speed

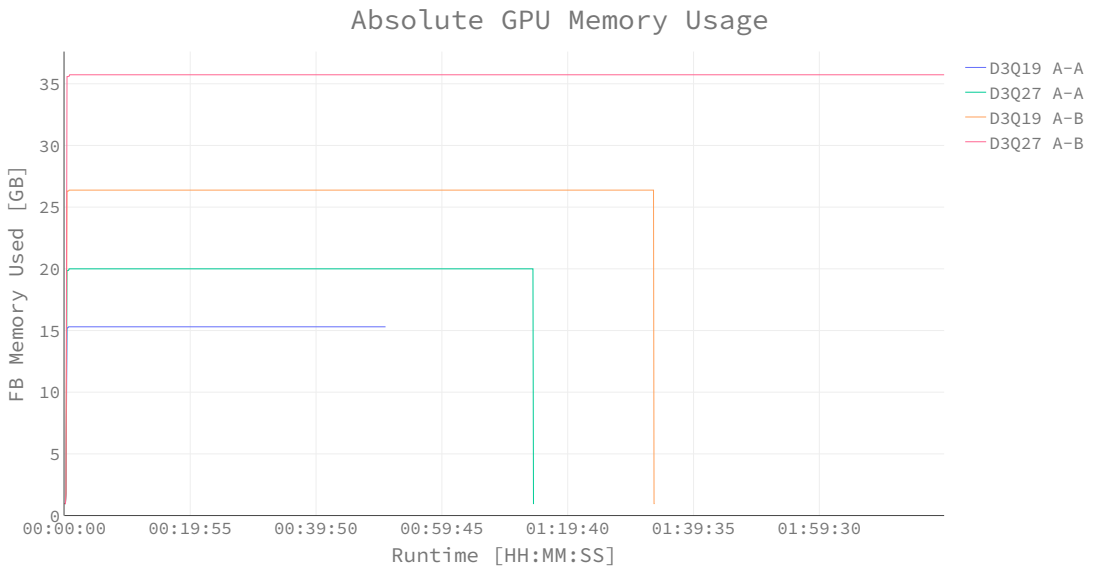

Figure 14: GPU FB memory usage and runtime of different stencil and pattern operation combinations of an lbmpy simulation.

of 0.05 over 1,000 time steps with 10 evenly spaced checkpoints. A value of 1.9 was used for the relaxation rate. The domain volume of 425^3 is chosen to fit within the 40 GB memory of one NVIDIA A100 GPU. Benchmark parameters (cf. Figure 2) are used to set the stencil and streaming pattern in the started jobs through command line parameters of the simulation script. The resulting Frame Buffer (FB) memory usage on the GPU usage of the jobs is shown in Figure 14. A quantitative comparison is given in Table 4. The FB memory saved by switching from a D3Q27 to a D3Q19 stencil is almost proportional to the corresponding ratio of PDFs per cell, while the runtime savings are more than proportional. Switching from the A-B to the A-A streaming pattern (thereby eliminating the need for a second buffer) does not quite halve the FB memory usage. These deviation results from the additional density and velocity fields, independent of stencil and streaming pattern. Only the storage of the PDFs is halved. Overall, a combination of the simpler stencil and the in-place streaming pattern, reduces the FB memory usage by over 57 % and the runtime even by over 63 %.

This use case demonstrates how xbat can be leveraged to trivially execute parameter studies for performance engineering of GPU-codes like lbmpy. Aside from absolute and relative memory usage, other basic GPU metrics, such as the GPU utilisation, device power draw, clock rate, and P-states are implemented. To provide additional GPU/NVLink metrics, the GPU measurements may be trivially extended by calling the corresponding functions in the `nvml.h` header of the NVIDIA Management Library.

Configuration	Mean GPU FB [GB]		Runtime [s]	
D3Q27 A-B	35.64		8,360	
D3Q19 A-B	26.23	-26.4 %	5,610	-32.9 %
D3Q27 A-A	19.87	-44.2 %	4,465	-46.6 %
D3Q19 A-A	15.21	-57.3 %	3,055	-63.5 %

Table 4: Impact of stencils and streaming methods on the GPU FB memory and simulation runtime in lbmpy.

4 Conclusion and Outlook

xbat helps to investigate the behaviour of software to optimise the hardware usage. CI workflows can be enhanced by xbat to automate benchmarking and to obtain feedback about performance changes between different software versions.[6] We have highlighted distinguishing features of xbat as well as a newly developed companion CLI that extends the use of xbat beyond the web-based GUI. Using four different, representative use cases, we demonstrated how xbat can be applied by research software engineers (RSEs) and researchers working in HPC.

The future development of xbat will focus on deepening the analytical capabilities available to users. A central goal is the introduction of a marker API that allows users to instrument their code with region annotations, relating distinct execution phases to the collected performance data and reflecting them in interactive graphs. Sensor abstractions for all supported metrics are implemented as C++ classes. Providing users with an additional interface for collecting custom job metrics at runtime (similar to the marker API and JUBE or ReFrame) is being considered. Additionally, extending our monitoring capabilities is an ongoing process, and more detailed GPU monitoring, including support for NVLink metrics, is already in development.

We are currently investigating automated bottleneck and anomaly detection, exploring a combination of heuristics and machine learning (ML) models to identify performance bottlenecks and provide actionable optimisation recommendations. If successful, this would further reduce the expertise required to interpret complex performance data.

Beyond these two pillars, development efforts will also target improved statistics and analysis features, an extended CLI for deeper workflow integration, and assistance in building hardware-aware performance models of user applications.

Acknowledgements: The HPC cluster HSUper has been provided by the projects hpc.bw and MaST. The projects hpc.bw and MaST are funded by dtec.bw – Digitalization and Technology Research Centre of the Bundeswehr which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU. Development of xbat was partially funded by the EuroHPC project MICROCARD.

Data availability: xbat configuration files and results are available online as part of the demo instance at https://demo.xbat.dev. Ancillary files are available on Zenodo [TSL+26a].

[6] This is actively used within the Microcard project as explained in the documentation: https://xbat.dev/docs/demo

Declaration on the use of AI: No generative artificial intelligence was used to produce the results and figures of this publication. Standard writing tools were used to assist with grammar, spelling, and language editing. GitHub Copilot code review and Claude were used in a limited capacity for the development process of xbat to provide feedback or suggestions.

Bibliography

[ABF+10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22(6):685–701, 2010.
doi:https://doi.org/10.1002/cpe.1553

[BEA+18] F. Badosa, A. Espinosa, C. Acevedo, G. Vera, A. Ripoll. A History-Based Resource Manager for Genome Analysis Workflows Applications on Clusters with Heterogeneous Nodes. *International Journal of Parallel Programming* 47(2):317–342, Sept. 2018.
doi:10.1007/s10766-018-0600-z

[BMW+09] P. Bailey, J. Myre, S. Walsh, D. Lilja, M. Saar. Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors. In *2009 International Conference on Parallel Processing*. P. 550–557. IEEE, Sept. 2009.
doi:10.1109/icpp.2009.38

[BWS+24] T. Breuer, J. Wellmann, F. Souza Mendes Guimarães, C. Himmels, S. Luehrs. JUBE. 2024.
doi:10.5281/ZENODO.11394333

[DBC+05] L. Djoudi, D. Barthou, P. Carribault, C. Lemuet, J.-T. Acquaviva, W. Jalby. MAQAO: Modular Assembler Quality Analyzer and Optimizer for Itanium 2. In *The 4th Workshop on EPIC architectures and compiler technology, San Jose*. Volume 200(5). 2005.

[DL11] J. Dongarra, P. Luszczek. LINPACK Benchmark. In Padua (ed.), *Encyclopedia of Parallel Computing*. Pp. 1033–1036. Springer US, 2011.
doi:10.1007/978-0-387-09766-4_155

[EGA+19] J. Eitzinger, T. Gruber, A. Afzal, T. Zeiser, G. Wellein. ClusterCockpit – A web application for job-specific performance monitoring. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. Pp. 1–7. IEEE, Sept. 2019.
doi:10.1109/cluster.2019.8891017

[For26] J. Forcier. Fabric – Pythonic remote execution. https://www.fabfile.org, 2026. Accessed: 2026-05-01.

[FV03] A. Fink, S. Voß. Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research* 151(2):400–414, 2003.

[GSPK15] M. Geier, M. Schönherr, A. Pasquali, M. Krafczyk. The cumulant lattice Boltzmann equation in three dimensions: Theory and validation. *Computers & Mathematics with Applications* 70(4):507–547, Aug. 2015.
doi:10.1016/j.camwa.2015.05.001

[GST+19] F. A. Gratl, S. Seckler, N. Tchipev, H.-J. Bungartz, P. Neumann. AutoPas: Auto-Tuning for Particle Simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Pp. 748–757. May 2019.
doi:10.1109/IPDPSW.2019.00125

[HHR23] F. Hennig, M. Holzer, U. Rüde. Advanced Automatic Code Generation for Multiple Relaxation-Time Lattice Boltzmann Methods. *SIAM Journal on Scientific Computing* 45(4):C233–C254, Aug. 2023.
doi:10.1137/22m1531348

[IVT26] Intel VTune Profiler. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html, 2026. Accessed: 2026-05-08.

[KKD+23] A. Khelifati, M. Khayati, A. Dignös, D. Difallah, P. Cudré-Mauroux. TSM-Bench: Benchmarking Time Series Database Systems for Monitoring Applications. *Proceedings of the VLDB Endowment* 16(11):3363–3376, July 2023.
doi:10.14778/3611479.3611532

[KMR+20] V. Karakasis, T. Manitaras, V. H. Rusu, R. Sarmiento-Pérez, C. Bignamini, M. Kraushaar, A. Jocksch, S. Omlin, G. Peretti-Pezzi, J. P. S. C. Augusto, B. Friesen, Y. He, L. Gerhardt, B. Cook, Z.-Q. You, S. Khuvis, K. Tomko. Enabling Continuous Testing of HPC Systems Using ReFrame. In Juckeland and Chandrasekaran (eds.), *Tools and Techniques for High Performance Computing*. Communications in Computer and Information Science 1190, pp. 49–68. Springer International Publishing, Cham, Switzerland, Mar. 2020.
doi:10.1007/978-3-030-44728-1_3

[LDF+25] W. G. Leinen, A. Das Sharma, A. Fink, N. Tippmann, A. Auweter, P. Neumann. xbat – An Easy-to-Use and Universally Applicable Benchmarking Automation Tool for HPC Software within the Project hpc.bw (dtec.bw). Conference poster, Universitätsbibliothek der HSU/UniBw H, 2025. ISC High Performance 2025, Hamburg, Germany, 10–13 June 2025.
doi:10.24405/20152

[LFN] W. Leinen, A. Fink, P. Neumann. Co-Scheduling and Oversubscription of MIP Solver Workloads for Throughput and Energy Efficiency on Multi-core Hardware. In preparation.

[LFN25] W. G. Leinen, A. Fink, P. Neumann. Balancing Energy and Performance: Efficient Allocation of Solver Jobs on High-Performance Computing Systems. Conference slides, Universitätsbibliothek der HSU/UniBw H, 2025. 34th European Conference on Operational Research (EURO 2025), Leeds, United Kingdom, June 22–25 2025. doi:10.24405/20237

[LFN26] W. Leinen, A. Fink, P. Neumann. Computational Insights into Solving Optimization Instance Sets with MIP Solvers on Multi-Core Hardware. Conference slides, Universitätsbibliothek der HSU/UniBw H, 2026. 24th Conference of the International Federation of Operational Research Societies (IFORS 2026), Vienna, Austria, July 12–17 2026.
doi:10.24405/24097

[Lin24] Linux perf wiki Contributors. perf: Linux profiling with performance counters. https://perfwiki.github.io/main/, 2024. Accessed: 2026-05-10.

[McC95] J. D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[NBB+14] C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C. W. Glass, H. Hasse, J. Vrabec, M. Horsch. ls1 mardyn: The Massively Parallel Molecular Dynamics Code for Large Systems. *Journal of Chemical Theory and Computation* 10(10):4455–4464, 2014.
doi:10.1021/ct500169q

[NDK+22] P. Neumann, J. A. Duffek, J. Kleinschmidt, W. G. Leinen, M. Breuer, S. Schmidt-Lauff, A. Fink, M. Mayr, M. Firmbach, A. Popp, A. Auweter. hpc.bw: A Supercomputer with Competence Platform for the Universities of the Federal Armed Forces. In *dtec.bw-Beiträge der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg: Forschungsaktivitäten im Zentrum für Digitalisierungs- und Technologieforschung der Bundeswehr dtec.bw: Band 1 – 2022*. Pp. 305–310. Universitätsbibliothek der HSU/UniBw H, 2022.
doi:10.24405/14569

[NDV+22] P. Neumann, A. Das Sharma, L. Viot, T. Trummler, A. Doehring, M. Son, T. Sander, M. Pfitzner, L. Zigan, M. Klein, F. Gratl, S. Newcome, H.-J. Bungartz, I. Nitzke, S. Homes, J. Vrabec, R. Stierle, J. Gross, A. Auweter, N. Tippmann. MaST: Scale-Bridging Exploration of Transcritical Fluid Systems. In *dtec.bw-Beiträge der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg: Forschungsaktivitäten im Zentrum für Digitalisierungs- und Technologieforschung der Bundeswehr dtec.bw: Band 1 – 2022*. Pp. 101–106. Universitätsbibliothek der HSU/UniBw H, 2022.
doi:10.24405/14537

[NNC26] NVIDIA Nsight Compute. https://developer.nvidia.com/nsight-compute, 2026. Accessed: 2026-05-08.

[NNS26] NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems, 2026. Accessed: 2026-05-08.

[PDD+24] H. Preuß, G. De Nayer, A. Das Sharma, P. Jarmatz, W. G. Leinen, V. Jafari, R. Horn, M. Breuer, A. Fink, P. Neumann. hpc.bw benchmark report 2022–2024. Technical report, Universitätsbibliothek der HSU/UniBw H, 2024.
doi:10.24405/17134

[Ram25] S. Ramírez. Typer: Build great CLIs. Easy to code. Based on Python type hints. https://typer.tiangolo.com/, 2025. Accessed: 2026-05-01.

[SM06] S. S. Shende, A. D. Malony. The TAU Parallel Performance System. *Int. J. High Perform. Comput. Appl.* 20(2):287–311, May 2006.
doi:10.1177/1094342006064482

[SMK26] S. Shehbaz, M. Mehabadi, K. B. Kent. Evaluating Time Series Databases Across Wide and Narrow Schema Designs for Appliance-Level Residential Energy Monitoring. In *SoutheastCon 2026*. Pp. 1–8. IEEE, Feb. 2026.
doi:10.1109/southeastcon63549.2026.11476662

[TAD+24] N. Tippmann, A. Auweter, A. Das Sharma, W. G. Leinen, P. Neumann, A. Fink. xbat: A Continuous Benchmarking Tool for HPC Software. In *dtec.bw-Beiträge der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg: Forschungsaktivitäten im Zentrum für Digitalisierungs- und Technologieforschung der Bundeswehr dtec.bw: Band 2 – 2024*. Pp. 91–97. Universitätsbibliothek der HSU/UniBw H, 2024.
doi:10.24405/16783

[THW11] J. Treibig, G. Hager, G. Wellein. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. In *Competence in High Performance Computing 2010*. Pp. 165–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
doi:10.1007/978-3-642-24025-6_14

[THW12] J. Treibig, G. Hager, G. Wellein. likwid-bench: An Extensible Microbenchmarking Platform for x86 Multicore Compute Nodes. In *Tools for High Performance Computing 2011*. Pp. 27–36. Springer Berlin Heidelberg, 2012.
doi:10.1007/978-3-642-31476-6_3

[TSL+26a] N. Tippmann, Z. Su, W. Leinen, R. Horn, S. Wanniarachchi, A. Das Sharma, A. Fink, A. Auweter, P. Neumann. Ancillary Data: Easy, Collaborative Benchmarking with the Extended Benchmarking Automation Tool xbat. Ancillary Data, Zenodo, 2026. 6th conference for Research Software Engineering in Germany (deRSE26), Stuttgart, Germany, 3–5 March 2026.
doi:10.5281/zenodo.21725914

[TSL+26b] N. Tippmann, Z. Su, W. Leinen, R. Horn, S. Wanniarachchi, A. Das Sharma, A. Fink, A. Auweter, P. Neumann. Easy, Collaborative Benchmarking with the Extended Benchmarking Automation Tool xbat. Conference slides, Zenodo, 2026. 6th conference for Research Software Engineering in Germany (deRSE26), Stuttgart, Germany, 3–5 March 2026.
doi:10.5281/zenodo.18864871

[WWP09] S. Williams, A. Waterman, D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM* 52(4):65–76, Apr. 2009.
doi:10.1145/1498765.1498785