The SPECpowerNext Benchmark Suite, its Implementation and New Workloads from a Developer's Perspective

Norbert Schmitt
University of Würzburg
Chair of Software Engineering
Germany
norbert.schmitt@uni-wuerzburg.de

Klaus-Dieter Lange
Hewlett Packard Enterprise
SPECpower Committee Chair
USA
klaus.lange@hpe.com

Sanjay Sharma
Intel
SPECpower Committee Vice-Chair
USA
sanjay.sharma@intel.com

Nishant Rawtani
Hewlett Packard Enterprise
SPECpower Committee Member
India
nishant.rawtani@hpe.com

Carl Ponder
NVIDIA
SPECpower Committee Member
USA
cponder@nvidia.com

Samuel Kounev
University of Würzburg
Chair of Software Engineering
Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT

Innovation needs a competitive and fair playing field on which products can be compared and informed choices can be made. Standard benchmarks are a necessity to create such a level playing field among competitors in the server market for more energy-efficient servers. That, in turn, motivates their engineers to design more energy-efficient hardware. The SPECpower_ssj 2008 benchmark drove the increase of server energy efficiency by 113 times for single CPU servers, or 19 times on average. Yet, with added functionality and load, they are expected to consume a rising amount of energy. Additionally, server usage in data centers has changed over time with new application types. To continue the effort of increasing server energy efficiency, a new version, *SPECpowerNext*, is under development. In this work, after a short introduction to SPECpower_ssj 2008, we present the new implementation of *SPECpowerNext* together with the standardized way to collect server information in heterogeneous data centers. We also give insight, including preliminary measurements, into two of *SPECpowerNext* new workloads, the Wiki and the APA workload, in addition to an overview of both workloads.

CCS CONCEPTS

• **Hardware** → **Power and thermal analysis**; • **Computer systems organization** → Cloud computing; • **Software and its engineering** → *Software creation and management*; • **General and reference** → **Performance**; **Measurement**; • **Applied computing** → *Data centers.*

KEYWORDS

Benchmark, Performance, Energy Efficiency, Power Consumption, SPEC, SPECpower_ssj 2008, SPECpowerNext, Server, Cloud, Data Center

ACM Reference Format:
Norbert Schmitt, Klaus-Dieter Lange, Sanjay Sharma, Nishant Rawtani, Carl Ponder, and Samuel Kounev. 2021. The SPECpowerNext Benchmark Suite, its Implementation and New Workloads from a Developer's Perspective. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3427921.3450239

ICPE '21, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8194-9/21/04…$15.00
https://doi.org/10.1145/3427921.3450239

1 INTRODUCTION

Standard benchmarks drive innovation in the industry by creating a competitive and fair playing field on which products are rated and compared. This competitiveness motivates engineers to design better versions of their products. When it comes to developing more energy-efficient servers, the SPECpower_ssj 2008 benchmark is a significant milestone with 728 publicly available results [20]. Through public results and the contribution of the industry, server energy efficiency (operations-per-watt server efficiency) has increased by 113 times for servers with a single CPU (from 190 to 21603), or 19 times on average [19]. As the energy consumption of data centers was estimated to be around 263 TWh in 2020 by Andrea et al. [1] and is, in the optimistic scenario, about to rise to 1137 TWh until 2030, it is clear that the effort for better energy efficiency must continue. Additionally, over the years since the first release of SPECpower_ssj 2008, the usage of servers has changed, and with it, the workloads that are attractive from a competitive point of view to the server manufacturers. In an effort to advance and increase the server energy efficiency, a new version, *SPECpowerNext*, is currently under development.

The new *SPECpowerNext* introduces modern workloads that stress different hardware capabilities simultaneously instead of one at a time, namely the Wiki workload. It also features a new workload to benchmark servers with state-of-the-art Auxiliary Processing Accelerators (APA), also known as GPGPU systems. To better accommodate novel workloads as well as ease current and future developments, *SPECpowerNext* is now based on the Chaffeur

WDK [2] and includes a standardized way for collecting data about the server that is benchmarked. *SPECpowerNext* is planned to be released in 2021.

In this work, we give a short overview of the SPECpower_ssj 2008, together with the SSJ workload, as a comparison point with the new implementation in Section 2. Afterwards, in Section 3, we also describe the benchmark's new implementation in Chaffeur WDK that makes workload development easier and faster. With more and more heterogeneous data centers, *SPECpowerNext* also includes Redfish, a standardized interface to collect management information for servers, replacing the more error-prone shell scripts. We then also introduce the new workloads with preliminary measurements for the Wiki workload that show what has been achieved already in stressing multiple system components and an overview of the APA workload in Section 4 and conclude the paper in Section 5.

2 THE SPECPOWER_SSJ 2008 BENCHMARK

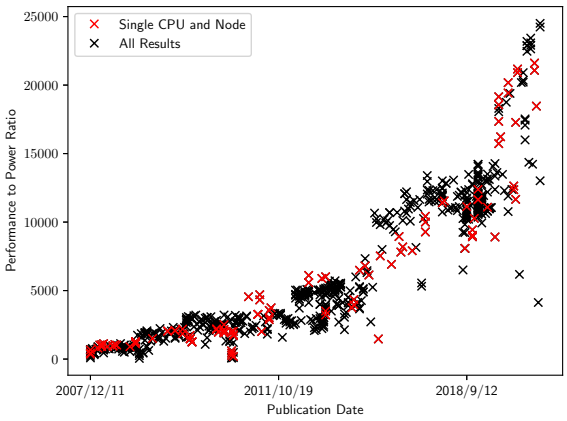
Before we illustrate and discuss the new implementation and novel workloads, we will describe briefly the current SPECpower_ssj 2008 benchmark. This is the first benchmark to allow power measurements alongside performance measurements to relate each other while at the same time acknowledging that servers in data centers are not continuously run at full load [14]. There are 728 publicly available results that show the performance-to-power ratio has steadily increased 113 times for servers with a single CPU and node (marked in red) in Figure 1a, with a minimum score of 190 and a maximum score of 21603. This development has taken place despite the increasing core count and amount of memory available in off-the-shelf hardware, visible in Figures 1b and 1c. For Figures 1a, 1b, and 1c, all 728 results have been parsed and filtered for non-compliance with the run and reporting rules [17], leaving 687 compliant results shown.

2.1 Overview

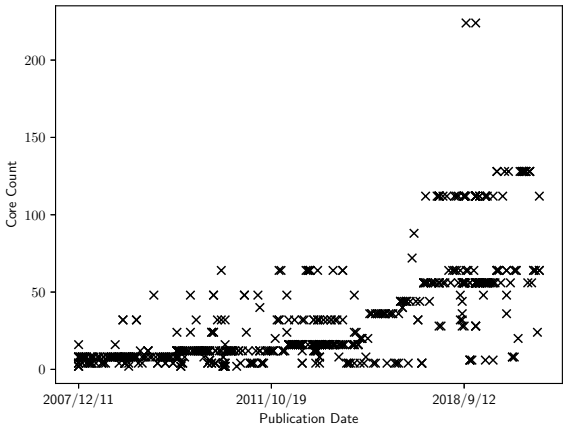
The minimal setup is shown in Figure 3a for SPECpower_ssj 2008 that consists of a minimum of two devices. The system under test (SUT) executes the workload and the controller device governs the benchmark execution through the control and collect system (CCS). The PTDaemon interface manages and collects the data from the two physical measurement devices, a power analyzer and a temperature sensor. Power and temperature data are then transmitted to the CCS, which measures the throughput and calculates the final score across all load levels. At the time when SPECpower_ssj 2008 was developed, all components needed to be self-implemented. Collecting server information, such as the CPU model, memory configuration, operating system, and Java version, had to be done with specialized shell scripts as no standardized method was available. And despite being developed before the five principles of a good benchmark had been discussed and published by Huppler et al. and Kistowski et al. [7, 22]. SPECpower_ssj 2008 was designed to adhere as closely as possible to these principles [2].

Relevance How closely the benchmark behavior correlates to behaviors that are of interest.

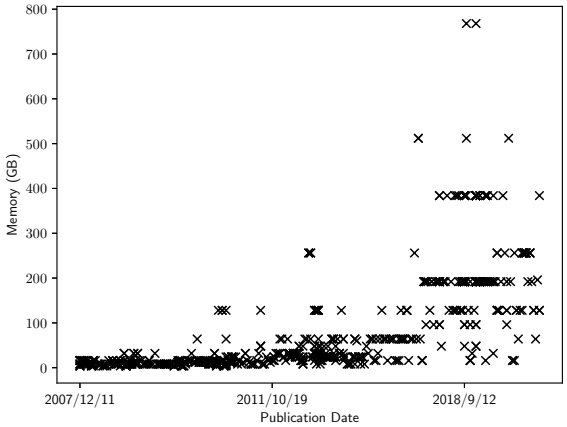
Reproducibility Consistently produce similar results when run with the same test configuration.



(a) Reported performance to power ratio in ssj_ops / sum of power.



(b) Number of cores installed during benchmark runs.



(c) Amount of memory installed during benchmark runs.

Figure 1: SPECpower_ssj 2008 results over time.

Fairness Allowing different test configurations to compete on their merits without artificial limitation.

Verifiability Providing confidence that the result is accurate.
Usability Avoiding roadblocks for users to run the benchmark.

2.2 Measurements

As servers are often not run at 100% load, the workloads are run at ten different load levels, an active idle run, and three calibration intervals. An example measurement execution with a single calibration interval and three load levels is shown in Figure 2. An actual SPECpower_ssj 2008 run would use load intensities from 100% down to 10% in 10% steps and an active idle with 0% load intensity. Each interval consists of a pre-measurement, measurement, and post-measurement phase. The pre- and post-measurement phases also are called ramp-up and ramp-down phase. The pre- and post-measurement phases let the system reach a steady-state at the current load-level. First, the SUT's maximum load must be determined by the three calibration runs saturating the SUT. The average of the last two calibration runs is used as the maximum throughput achievable by the SUT. From the resulting maximum load, the throughput is then determined for each load level. To be able to hold a steady load level below 100%, the workloads are designed as transactional loads. This means that a transaction blocks the resources until finished and before a new transaction is dispatched. By calibrating for the maximum load at 100%, the benchmark harness can then calculate the number of transactions for a particular load level. No transactions will be dispatched for the active idle interval. The standard ramp-up and ramp-down phases for each interval of SPECpower_ssj 2008 are 30 seconds, and the measurement phase is 240 seconds.

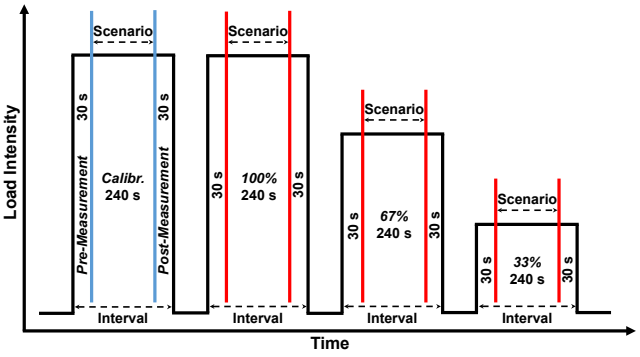


Figure 2: Example measurement execution with single calibration interval (blue), three load levels (red), and without active idle interval [8].

2.3 The SSJ Workload

The SSJ workload is designed as an ordering and warehouse application on which multiple user interactions are possible. The interactions are mixed but must be in a definite order. The mix of interactions stresses the CPU and memory of the SUT. SSJ supports six different user interactions, implemented as transactions that occur at a specific frequency. The rate at which a transaction occurs is approximated by the benchmark harness to match the rates shown below. The transactions are (occurrence rates are shown in parentheses) [14]:

- New Order (30.3%): A new order is inserted into the system.
- Payment (30.3%): Record a customer payment.
- Order Status (3.0%): Request the status of an existing order.
- Delivery (3.0%): Process orders for delivery.
- Stock Level (3.0%): Find recently ordered items with low stock levels.
- Customer Report (30.3%): Create a report of recent activity for a customer.

3 REIMPLEMENTING AN INDUSTRY STANDARD BENCHMARK

The reimplementation of a well-received and successful benchmark like SPECpower_ssj 2008 is a challenging task. Not only should it fulfill certain criteria but, at the same time, ease development and reduce the workload on the result submission reviewers. Both are achieved by using the Chauffeur WDK and the Redfish standard described in the following sections. It must be stated that all presented implementation details, measurements, or other information are from a pre-release version of *SPECpowerNext* and are subject to change in the future if necessary.

3.1 Chauffeur WDK

SPECpowerNext is based on Chauffeur WDK, developed in 2013. The main goals of Chauffeur WDK are coherent with the design goals of the server efficiency rating tool (SERT) [16], which is also based on Chauffeur WDK. With Chauffeur WDK used in the SERT and *SPECpowerNext*, the maintenance overhead for *SPECpowerNext* is reduced. Freeing developers to focus on the implementation on new workloads rather than the benchmark harness. The goals are the following [2]:

Stress multiple system components. Chauffeur WDK allows the sequential execution of so-called worklets that stress a server's components.
Multiple synthetic worklets. Worklets can be developed easily with Chauffeur WDK, allowing to write a large variety of synthetic codes.
Multiple load levels. Chauffeur WDK already has implemented a graduated measurement series (shown in Figure 2) that supports running a worklet at user-defined load levels after a calibration.
Cross-platform support. Chauffeur WDK is written in Java, allowing cross-platform support.

All the mentioned goals apply to *SPECpowerNext*, making Chauffeur WDK a good choice as a basis for the new implementation of *SPECpowerNext*. Additionally, Chauffeur WDK has a very similar setup than SPECpower_ssj 2008 shown in Figure 3a and 3b. Instead of a custom built CCS, which can only run the SSJ workload, Chauffeur WDK uses a more generalized Director and Reporter. The Director can run a variety of workloads independent of their actual implementation and is responsible for the execution of the workloads and worklets. The Reporter collects and compiles the results of a benchmark run in a better readable format. The second large alteration is the differentiation of workloads and worklets.

A worklet is the smallest unit that is executed on the SUT that stresses one or more system components like CPU and memory. A workload, on the other hand, consists of one or more worklets. The remaining setup, power, and temperature measurement collection are handled as in SPECpower_ssj 2008 by the PTDaemon interface. Chauffeur WDK is written in Java, making it portable across multiple systems.

SPECpowerNext also comes with a newly developed GUI on the controller (see Figure 3c). The GUI makes the proper configuration, complying with the run and reporting rules, of *SPECpowerNext* easier and less error-prone by checking user input early in the configuration process. It also reduces the amount of in-depth knowledge of the configuration files for the benchmark user, making the benchmark more accessible to a wider audience.

Chauffeur WDK provides an easy-to-use interface to develop new worklets that can be run in benchmark suites using it. In the simplest form, only three interfaces need to be implemented: The `SuiteDescription`, the `User`, and the `Transaction`. However, abstract classes are provided to simplify further the process of workload development. This strong abstraction from how workloads and worklets are executed allows for faster and easier development. Its Java implementation also makes it platform-independent, allowing to develop workloads and worklets on one system while running them on a multitude of Java-supported systems. Also, native code is supported through the Java native interface. While the current workloads do not depend on networked resources, Chauffeur WDK also allows implementing workloads that need network communication. Additionally, Chauffeur WDK is maintained by the Standard Performance Evaluation Corporation (SPEC), allowing the developer to focus on the development of its workloads instead of the necessary infrastructure code to execute them and collect data [2].

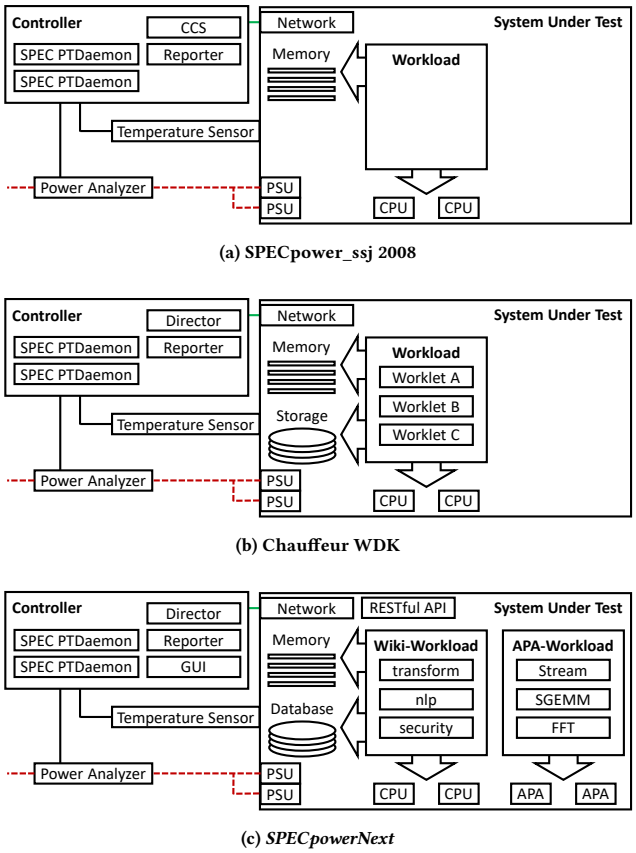


Figure 3: Components and minimal setup of SPECpower_ssj 2008, Chauffeur WDK, and *SPECpowerNext* [15, 18]

3.2 Redfish

Today's data centers are heterogeneous in nature, (i.e., the compute servers, storage, and networking devices may consist of multiple different vendors). Here, each vendor would use his own propriety standards, tools, and protocols, each of which requires significant expertise, thereby limiting the scalability of the data center or result in vendor lock-ins. The Redfish Standard from DMTF [5] helps in preventing these vendor lock-ins and increasing data center scalability by providing a standard suite of specifications to manage these heterogeneous components using a simple and secure HTTPS interface and delivered as RESTful APIs on the SUT, shown in Figure 3c. The remainder of this section is based on [6] and [4].

Within the *SPECpowerNext* benchmark, the Redfish standard automatically is used to gather hardware information of the SUT. The flow for retrieving hardware information is shown in Figure 4. When the Redfish data collection module is first invoked by the *SPECpowerNext* harness, an authenticated connection is then established to the Redfish interface of the SUT. After successful authentication into Redfish, each API URL represents either a collection of resources or a singleton resource. A group of similar singleton resources represents a collection resource. For example, multiple computer systems are contained as members of a "Systems" collection, where each system instance will have its own CPU, memory, etc. The Redfish standards dictate the output schema and naming convention of Resource collections. Moreover, Redfish also dictates the output schema of the singleton resources; however, the naming convention of singleton resources is left open to the hardware vendor implementing Redfish. For example, the API of a "Systems" collection is "`redfish/v1/Systems`" for which the path and schema are well defined by the Redfish standards. However, a singleton resource under "Systems" collection for which the schema is defined by Redfish could be represented by `<sr>` in the API URL "`redfish/v1/Systems/<sr>`" where the exact naming convention of `<sr>` can be defined by the hardware vendor. A module within *SPECpowerNext* self discovers these singleton resource names by taking advantage of the Redfish's Hypermedia capabilities, (i.e., the collection resources are aware and expose the identifiers of the pool of all available singleton resources using their corresponding APIs). This property allows us to identify the singleton resource names used by the vendor by gathering the information from the data found in the collection resource output. Using this technique, the module is able to construct the complete APIs for all the singleton resources from which the information needs to be gathered. Once all these APIs have been constructed, the module performs REST calls to these APIs using HTTPS protocol and stores the raw JSON

response obtained for each of these REST calls. The Redfish module then parses these JSON outputs obtained to collate the necessary system information. Finally, the module stores and converts the results into a unified format as expected by the *SPECpowerNext* test harness for further processing.

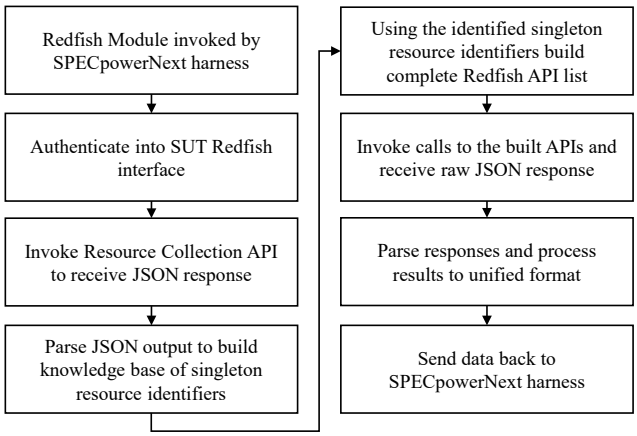


Figure 4: Gathering hardware information with Redfish.

The introduction of automated hardware data collection and processing in *SPECpowerNext* benchmark would enable auto-populating the Full Disclosure Reports (FDR), therefore, simplifying the benchmark users task describing the execution environment. Additionally, it would also assist during the result review process of the committee as data collected using Redfish can undergo lesser scrutiny, saving reviewers time, as compared to results where the FDR has been populated manually.

4 NEW WORKLOADS

To adapt to the changing usage of cloud servers and with new upcoming usage scenarios, new workloads for a relevant benchmark suite are necessary. We, therefore, introduce the new workloads, Wiki and APA, togehter with their included worklets in the following sections.

4.1 Wiki Workload

The Wiki Workload consists of three distinct worklets:

- **Transformation.** The transform worklet includes compression and decompression, data serialization, and JSON parsing. Additional data transformation operations are being considered for the final release.
- **Natural Language Processing (NLP).** The nlp exercises Natural Language Processing transactions using the OpenNLP library. Transactions include language detection, tokenization, sentence detection, and part of speech detection.
- **Security.** The security worklet tests a variety of security operations, including encryption, decryption, signatures, secret key generation, and key exchange. It uses multiple algorithms for each of these operations, including AES-128, AES-256, CHACHA20_POLY1305, SHA256, SHA384, RSA, and ECDSA, and others.

To show that the Wiki workload can stress different components of the SUT, as depicted in Figure 3c, we measured an example run of *SPECpowerNext* on a state-of-the-art HPE ProLiant DL380 Gen10 server. The server is equipped with two Intel® Xeon® Platinum 8280 CPUs with 28 cores and 56 threads each, as well as 192GB of memory in 12 modules. For hard drives, two 1TB disks in RAID 0 mode are installed. We took measurements with *SPECpowerNext* PK15, a pre-release candidate, so all measurements are preliminary.

The results in Figure 5 clearly show the four load levels in the CPU utilization. While visibly clear, they have expected fluctuations. These fluctuations can be explained as the CPU utilization must not necessarily be correlated to the load level. In fact, as already mentioned in the SPECpower_ssj 2008 design document, CPU utilization is measured in different ways on different platforms and *SPECpowerNext*, as well as SPECpower_ssj 2008, target the throughput instead [15]. Yet, Figure 5 shows that the Wiki workload can stress the CPU of the SUT to a certain degree at different load levels. On other platforms, though, the actual CPU utilization could look different. All three worklets scale well over the four measured load-levels.

The available (free) memory of the SUT also is shown, but the load levels are not visible. To observe the different load levels in the amount of available memory during a measurement interval, the executed workload must be specifically tuned to do so. While the Wiki workload is synthetic, its design goal is a mixed workload not explicitly adjusted to memory. Yet, the Wiki workload still should utilize most of the installed memory as it clearly does, bringing the free memory down to about 75GB of the 192GB installed in the NLP worklet. For the transformation and security worklet, the available memory is slightly higher, yet all three worklets utilize the SUT's memory as intended. The reason behind the minor difference in available memory between the NLP and the other two worklets most likely stems from the implementation but is something that can be investigated in the future.

Figure 5 also highlights that the Wiki workload stresses not only the CPU and memory as the SSJ workload (described in Section 2.3) but has an additional IO component. *SPECpowerNext* achieves IO stress through a large number of Wikipedia articles, approximately 30GB, stored on the disc. As can be seen, the IO component mainly consists of file read operations and close to zero file write operations. File read operations also clearly show the four different load levels, 100%, 75%, 50%, and 25%, for the transformation measurement interval. For the NLP and security measurement series, the file operations are minor. While the Wiki workload seems to stress IO through numerous file operations, our measurements show a current downside of the new workload; the bytes read and written to disk are small. Most data transfers happen at the end of a measurement series, reaching maximum values of just over 6MB per second. This issue of low data transfer to and from the disk is a matter that needs to be addressed in future versions of the Wiki workload. It is also worth mentioning that most file operations are file read operations while the actual data transmissions to and from disk are data writes.

The Wiki workload is, therefore, a suitable new workload for the upcoming *SPECpowerNext*, allowing it to stay relevant for server manufacturers. Hence, *SPECpowerNext* further fosters the development of more energy-efficient hardware over time. Yet, the newly

Figure 5: Preliminary measurements of CPU time, available memory, file operations, and disk data rates on a *SPECpowerNext* Wiki workload run. Measurement phases with four load levels, 100%, 75%, 50%, and 25%, are highlighted for each of the three worklets.

introduced stress on the IO component of the server needs more attention in future versions.

4.2 Auxiliary Processing Accelerator Workload

While many APA benchmarks exist, for example, LAMMPS [13] for atomic/molecular simulations or SHOC [3] for heterogeneous computing, their focus is on the maximum reachable performance. The new APA workload in *SPECpowerNext*, on the other hand, is for measuring the power-to-performance ratio under different load levels, a novelty in industry-standard benchmarks containing workloads aiming at accelerator hardware. The following section is based mainly on the previous work of Kistowski et al. [23].

As shown by Kistowski et al., by using transactional loads necessary for generating load levels, GPUs also scale with the workload. However, the scaling is not as pronounced as the CPU scalability. The reasons are twofold. First, on a system with an accelerator, the CPU adds to the static power reducing the range of scalability, even in the case when it is not computing. Second, accelerators are designed for maximum load in HPC environments and have fewer power-saving optimizations than CPUs. The CPU's has also a higher scalability when an OpenCL Fast-Fourier-Transformation (FFT) is run on the CPU without an accelerator.

The new APA workload transfers the findings in [23] into an industry-standard benchmark. Next to the FFT worklet, which has

been reimplemented, the *SPECpowerNext* APA workload implements nine additional worklets, totaling ten worklet kernels, designed for transactional loads. APA worklet kernels are package executables compiled explicitly for the execution on a GPU.

There are currently two dominating languages to write kernels, either CUDA or OpenCL. CUDA is NVIDIA's vendor-specific kernel language compatible only with NVIDIA GPUs [11]. Nevertheless, it is popular through a large number of libraries. OpenCL supports the majority of hardware accelerators and also can be run solely on the CPU if necessary. This is achieved by compiling the kernel at run-time [21]. For each of the following mentioned worklets, two kernel implementations exist, one for CUDA and one for OpenCL.

- Stream/Copy
- Stream/Scale
- Stream/Add
- Stream/Triad
- Stream/DAXPY
- Stream/Fill
- Stream/Sum
- Stream/Dot
- SGEMM
- FFT

The APA worklets *Copy*, *Scale*, *Add*, and *Triad* are inspired, among others, by the Stream memory bandwidth benchmark [9, 10]. The *Copy* worklet makes a copy of a memory location to another. *Scale* multiplies a value in a memory location with a scalar value and stores the result in a new memory location. The *Add* worklet sums the values of two memory locations, storing the result in a third one. At the same time, the *Triad* first uses a scalar value multiplication on one memory location before the addition but is identical to *Add* otherwise. *DAXPY* works very similar to *Triad* but uses vectors instead of single memory locations. The *Fill* workload writes a constant scalar into a memory location. The *Sum* worklet works by summing up consecutive memory locations in a single value stored to a new memory location. *Dot*, as the name implies, computes the dot product on matrices. *SGEMM* performs a matrix-matrix operation with the following pseudocode 1, from [12], where `alpha` and `beta` are scalar values, and `A`, `B`, and `C` are matrices. As already mentioned, the *FFT* worklet performs Fast-Fourier-Transformations. The *SGEMM* and *FFT* worklets are implemented with the help of the cuBLAS and cuFFT libraries in case of the CUDA implementation, and the clBLAS and clFFT libraries for OpenCL.

```
C := alpha * op(A) * op(B) + beta * C        (1)
```

Figure 6 shows a general sequence of a transaction in an APA worklet similar to a typical transaction of other worklets. First, there is an initialization phase to prepare test-data for a compute-device, followed by the device's buffers' preparation, or putting it differently, transferring the test data to the device. After the data is transferred to the device, the actual computation executes, results are retrieved, and finally validated. Validation, though, is handled in the background, not to interfere with the compute-device.

In this section, we highlighted, based on previous work, that APA workloads do show scalability across load levels in transactional workloads and how the prototype of an FFT worklet resulted in ten worklets for servers with an APA. An example of how research can be transferred into sophisticated industry-standard benchmarks.

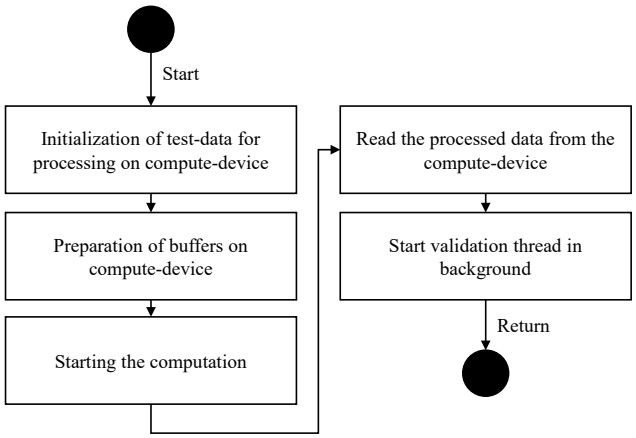


Figure 6: Typical APA worklet transaction execution.

5 CONCLUSION

To drive innovation of energy-efficient servers for large scale data centers, new benchmarks are required. While the SPECpower_ssj 2008 benchmark certainly helped to increase the power-to-performance ratio by 113 times for single CPU and single node servers, the usage servers in data centers changed over time with new workloads and a more heterogeneous hardware landscape. For this reason, we presented the current state of development of the *SPECpowerNext* benchmark. It is based on a technically mature framework, Chauffeur WDK that allows the benchmark developers to focus their time on designing and implementing new workloads instead of the surrounding benchmark harnesses. To accommodate heterogeneous data centers, the modern Redfish standard is introduced to *SPECpowerNext*. Redfish additionally reduces the time necessary to review benchmark results. To satisfy the changing application mix, *SPECpowerNext* introduces two new workloads. First is the Wiki workload that allows for stressing multiple hardware components simultaneously, although disc operations are a current work-in-progress as the preliminary measurement is showing. Second is the APA workload for stressing state-of-the-art hardware accelerators like GPUs with novel transactional loads able to reach specific load levels. Yet, the development of *SPECpowerNext* is not completed.

ACKNOWLEDGEMENTS

The authors wish to acknowledge current and past members of the SPECpower Committee, and SPEC Research Power Working Group who have contributed to the design, development, testing, and overall success of SPECpower_ssj 2008. SPECpower, SPECpower_ssj, and PTDaemon are registered trademarks of the Standard Performance Evaluation Corporation. All rights reserved; see spec.org as of 10/24/2020.

REFERENCES

[1] Anders Andrae and Tomas Edler. 2015. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges* 6, 1 (Apr 2015), 117–157. https://doi.org/10.3390/challe6010117

[2] Jeremy Arnold. 2013. Chauffeur: A framework for measuring Energy Efficiency of Servers.

[3] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 63–74.

[4] DMTF. 2019. *End-to-End Interoperable Management: The Standards Requirement for Modern IT*. White Paper. Distributed Management Task Force.

[5] DMTF. 2019. *Redfish Specification*. Specification DSP0266 Version 1.8.0. Distributed Management Task Force.

[6] DMTF. 2020. *Redfish – Simple and Secure Management for Converged, Hybrid IT*. Technical Note. Distributed Management Task Force.

[7] Karl Huppler. 2009. The Art of Building a Good Benchmark. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–30.

[8] Samuel Kounev, Klaus-Dieter Lange, and Jóakim von Kistowski. 2020. *Systems Benchmarking* (1 ed.). Springer International Publishing. https://doi.org/10.1007/978-3-030-41705-5

[9] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

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

[11] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable Parallel Programming with CUDA: Is CUDA the Parallel Programming Model That Application Developers Have Been Waiting For? *Queue* 6, 2 (March 2008), 40–53. https://doi.org/10.1145/1365490.1365500

[12] nVidia. 2020. cuBLAS Library User Guide v11.1. https://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf.

[13] Steve Plimpton, Paul Crozier, and Aidan Thompson. 2007. LAMMPS-large-scale atomic/molecular massively parallel simulator. *Sandia National Laboratories* 18 (2007), 43.

[14] SPEC. 2012. Design Document SSJ Workload SPECpower_ssj2008. https://www.spec.org/power/docs/SPECpower_ssj2008-Design_ssj.pdf.

[15] SPEC. 2012. Design Overview SPECpower_ssj2008. https://www.spec.org/power/docs/SPECpower_ssj2008-Design_Overview.pdf.

[16] SPEC. 2013. Server Efficiency Rating Tool (SERT) Design Document 1.0.2. https://www.spec.org/sert/docs/SERT-Design_Doc.pdf.

[17] SPEC. 2016. SPECpower_ssj2008 Run and Reporting Rules. https://www.spec.org/power/docs/SPECpower_ssj2008-Run_Reporting_Rules.html, Accessed: 21.10.2020.

[18] SPEC. 2017. Chauffeur™ Worklet Development Kit (WDK) User Guide 2.0.0. https://www.spec.org/chauffeur-wdk/docs/Chauffeur-User_Guide.pdf.

[19] SPEC. 2019. Beyond performance to power consumption. https://www.spec.org/30th/power.html, Accessed: 06.10.2020.

[20] SPEC. 2020. All Published SPECpower_ssj2008 Results. https://www.spec.org/power_ssj2008/results/power_ssj2008.html, Accessed: 13.10.2020.

[21] J. E. Stone, D. Gohara, and G. Shi. 2010. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science Engineering* 12, 3 (2010), 66–73. https://doi.org/10.1109/MCSE.2010.69

[22] Jóakim von Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. 2015. How to Build a Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)* (Austin, TX, USA) *(ICPE '15)*. ACM, New York, NY, USA. https://doi.org/10.1145/2668930.2688819

[23] Jóakim von Kistowski, Johann Pais, Tobias Wahl, Klaus-Dieter Lange, Hansfried Block, John Beckett, and Samuel Kounev. 2019. Measuring the Energy Efficiency of Transactional Loads on GPGPU. In *Proceedings of the 19th ACM/SPEC International Conference on Performance Engineering (ICPE '19)*. ACM, New York, NY, USA.