Computing and Software for Big Science (2021) 5:28
https://doi.org/10.1007/s41781-021-00074-y

ORIGINAL ARTICLE

HEPiX Benchmarking Solution for WLCG Computing Resources

Domenico Giordano[1] · Manfred Alef[2] · Luca Atzori[1] · Jean-Michel Barbet[3] · Olga Datskova[1] · Maria Girone[1] · Christopher Hollowell[5] · Martina Javurkova[4] · Riccardo Maganza[1] · Miguel F. Medeiros[1] · Michele Michelotto[6] · Lorenzo Rinaldi[9] · Andrea Sciabà[1] · Randall J. Sobie[7] · David Southwick[1,8] · Tristan Sullivan[7] · Andrea Valassi[1]

Received: 30 August 2021 / Accepted: 8 November 2021
© The Author(s) 2021

Abstract

The HEPiX Benchmarking Working Group has developed a framework to benchmark the performance of a computational server using the software applications of the High Energy Physics (HEP) community. This framework consists of two main components, named HEP-Workloads and HEPscore. HEP-Workloads is a collection of standalone production applications provided by a number of HEP experiments. HEPscore is designed to run HEP-Workloads and provide an overall measurement that is representative of the computing power of a system. HEPscore is able to measure the performance of systems with different processor architectures and accelerators. The framework is completed by the HEP Benchmark Suite that simplifies the process of executing HEPscore and other benchmarks such as HEP-SPEC06, SPEC CPU 2017, and DB12. This paper describes the motivation, the design choices, and the results achieved by the HEPiX Benchmarking Working group. A perspective on future plans is also presented.

Keywords CPU benchmark · GPU benchmark · High throughput computing · WLCG · LHC computing · HEP experiments · High-Energy Physics · Heterogeneous computing

Introduction

The HEP-SPEC06 (HS06) benchmark [1], based on SPEC CPU 2006 [2], is currently used by the Worldwide LHC Computing Grid (WLCG) [3] community to estimate the performance of a computing server. HS06 is adopted by the WLCG as a performance metric for resource capacity planning, hardware acquisition, pledging of future resources, and usage accounting of the experiments.

HS06 has been used for over a decade, satisfying the WLCG requirements in a landscape that progressively evolved from CPUs with a few cores to multi-cores CPUs. When HS06 was established, the HEP applications shared several commonalities with the SPEC CPU 2006 workloads included in HS06: they were characterized by single-threaded and single-process applications, compiled in 32-bit mode, with a memory footprint of about 1 GB per process. Since then, the HEP-workloads have significantly changed and evidence of scaling deviation with respect to HS06 has been reported [4]. Even if HS06 may continue to be a viable benchmark for evaluating the performance of x86 CPUs, the community will soon require a benchmark to evaluate also

Innovation in HEP software and computing for the challenges of the next decade" Guest editors: Ian Bird, Simone Campana, Graeme A. Stewart S.I. Innovation in HEP software and computing for the challenges of the next decade (invitation only).

✉ Domenico Giordano
domenico.giordano@cern.ch

1 CERN, Geneva, Switzerland

2 KIT, Karlsruhe, Germany

3 Subatech UMR 6457, CNRS-IN2P3, IMT Atlantique, Université de Nantes, Nantes, France

4 University of Massachusetts, Amherst, USA

5 Brookhaven National Laboratory, Upton, USA

6 INFN, Istituto Nazionale di Fisica Nucleare, Padua, Italy

7 Department of Physics and Astronomy, University of Victoria, Victoria, BC, Canada

8 University of Iowa, Iowa, USA

9 Dipartimento di Fisica e Astronomia, Università di Bologna and INFN, Bologna, Italy

Published online: 14 December 2021

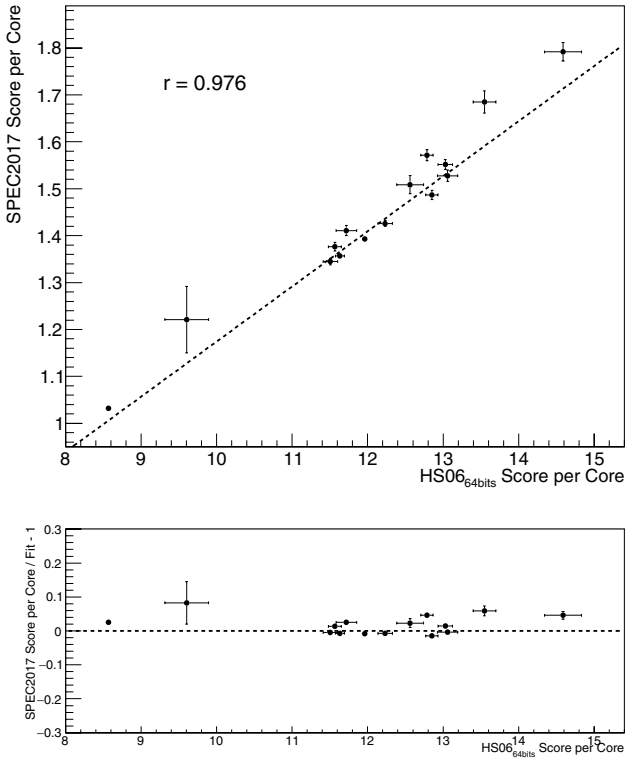

Fig. 1 SPEC CPU 2017 vs HS06$_{64bits}$ scores (top). Each point corresponds to a different CPU model. The scores are normalized to the number of available cores. The correlation coefficient of the two samples is 0.976. The line is a linear fit to the data points where the y-intercept was fixed to zero. The normalized residuals with respect to the fit values are also reported (bottom). The data used in this plot were obtained from Ref. [7]

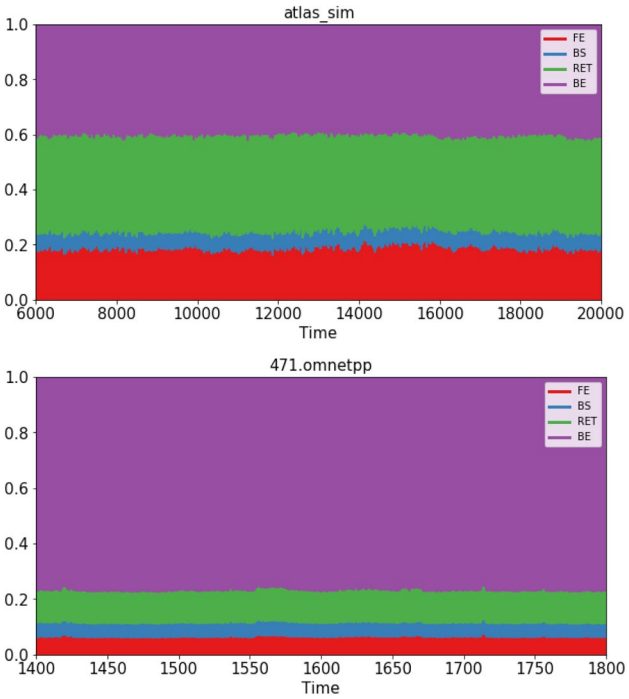

Fig. 2 Percentage of the CPU cycles spent in front-end stalls (FE), back-end stalls (BE), in bad speculation (BS), or in completed executions retired from the execution stack (RET), as a function of the application's running time, for the ATLAS simulation (top) and the HS06 omnetpp application (bottom). The initialization and finalization phase of the execution have been excluded. This plot was obtained from Ref. [11]

emerging hardware and software technologies to be adopted in the coming years. From the hardware point of view, CPU architectures have evolved and high-throughput computing has become heterogeneous. Hardware accelerators such as GPUs and FPGAs, as well as non-x86 CPU architectures as ARM, are being adopted. An HEP benchmark should provide a profiling of all these architectures, remaining representative of the HEP applications that will also run on the same architectures. From the application point of view, the HEP software is being redesigned to take advantage of the new architectures, and to exploit multi-threading, vectorization, and parallel computing. Although some of these innovations are not yet adopted in the production software of the experiments, an HEP benchmark should be designed to include them. Last but not least, envisaging a transition to an alternative benchmark is particularly important, because the support for SPEC CPU 2006 ended in 2018; therefore, any further development is not possible.

The HEPiX Benchmarking Working Group [5] has been tasked by WLCG to find a replacement for HS06 that meets the needs of the community. The first alternative considered was SPEC CPU 2017 [6]. It contains a larger application set than the 2006 version, at the cost of a longer compiler time. SPEC CPU 2017 was found highly correlated with HS06 when executing the C++ applications included in its suite (Fig. 1) [7]. Therefore, it can be a natural replacement of HS06, coming with the same support from the SPEC organization and a simple conversion factor to link with HS06.

Note that the HS06 score reported in Fig. 1 is determined when compiling the applications with the 64-bits compiler flag (HS06$_{64bits}$). This choice, common to other measurements done in this report, is a consequence of a number of studies [8, 9] showing how the official HS06 configuration, built with 32-bits compiled flag, reports a score that is systematically lower than the 64-bits version by a factor of 10–20%. Although this discrepancy was known, the official HS06 never changed to the 64-bits flag when the HEP software moved to 64 bits. The choice of keeping the 32-bits compiler flag was meant to avoid a redefinition of the accounting values that the move to 64 bits would have implied.

Replacing HS06 with SPEC CPU 2017 would not solve a conceptual dichotomy that has caused criticism also for HS06: both suites do not include any workloads of the HEP community and they uses the CPU resources differently. A number of studies [10, 11] have highlighted the

different CPU usage patterns of HS06, SPEC CPU 2017, and HEP applications as measured with CPU performance counters. Comparisons of the application execution cycles highlighted deviations of the order of 60% between the the HEP applications and HS06/SPEC CPU 2017, whereas the HEP applications were consistent with each other to within 20%. As an example, Fig. 2 shows the percentage of CPU cycles spent in each of the four categories of the execution stack for one LHC experiment application and one HS06 application.

As a result, the Benchmarking Working Group has explored a benchmark solution based on HEP applications, that would be by construction correlated with the workloads of the HEP community. Even though HEP applications are complex software packages, and using them as benchmark applications has been challenging in the past, new and established IT practices make this new approach feasible. For example, OS-level virtualization enables the encapsulation of the experiment software stack, data and configuration in a standalone container image without any additional dependency or requirement for external network connectivity.

The concept of a HEP-specific benchmark was first proposed at the WLCG Workshop in Manchester (2017) [12]. This led to the creation of the HEP Benchmarks project, and the development of a software framework to benchmark the performance of computing servers using HEP applications [13]. A key requirement of the framework is that it must be easy to install and use, and there must be a plan for long-term support. The project is maintained on the GitLab infrastructure at CERN, and comprises several repositories that include benchmarks, orchestration, and analysis packages [14]. The WLCG would like an open-source solution with a free license that would guarantee that the framework's components have copyright protection, while still allowing for its inclusion in derivative work. Therefore, all the components are released under GNU GPL v3, and similar license conditions are verified for the selected application software from the experiments. The benchmark framework is free of charge, avoiding the need to acquire a license, in contrast to HS06 and SPEC CPU 2017 that are proprietary packages requiring licenses at each location.

This paper describes the current state of the HEP Benchmarks project. Section "HEP-Workloads" describes HEP-Workloads, which is the collection of applications provided by the HEP experiments and used to create the HEPScore benchmark that is presented in Sec. "HEPscore". The HEP Benchmark Suite, used to run benchmark applications and to collect, store, and process benchmark data, is described in Sec. "HEP Benchmark Suite". Section "Results Using HEPscore$_\beta$" details the properties of a proof-of-concept benchmark (HEPscore$_\beta$), defined from a set of current HEP applications as demonstrator of the proposed approach. Section "Outlook" provides an overview of the future plans of the Benchmarking Working Group.

HEP-Workloads

HEP-Workloads is a collection of applications provided by several experiments. The repository [14] contains the code and infrastructure, both common and workload-specific, to build a standalone HEP-Workloads container image for each application.

Each HEP-Workloads container encapsulates the software and input data needed to run the application of a specific experiment. The software of the HEP experiments is typically stored in the CVMFS file system [15]. CVMFS is a remote file system, whereas the HEP-workload containers must contain all the software to avoid any dependency on remote services. The benchmark applications typically need a subset of the software stack of an experiment. As a result, a procedure has been developed, based on the CVMFS Trace and Export utilities, to export the application software from CVMFS to a local folder inside a container. The procedure performs a first run of the application with access to the CVMFS mount point to trace the accessed software files. Subsequently, the Export utility copies the traced files to a local archive that can be included in the HEP-Workloads container image. The CVMFS Trace and Export utilities simplifies the building of the HEP-Workloads containers, avoiding the installation of large software packages and their dependencies. Note that the framework developed in HEP-Workloads still includes as an option the installation of software via package management systems.

The HEP-Workloads containers are built by the Benchmarking Working Group with the support of the software experts of the experiments. The build procedure is implemented in the HEP-Workloads GitLab repository and leverages the GitLab continuous integration framework. The experts need to prepare an orchestrator script, which sets the runtime environment accessing CVMFS and runs the experiment's application. Once the application has finished, the orchestrator parses the output logs to determine the event throughput, which is used as the benchmark score.

Each HEP-Workloads container includes a configuration file for the application and, in some cases, one or more input files with event data or conditions data needed for processing the events. The number of events to be processed is configurable, which allows one to adjust the duration of the execution. The size of the input data file depends on the size of a single event and on the maximum number of events that can be processed.

HEP-Workloads currently includes a preliminary set of applications that generate, simulate, digitize, and reconstruct

Table 1 Preliminary list of the HEP-Workloads

HEP-workload	License	Threads	Size [GB]	Events/threads	Runtime [min]	*wl-score* [Events/s]
ATLAS GEN	Apache v2 [23]	1	0.5	200	14	384
ATLAS SIM	Apache v2 [23]	1	1.9	10	90	0.064
Belle II GEN-SIM-RECO	GNU LGPL v3 [24]	1	0.9	50	8	5.44
CMS GEN-SIM	Apache v2 [25]	4	2.0	20	15	0.73
CMS DIGI	Apache v2 [25]	4	4.0	50	9	3.58
CMS RECO	Apache v2 [25]	4	2.9	50	14	2.20
LHCb GEN-SIM	GNU GPL v3 [26]	1	0.7	5	33	90.3

The ATLAS, CMS, and LHCb workloads use their *Run2* software. The table includes the software license, number of threads, the size of the container image, the number of events processed per thread, the runtime, and the *wl-score*. The runtime and *wl-score* were measured on a reference server: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz, with hyper-threading enabled

HEP events [16] from the ATLAS [17], CMS [18], and LHCb [19] experiments at the CERN Laboratory in Geneva and the Belle II experiment [20] at the KEK Laboratory in Tsukuba, Japan (Table 1). The current set of benchmark workloads from the LHC experiments use the older *Run2* software for the collision data collected up to 2018. Workloads using the newer *Run3* software, for data to be collected in 2022 and beyond, will be added once available. The Belle II benchmark workload uses the latest software.

The size of the HEP-Workloads containers ranges between 1 and 4 GB (including the software and input data files). The images are hosted and distributed in the CERN GitLab container registry.

An orchestrator script within each HEP-Workloads container manages the configuration of the environment, the start of the application, the error handling, and the extraction of the results. The orchestrator produces a benchmark report in a JSON format, as shown below, that includes the workload scores (*wl-scores*) and the status flag of the run (*log*).

```
{...,
 "report": {
           "wl-scores": {"gen-sim": 18.8476,
                         "gen": 95.9261,
                         "sim" : 24.2931 },
            "log": "ok"},
...
}
```

Currently, all HEP-workloads are event-based applications, and *wl-scores* is the total number of processed events per second.

One can configure the orchestrator to simultaneously run multiple, independent copies of the same application. The default running-mode, named the *full-load* configuration, exploits all the available cores of a server and ensures that resources are not over-committed. The number of copies of the application is determined by the number of cores in the server and the number of threads/processes per application copy.

The orchestrator and the HEP application run unprivileged. This feature makes it possible to run the HEP-Workloads on HPC sites that, for security reasons, are averse to any elevated permissions or processes.

To create a reliable benchmark to replace HS06, it is critical that the results are reproducible for a given configuration file and input file. This requires that the same event sequence must be processed in repeated measurements. This is enforced by fixing in the application's configuration the parameters that affect the sequence, such as the type of physics event to be simulated, the random number seed to be used, and the number of events per thread to be processed.

Figure 3 shows histograms of the events processed per second of each of the HEP-Workloads listed in Table 1 run on a single server. Each histogram is fitted with a Gaussian distribution and is shown as a solid line in the figure. For each of the HEP-Workloads, the ratio of the standard deviation to the mean value, obtained from the fit, is less than 1% of the fitted mean values, demonstrating the high level of reproducibility of these measurements. Similar results are obtained for the other servers studied in this work.

HEPscore

HEPscore is the utility that orchestrates the execution of multiple HEP-Workloads containers and determines the benchmark score of a given compute server [14]. Similar to HS06, each HEP-workload i is executed multiple times (three by default) on a given server m and the application score, $a_i(m)$, for the ith workload is the median value of the *wl-scores* in successful runs to minimize the impact of fluctuations.

The individual application scores $a_i(m)$ are normalized to the scores obtained on the reference server $a_i(m_R)$. The

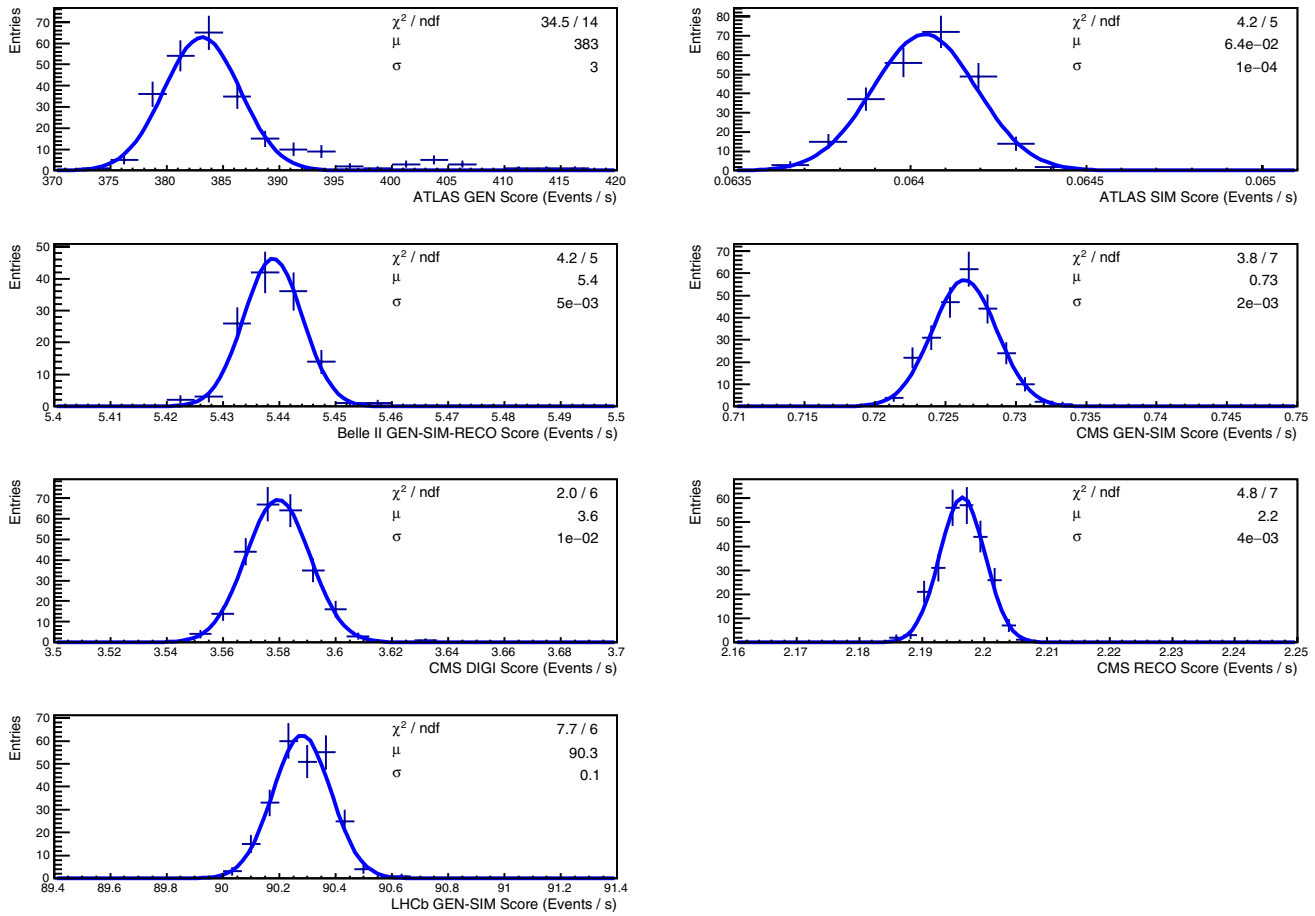

Fig. 3 Histograms of the measurements of the events processed per second of the seven HEP-Workloads on the reference server Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz, with hyper-threading enabled. Note that each entry is the median of three sequential measurements of the workload

last column of Table 1 shows the *wl-score* obtained on the reference server m_R, given in events per second. The normalized scores $a_i^{\text{norm}}(m) = a_i(m)/a_i(m_R)$ are combined into a single benchmark value via the weighted geometric mean. The weights w_i allow one to adjust the relative contributions of the individual applications. This approach is similar to the method used to compute the HS06 score; however, HS06 does not allow one to apply weights to the individual applications.

The HEPscore value for a set of n HEP-Workloads on a server m relative to the reference server m_R is given by

$$\text{HEPscore}(m, m_R; \mathbf{p}) = \alpha \prod_{i=1}^{n} \left(\frac{a_i(m;\mathbf{p})}{a_i(m_R;\mathbf{p})} \right)^{\frac{w_i}{\sum_{j=1}^{n} w_j}}, \quad (1)$$

where n is the number of workloads, $\mathbf{p}$ are the configuration parameters of HEPscore and HEP-Workloads, and α is a normalization factor that is used to rescale the value to a convenient numerical range. The method remains valid across different system architectures and accelerators, as long as the HEP-Workloads have been built for these systems.

HEPscore executes the HEP-Workloads container images via Docker [21] or Singularity [22] container engines. HEPscore provides an option to force Singularity user namespace-based container execution, which is needed if HEPscore is executed within a Singularity container (nested container environments), like it would be in pilot jobs of many WLCG sites. HEPscore also includes an option to clear the container image cache, as the container images and working directories can easily expand to many tens of gigabytes and saturate the working space of the server under study, causing benchmark failures.

HEPscore is agnostic to the internal details of workloads. There are few constraints that the design imposes on the workloads. From the execution perspective, the

workloads must be containerized, have as entrypoint an orchestrator script (see Sec. "HEP-Workloads"), and accept command line arguments to be configured. From the reporting point of view, the workloads must produce a report output file in JSON format, and include in it the workload scores and the success or failure flag of the run.

HEPscore is configured with a YAML file that includes the list of HEP-Workloads containers and their respective configuration parameters. HEPscore creates JSON or YAML output reports containing the overall score as well as the individual results of each HEP-Workloads containers. Runtime parameters, execution time, and system metadata information, such as OS kernel version and container platform, are also included in the report.

HEPscore can run different benchmarks by modifying the configuration parameters **p**, and including a different sets of HEP-workloads. Each parameter set **p** is reported in the benchmark result, via a unique identifier, to avoid misidentification.

In addition, conventional names are assigned to the parameter set **p**. For example, the HEPscore software is being released with a proof-of-concept configuration named HEPscore$_\beta$, based on the set of workloads in Table 1 with the exclusion of the ATLAS SIM workload. [1]

This approach is common also for SPEC CPU 2006: HS06 is the all_cpp.bset [1] set of applications included in that suite. This is only one of the many possible sets of applications that can run using SPEC CPU2006.

A WLCG Task Force [27], consisting of members from the global HEP community, was formed in 2020 to evaluate the feasibility of replacing HS06 benchmark with HEPscore benchmark for the WLCG computing resources. The role of the Task Force is to identify the composition **p** for the new benchmark, that is temporarily named HEPscore*X*. The composition of the new benchmark will be defined using the latest HEP applications, that will be used in production in the coming years. The work of the Task Force is still in a preliminary phase and is beyond the scope of this paper.

HEP Benchmark Suite

The HEP Benchmark Suite is an orchestrator that can run the HEPScore utility described in the previous sections, and also non-HEP benchmarks such as HS06, SPEC2017, and DB12 [28]. The Suite is a lightweight package written in Python 3, relying only on a few dependencies to ensure its ease of portability and packaging.

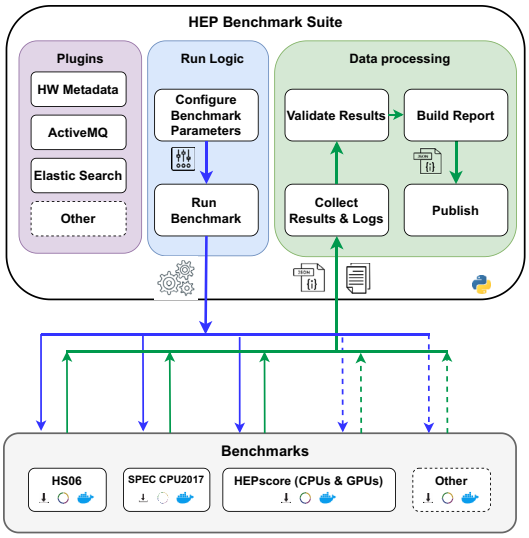

Fig. 4 HEP Benchmark Suite design. The figure depicts the Suite main functional blocks together with the decoupled benchmarks

The design of the Suite is presented in Fig. 4, and has three main components: Plugins, Run Logic, and Data Processing blocks. The Plugins block contains all add-on features such as the hardware metadata and communication interfaces with external services. The Run Logic block executes benchmarks with options specified in a single configuration file, where parameters such as a list of the benchmarks, the number of cores, and user tags are specified. The Data Processing block is responsible for collecting and processing the benchmark data for the final report.

The benchmarks, as shown in Fig. 4, are decoupled from the Suite. The list of benchmarks can be modified or new ones added without impacting the Suite. To run HS06 and SPEC CPU 2017 with the Suite, one must use the container image [29] developed by the Benchmarking Working Group to orchestrate the SPEC executions and ensures standardization of compilation flags and reporting. This container image does not distribute HS06 and SPEC2017 due to license constraints and users will need to provide access to a SPEC installation and license.

All the benchmark results, together with their running conditions, are stored in a JSON structure (Fig. 5). The host metadata comprise all the information to uniquely identify a host such as the user-defined tags, and software and hardware metadata. The list of user defined tags is the only JSON object that can be modified by the user to add extra information and enrich metadata. All the other JSON objects are automatically populated by the Suite. The modular JSON structure enables future schema expansion; therefore, a dedicated report field is used for the schema versioning.

[1] ATLAS SIM workload is excluded because of its long running time, that would make the HEPscore$_\beta$ execution extremely long, without a direct benefit for the studies that will be described in the following paragraphs.

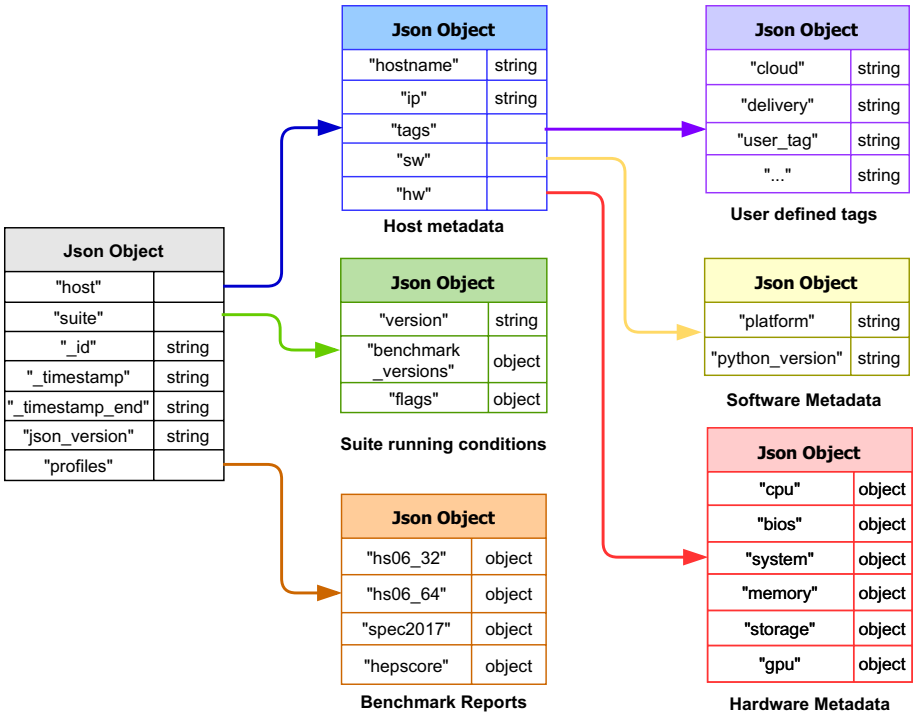

Fig. 5 HEP Benchmark Suite JSON metadata. This metadata structure is modular to allow future expansion

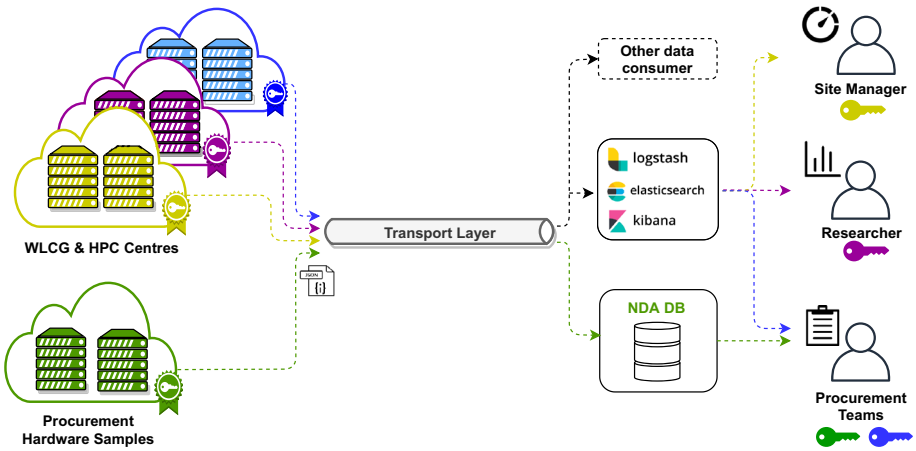

Fig. 6 Flow of the report produced by the HEP Benchmark Suite run. The report can be optionally published to a transport layer where it can be digested by several consumers

The report file is saved locally at the end of benchmark completion. The Suite includes the option of publishing the results into a remote ActiveMQ [30] message queue, from where multiple clients, with different levels of access privileges, can consume the injected data (Fig. 6). For example, site managers can benchmark their clusters and later inspect these results with the assistance of visualization frameworks. Additionally, having the benchmark results on these frameworks allows researchers to easily integrate them in their data analytic solutions.

Results Using HEPscore$_\beta$

This section describes the use of the HEP Benchmark Suite with the HS06$_{64bits}$ and HEPscore$_\beta$ benchmarks. HEPscore$_\beta$ is not the future HEP benchmark as it is based on older software of the LHC experiments and not all the experiments are included. Furthermore, the applications used in the measurements are equally weighted giving a higher significance to the CMS software with its three distinct applications. However, running the Suite to measure HS06 and HEPscore$_\beta$ provides valuable feedback on the functionality of the infrastructure and information on the usability of the HEPscore as a benchmark.

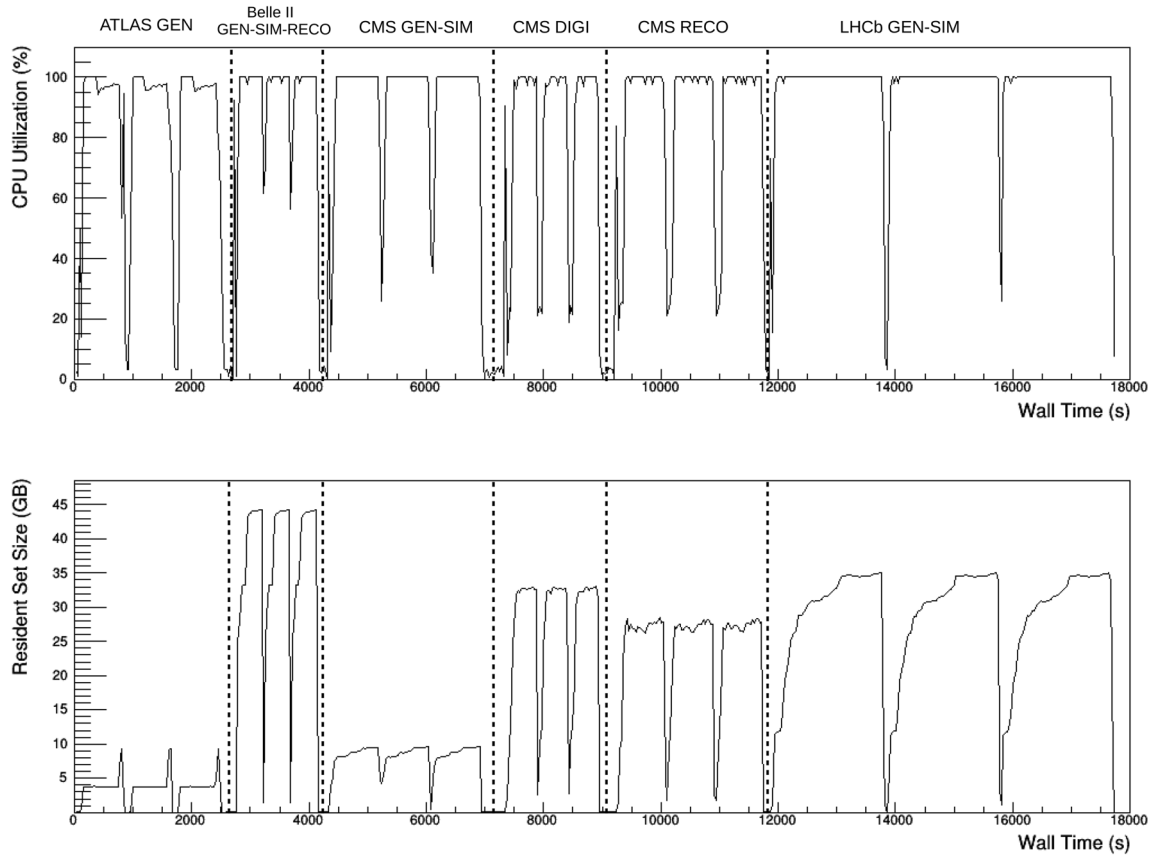

Fig. 7 CPU (top) and memory (bottom) utilization on the reference server measured during an HEPscore$_\beta$ execution. The results show six different regions corresponding to the benchmark applications. Within each region, there are sections associated with the three runs of the application. The measurement was performed using the tool Prmon [31]

The benchmarks were run on servers at a number of WLCG and HPC sites used by the HEP community to evaluate a wide range of CPU models. Fifteen different x86 Intel and AMD CPU models have been studied including newly released and older models with physical cores ranging from 16 to 128. One server, equipped with a dual Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz, was defined to be a reference server that could be used to normalize the benchmark measurements.

The servers were either full bare-metal servers or large virtual machines. The virtual machines were sized and pinned to a whole single CPU socket, to be representative of the CPU performance, modulo a small overhead due to the virtualization layer. The servers were configured with hyper-threading either enabled or disabled; for some CPU models, there are results for both cases.

The HEP Benchmark Suite orchestrated the execution of the HS06$_{64\text{bits}}$ and HEPscore$_\beta$ benchmarks. The Suite was configured to use the *full-load* running-mode, so that all the available cores were utilized. The CPU (top) and memory (bottom) utilization during the execution of HEPscore$_\beta$ on the reference server is shown in Fig. 7. The vertical lines indicate the separation between the different HEPscore$_\beta$ applications that are executed sequentially. Within each of the six sections, there are three runs of the benchmark. The results show that the CPUs are fully exploited.

A comparison of the relative deviations of HEPscore$_\beta$ and HS06$_{64\text{bits}}$ benchmarks, normalized to the reference server, is shown in Fig. 8 (top) for multiple CPU models. Servers configured with HT enabled or disabled are shown as open and full circles, respectively. The different CPU models are identified by their HS06$_{64\text{bits}}$ score on the x-axis. Older CPU models have lower HS06$_{64\text{bits}}$ score, mainly because they have a lower number of available cores. Newer CPU models populate the rightmost region of the plots. Deviations up to 20% between the HS06$_{64\text{bits}}$ and HEPscore$_\beta$ are observed. This shows that, with respect to HS06, the HEPscore approach may provide a more accurate estimation of the computing power available to typical HEP applications.

The individual HEP applications composing HEPscore$_\beta$ are also studied in Fig. 8 (bottom). The relative deviations of their score from the HS06$_{64\text{bits}}$ value can be as large as 40%. Note that the compared scores are normalized to the reference server; therefore, by construction, the relative

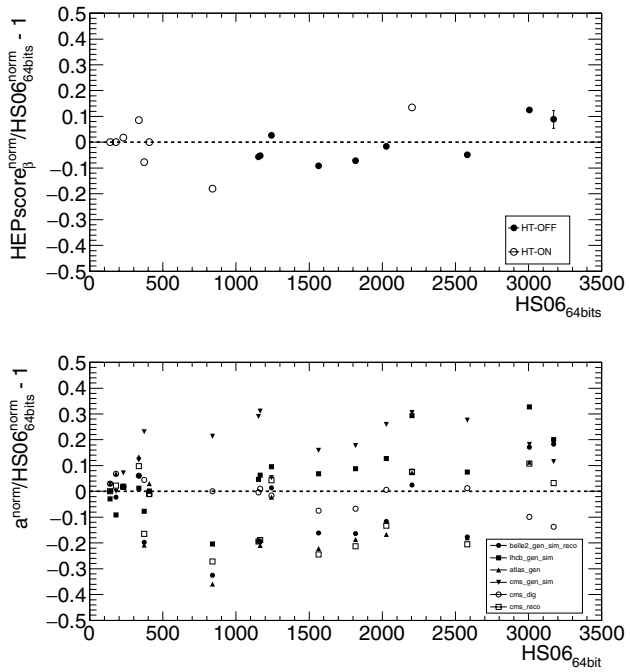

Fig. 8 Relative deviation of the HEPscore$_\beta$ and HS06$_{64bits}$ scores measured on multiple servers and normalized to the score of the reference server (top). Each CPU model is identified on the *x*-axis by the measured HS06$_{64bits}$ score. The normalized score deviations from HS06$_{64bits}$ for each application composing HEPscore$_\beta$ are also reported (bottom). Each marker represents a given CPU model, with HT enabled (open circles) or disabled (full circles)

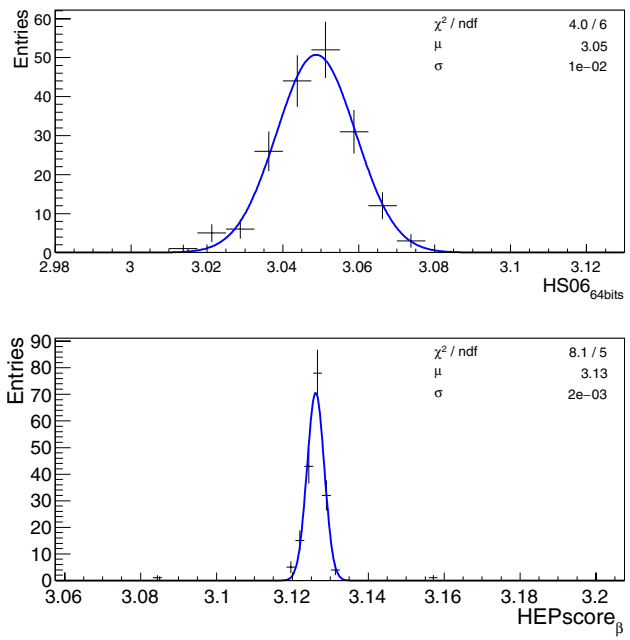

Fig. 9 Distribution of HS06$_{64bits}$ (top) and HEPscore$_\beta$ (bottom) scores of 180 identical servers normalized to the score of the reference server

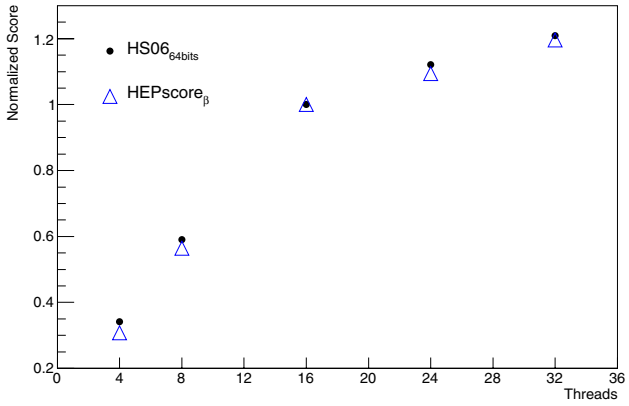

Fig. 10 HS06$_{64bits}$ (points) and HEPscore$_\beta$ (open triangles) measurements on the reference server when only a subset of the hardware threads were loaded by the benchmark application. The values have been normalized to the measurement using 16 threads, which corresponds to the number of physical cores of the server

deviations are equal to zero for this server. The deviations of the individual HEP applications are smaller for the oldest servers, and grow for the newest servers, as well as their relative spread grows.

One of the entries in Fig. 8 is the average of measurements performed on 180 identical servers, where the benchmarks are only run once on each server. The distributions of these HS06$_{64bits}$ and HEPscore$_\beta$ measurements show that the standard deviation of the distribution of HS06$_{64bits}$ and HEPscore$_\beta$ relative to the mean value is approximately 0.34% and 0.06%, respectively (Fig. 9). The result demonstrates that HEPscore$_\beta$ is reproducible in measurements across similar servers with a resolution better than HS06$_{64bits}$.

The performance of the reference server with HT enabled when only a subset of the 32 logical cores were loaded by the benchmark application is shown in Fig. 10. The values for HS06$_{64bits}$ and HEPscore$_\beta$ have been normalized to the measurement using 16 threads, which corresponds to the number of physical cores of the server. The scaling trend is similar for both benchmarks and verifies the benefit of enabling hyper-threading where there is a 20% increase in throughput using 32 threads.

The results show that the HEP Benchmark Suite is able to orchestrate the running of the benchmarks over a variety of servers at different sites under different and often restrictive conditions. The HEPscore$_\beta$ results indicate also that a configuration of HEPscore may yield a potential replacement for HS06.

Outlook

The HEPiX Benchmarking Working Group is aware of the increasing adoption of heterogeneous systems by the HEP experiments and is developing a single benchmark solution that will also cover these emerging systems.

As described in Sec. "HEPscore", HEPscore has been designed to profile a compute system as a whole, including the contribution of co-processors to the overall system performance. HEPscore combines the event throughput (*wl-score*) of the HEP-Workloads into a single score, that is independent from the underlying hardware resources of the server. For example, the score will include the impact of a co-processor if it is utilized by HEP-Workloads. If the co-processor is not used by any of the HEP-Workloads, then the score will be identical to the value obtained with a server that has no co-processor.

For this reason, the Working Group is looking for HEP applications that can leverage not only traditional processors but also co-processors. To make the HEPscore benchmark meaningful for procurement, pledges, and accounting of heterogeneous resources, these applications need to be representative of the future workloads that will run on the WLCG heterogeneous resources.

The Working Group has already identified three GPU-capable HEP applications and created prototype HEP-Workloads container images for each application: (i) a simulation of particle trajectories in the LHC by the CERN Accelerator Group [32]; (ii) a physics event generator application [33]; and (iii) the CMS HLT online reconstruction [34]. Currently, these applications are not part of any WLCG production activity and, for this reason, were not included in HEPscore$_{\beta}$. Nevertheless, the Working Group sees the value of anticipating the developments in this area and is preparing a proof-of-concept HEPscore configuration for heterogeneous resources with a measurement campaign.

The Working Group is also extending the HEP benchmarks to ARM processors. One of the first achievements has been the ability to benchmark a small number of these processors (ThunderX2, AWS Graviton2) with HS06 and SPEC CPU 2017. This was made possible by two actions: (i) the container image to run HS06 and SPEC CPU 2017 has been built both on x86 and on ARM processors; (ii) the SPEC CPU 2006 toolkit, which does not include native support for modern ARM processors, has been extended to them.[2] Similarly, HEP-Workloads container images will be extended to the ARM systems, by accessing the CVMFS areas of the experiments that contain the libraries compiled on ARM chips.

[2] The procedure to include this extension to a site-owned SPEC CPU 2006 distribution is documented in Ref. [29].

Summary

This paper has described the HEP Benchmarks project developed by the HEPiX Benchmarking Working Group for measuring the CPU performance of a server using HEP applications. The project is motivated by the need to find a replacement for the HS06 benchmark that is used to benchmark x86 CPUs, which is currently the standard benchmark of the WLCG.

The project includes the utility HEPscore, designed to aggregate multiple profiling figures in a single benchmark score. HEPscore adapts well to Grid and HPC centers, and allows future extension to heterogeneous environments. The results presented using the demonstrator benchmark, HEPscore$_{\beta}$, show that it may be possible to create a new benchmark for CPUs based on HEP applications. The new benchmark will be developed once the LHC experiments finalize their *Run3* software and provide the Working Group with their reference workloads. The workloads of other non-LHC experiments will also be added to the set of HEP-Workloads. The Working Group will make its recommendations to the WLCG Task Force, who will then determine whether the new benchmark meets the requirements of the HEP community.

Acknowledgements We would like to thank Matthew Ens for his assistance in finalizing the manuscript. T. Sullivan and R.J. Sobie would like to thank the support of the Natural Sciences and Engineering Research Council of Canada.

Funding Open access funding provided by CERN (European Organization for Nuclear Research).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Michelotto M et al (2010) A comparison of HEP code with SPEC1 benchmarks on multi-core worker nodes. J Phys Conf Ser 219:052009. https://doi.org/10.1088/1742-6596/219/5/052009
2. Henning JL (2016) SPEC CPU2006 benchmark descriptions. SIGARCH Comput Archit News 34:1. https://doi.org/10.1145/1186736.1186737
3. WLCG (2021) The Worldwide LHC Computing Grid. http://wlcg.web.cern.ch. Accessed 11 Nov 2021

4. Charpentier P (2017) Benchmarking worker nodes using LHCb productions and comparing with HEP-SPEC06. J Phys Conf Ser 898:082011. https://doi.org/10.1088/1742-6596/898/8/082011
5. HEPiX Benchmarking Working Group (2021). https://w3.hepix.org/benchmarking.html. Accessed 11 Nov 2021
6. SPEC CPU (2017) Standard Performance Evaluation Corporation. https://www.spec.org/cpu2017/. Accessed 11 Nov 2021
7. Giordano D, Alef M, Michelotto M (2019) Next generation of HEP CPU benchmarks. EPJ Web Conf 214:08011. https://doi.org/10.1051/epjconf/201921408011
8. Giordano D (2021) HS06 -m32 Vs -m64. HEPiX Benchmarking Working Group. https://indico.cern.ch/event/624834/contributions/2646335/attachments/1486043/2307590/HS06_32_Vs_64_bits-giordano.pdf. Accessed 11 Nov 2021
9. Roy G et al (2015) Evaluation of containers as a virtualisation alternative for HEP workloads. J Phys Conf Ser. https://doi.org/10.1088/1742-6596/664/2/022034
10. Muralidharan S, Smith D (2019) Trident: an automated system tool for collecting and analyzing performance counters. EPJ Web Conf 214:08024. https://doi.org/10.1051/epjconf/201921408024
11. Giordano D, Santorinaiou E (2020) Next generation of HEP CPU benchmarks. J Phys Conf Ser 1525:012073. https://doi.org/10.1088/1742-6596/1525/1/012073
12. Giordano D (2017) Benchmark Working Group Update. WLCG Workshop in Manchester. https://indico.cern.ch/event/609911/contributions/2620190/attachments/1480455/2295576/WLCG_Workshop_2017_benchmarking_giordano.pdf. Accessed 11 Nov 2021
13. Valassi A et al (2020) Using HEP experiment workflows for the benchmarking and accounting of WLCG computing resources. EPJ Web Conf 245:07035. https://doi.org/10.1051/epjconf/202024507035
14. HEP-Benchmarks Repository (2021). https://gitlab.cern.ch/hep-benchmarks Accessed 11 Nov 2021
15. Blomer J et al (2017) New directions in the CernVM file system. J Phys Conf Ser 898:062031. https://doi.org/10.1088/1742-6596/898/6/062031
16. A definition of the Monte Carlo applications to generate, simulate, digitize and reconstruct particle physics collision events. In: A roadmap for HEP software and computing R&D for the 2020s computing and software for big science, vol 3, p 1. https://doi.org/10.1007/s41781-018-0018-8
17. Aad G et al (2008) The ATLAS experiment at the CERN large hadron collider. JINST 3:S08003. https://doi.org/10.1088/1748-0221/3/08/S08003
18. The CMS Collaboration (2008) The CMS experiment at the CERN LHC. JINST 3:S08004. https://doi.org/10.1088/1748-0221/3/08/S08004
19. Alves AA et al (2008) The LHCb detector at the LHC. JINST 3:S08005. https://doi.org/10.1088/1748-0221/3/08/S08005
20. Kou E et al (2019) The Belle II Physics ok. Progr Theor Exp Phys 12:123C01. https://doi.org/10.1093/ptep/ptz106
21. Docker containers (2021). https://www.docker.com/resources/what-container. Accessed: 11 Nov 2021
22. Singularity (2021). https://sylabs.io/singularity/. Accessed 11 Nov 2021
23. Atlas Athena license (2021). https://gitlab.cern.ch/atlas/athena/blob/master/LICENSE. Accessed 11 Nov 2021
24. Belle II license (2021). https://github.com/belle2/basf2/blob/main/LICENSE.md. https://doi.org/10.5281/zenodo.5574115. Accessed 11 Nov 2021
25. CMS CMSSW license (2021). https://github.com/cms-sw/cmssw/blob/master/LICENSE. Accessed 11 Nov 2021
26. LHCb license (2021). https://gitlab.cern.ch/lhcb/Gauss/-/blob/master/COPYING. Accessed 11 Nov 2021
27. WLCG HEPscore deployment Task Force (2021). https://indico.cern.ch/event/969947/. Accessed 11 Nov 2021
28. Charpentier P (2017) Benchmarking worker nodes using LHCb productions and comparing with HEPSpec06. J Phys Conf Ser 898:082011. https://doi.org/10.1088/1742-6596/898/8/082011
29. HEP-SPEC Container Orchestrator (2021). https://gitlab.cern.ch/hep-benchmarks/hep-spec. Accessed 11 Nov 2021
30. ActiveMQ (2021). https://activemq.apache.org. Accessed 11 Nov 2021
31. Venkitesh A et al (2020) Optimization of software on high performance computing platforms for the LUX-ZEPLIN dark matter experiment. EPJ Web Conf 245:05012. https://doi.org/10.1051/epjconf/202024505012
32. De Maria R et al (2019) SixTrack Version 5: status and new developments. In: Proceedings of IPAC 2019. Melbourne, Australia: JACoW, 2019, pp 3200–3203. https://doi.org/10.18429/JACoW-IPAC2019-WEPTS043
33. Valassi A et al (2021) Design and engineering of a simplified workflow execution for the MG5aMC event generator on GPUs and vector CPUs. EPJ Web Conf 251:03045. https://doi.org/10.1051/epjconf/202125103045
34. Bocci A et al (2020) Bringing heterogeneity to the CMS software framework. EPJ Web Conf 245:05009. https://doi.org/10.1051/epjconf/202024505009

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.