Pinpoint resource allocation for GPU batch applications

Tim Voigtländer[1], Manuel Giffels[1], Günter Quast[1], Matthias Schnepf[1] and Roger Wolf[1]

[1]Karlsruhe Institute of Technology, Karlsruhe, Germany

E-mail: tim.voigtlaender@kit.edu, Manuel.Giffels@kit.edu, guenter.quast@kit.edu, matthias.schnepf@kit.edu, roger.wolf@kit.edu

Abstract. With the increasing usage of Machine Learning (ML) in High energy physics (HEP), there is a variety of new analyses with a large spread in compute resource requirements, especially when it comes to GPU resources. For institutes, like the Karlsruhe Institute of Technology (KIT), that provide GPU compute resources to HEP via their batch systems or the Grid, a high throughput, as well as energy efficient usage of their systems is essential. With low intensity GPU analyses specifically, inefficiencies are created by the standard scheduling, as resources are over-assigned to such workflows. An approach that is flexible enough to cover the entire spectrum, from multi-process per GPU, to multi-GPU per process, is necessary. As a follow-up to the techniques presented at ACAT 2022, this time we study NVIDIA's Multi-Process Service (MPS), its ability to securely distribute device memory and its interplay with the KIT HTCondor batch system. A number of ML applications were benchmarked using this approach to illustrate the performance implications in terms of throughput and energy efficiency.

1 Usage of GPU resources by the CMS Collaboration

High energy physics (HEP) has become a compute resource intensive field of research due to the immense amounts of data provided by the CERN LHC [1] and its experiments. This trend is sure to continue as the LHC enters its high luminosity phase in the coming years and the available data increases by close to an order of magnitude. The drive towards efficient use of the necessary, but costly hardware has been a topic of increasing importance for all related experiments, including the Compact Muon Solenoid (CMS) Collaboration [2]. The use of GPUs has been a topic of great importance for this goal as CMS estimates that GPU resources can provide up to 2.8 times the amount of compute capacity for the same price when compared to CPU hardware, thereby leading to a 20-26% increase of available compute resources by 2030 [3]. In addition, many machine learning based analyses require the massive parallel processing power provided by GPU resources to be feasible at all. One GPU cluster, that has been set up for the purpose of supporting CMS and local HEP analyses, is the Throughput Optimized Analysis System (TOpAS) [4]. It is a throughput oriented compute cluster with 24 V100S, 8 V100 [5] and 24 A100 [6] NVIDIA GPUs, providing a significant amount of parallel processing power to end user analyses of all shapes and sizes.

This paper is a follow-up to a 2022 ACAT contribution [7] and builds on the presented topics while providing an alternative to the NVIDIA Multi-Instance GPU (MIG) [8] setup discussed at that time. The main feature that is benchmarked here is NVIDIA's Multi-Process Service (MPS) [9], a software that supports the execution of concurrent CUDA [10] processes on a GPU. In addition, a comparison with the previously covered MIG is drawn and the use of the MPS software as part of the TOpAS batch system is illustrated.

2 Resource granularity in the context of GPU

To keep operating costs as low as possible, one of the most important goals of HEP-computing is the maximal utilization of its deployed hardware. The drive for maximal utilization applies to common resources like CPUs and memory, but even more so for costly hardware like GPUs.

In many cases, a single workload will not fully utilize a hardware resource and multiple different tasks (in this context also referred to as jobs) have to run concurrent on the same hardware to achieve optimal utilization. Such sharing is inherently risky, as it is uncertain to which degree jobs influence each other or if there are enough resources to run all of them at once. One solution to some of these issues can be found in the various implementations of batch systems.

The implementation of choice for TOpAS, HTCondor [11], is commonly used in HEP computing for CMS because of its adaptability and open source nature. It provides resource provision, job scheduling and limited isolation between the jobs. The jobs that are sent to match with the available job slots through such a system can have widely varying hardware requirements, depending on their tasks. Especially with GPU-jobs submitted by end users for their analyses, this can range anywhere from a few percentages of a single GPU's processing power to the use of multiple GPUs at once. While isolation is important, a movable boundary between the job slots is necessary to properly fit as much work as possible on the hardware. For some hardware like RAM or scratch space, this is fairly easy, as they can be distributed in pieces of almost any size. CPU cores are more coarsely granular, and real life clusters often do not assign specific CPU cores to jobs, but instead only keep track of the overall CPU usage across all available cores.

GPUs would profit from a similar mechanism, but two reasons make this challenging. Each GPU possesses an amount of device memory (VRAM) that is used to store data while it is worked on by the GPU. This need for data locality makes it difficult to treat multiple GPUs uniformly, as one would do for CPUs, leading to them being individually assigned to their respective job(s). In addition, performance deteriorates with multiple processes running on the same GPU without additional supporting software or hardware features. Due to these complications, the default of HTCondor is that one GPU can only be assigned to at most one job, although it is possible to assign multiple GPUs to a single job. One solution to this issue of coarse granularity of GPU resources has been the topic of the previous contribution [7] and this paper builds on those findings by presenting an alternative.

3 NVIDIA MPS

While the previous work has concentrated on comparing MIG with the default case, this paper concentrates on the more software based approach of MPS. The MPS software exists and is supported in its current form since NVIDIA's *Volta* GPU-generation. While a software with the same name was available for older graphics cards, it does not share the same features as with newer cards. We will focus on the post-*Volta* capabilities of this service in this paper, but information on the pre-*Volta* behavior can be found in the documentation [9]. It should be noted that both MIG and MPS can be used together if necessary.

The default behavior of post-*Volta* NVIDIA GPUs, when confronted with multiple processes trying to access them, is to apply time slicing. Each of the processes opens a context for the GPU, but these contexts are not allowed to execute concurrently and are instead allotted time on the GPU in a round-robin scheduling. While such usage is safe, it also leads to severe under utilization of resources, especially when a larger number of small scale contexts is involved, since only one of them can work at a given time. Tests with OpenCL [12] on NVIDIA V100 and A100 GPUs have indicated that this is also the default behavior for non-CUDA processes working with the GPU. The previously discussed MIG approach attempts to solve this issue by splitting the GPU into multiple smaller parts, each of which acts as an independent GPU, leading to multiple concurrent contexts when viewing the GPU as a whole.

MPS goes the opposite direction by taking multiple contexts and fusing them into one larger context. This also results in the intended *one context per GPU*, as with the MIG approach. A sketch of the default, the MIG and the MPS use case is illustrated in figure 1. MPS provides fully isolated GPU address spaces for VRAM protection and limited error containment. This ensures that if any of the contexts combined by MPS experience a fatal error that affects the entire GPU, all processes generate the error, protecting them from incorrect results. It also supports limited execution resource provisioning for both, active thread usage and pinned device memory, enabling protection regarding overbooking of device resources. These features are helpful in isolating processes from each other, but they are not as effective as the ones provided through MIG, as will be discussed in section 6.

The setup for MPS is very flexible, as it optimizes any CUDA context that has access to it, enabling it to work with contexts of widely varying sizes at the same time. This makes it well suited for the use with the variety of different workloads that are sent to a batch system, as no reconfiguration is necessary compared to MIG. The theoretical maximum impact would be achieved with up to 48 processes sharing a single GPU efficiently, although it is unlikely that this many GPU processes would still fit the overall resource limits of the hardware.

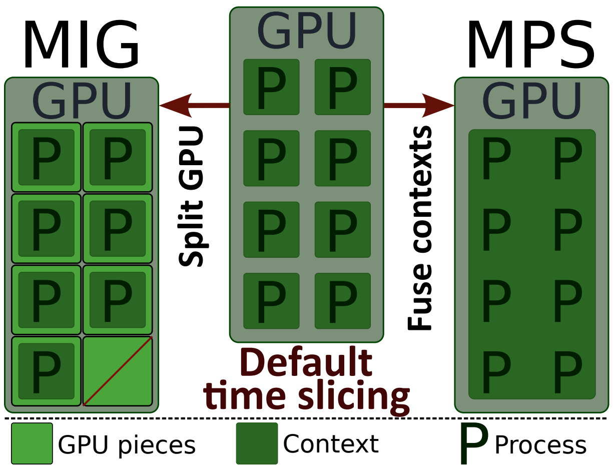

Figure 1: Sketch of MIG and MPS behavior compared to default. Time slicing is only used with multiple contexts per GPU/GPU-piece. MIG reserves one piece for overhead.

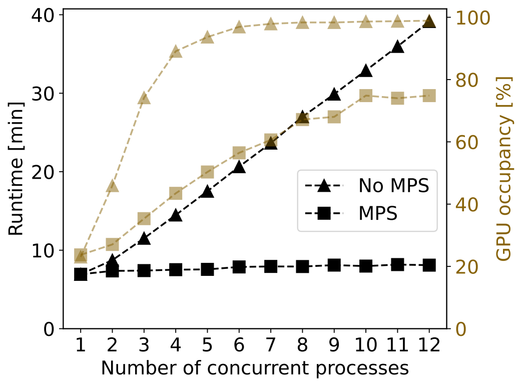

Figure 2: Runtime and GPU occupancy of bare-metal workload with and without MPS.

3.1 Details on special MPS behavior
There are a number of behavior details that are not obvious from the documentation alone, which will be touched upon in the following, as they are of importance for the later introduction of MPS into our local HTCondor setup.

All processes that MPS fuses have to be from the same user. Processes from different users will never run in the same MPS process on the same device, instead they will wait for the previous user's processes to finish before switching the *owner* of the MPS process to the new user. As all jobs run as a common user on TOpAS, instead of themselves, this is not an issue. As a result, the setup presented in the following can be used by an entire group that submits jobs to such a batch system without any switching of the active MPS owner.

By default, one MPS process will be spawned for the entire machine, independent of how many GPUs are installed. For large numbers of client processes that have to be managed by MPS, this can lead to performance issues as the MPS process might be overwhelmed. The MPS documentation mentions a limit of 48 client CUDA contexts per device. Tests with a machine with multiple GPUs have shown that slowdown starts to occur around 50 processes managed by one server. This does align with the stated 48 processes, but it indicates that the limit is not just for the number of contexts per GPU, but also for the number of allowed contexts per MPS process. Setting up an MPS process for each GPU individually lead to improved performance when compared to a single MPS process for the entire machine.

Any context that is not fused by an MPS process will be placed into the time-sliced scheduler together with the MPS context. It is therefore possible to have multiple MPS processes run on the same GPU, although it is not recommended for performance reasons. It also allows to explicitly exclude certain contexts from an MPS process, which is necessary for lightweight monitoring processes that run as non-default users. Normally, they would have to wait for all other work of the MPS process to finish before running a simple query, but in this setup, they can run interleaved through time slicing with the main MPS processes, leading to marginal impacts on the overall performance.

4 Viability check and performance benchmarks
In order to test whether MPS can provide performance improvements to a HEP compute cluster, a set of benchmarks based on the ones from the 2022 contribution [7] were performed. Both, the workloads and the hardware were the same as before. A set of small scale TensorFlow [13] neural network training was run on a machine with eight NVIDIA A100 GPUs. The workload is part of a real HEP workflow and consists of a total of 414 jobs with slightly differing input parameters. For the first two benchmarks, the

same single job as in the previous contribution was used. For the final benchmark, all jobs were used. For additional details, we refer to the ACAT 2022 contribution [7].

4.1 Bare-metal performance benchmark

In the first benchmark, the same workload with a fixed number of training epochs was run on bare metal, once with MPS activated and once without. In all runs, processes were also assigned two CPU threads each. To see the direct impact on concurrent processes, multiple runs with differing numbers of instances of the same training were run on one A100 GPU. The maximum number of concurrent copies was limited to a maximum of twelve due to VRAM constraints in this benchmark for both versions. The results can be seen in figure 2. It can be observed, that the runtime without MPS increases close to linearly as more concurrent processes are added to the GPU. The reported GPU occupancy also increases sharply, without visibly benefiting the runtime of the workload, indicating the expected time slicing behavior discussed in section 3. For the benchmark with MPS, the runtime shows little to no increase and the GPU occupancy grows much more gradually with the increasing number of processes. This indicates that time slicing was not applied here, and instead the different processes perform their work on the GPU concurrently. The overall increase in throughput measured for twelve concurrent processes is close to a factor of five.

4.2 Energy efficiency benchmark

The second benchmark intends to judge the energy efficiency of different methods, described as the amount of energy necessary to run the training process of the workload for a fixed number of epochs. The benchmark scenarios aim to simulate the optimal high throughput case in which as many of the available hardware resources as possible are busy, as this is also the point at which the hardware runs the most energy efficient. Apart from the CPU only scenario, none of the scenarios manage to completely utilize the available CPU resources. While this is not ideal, it was not possible to fully utilize CPU and GPU at the same time during the GPU scenarios. It is therefore not possible to give a perfectly fair comparison between the methods, as additional work could have been performed on the remaining CPU threads with unknown impact on the overall power consumption. The tools used to measure the power usage during the benchmark (*ipmitool* [14] for overall power consumption and *nvidia-smi* [15] for GPU power consumption only) state a systematic uncertainty of 5 %, which leaves us with a somewhat uncertain but still meaningful result.

There are five setups that have been considered. Firstly, the CPU only setup (CPU), in which each training was assigned four CPU threads and a total workload of 64 instances were running concurrently to occupy the 256 available CPU threads of the machine. The choice for this was determined by a set of earlier test runs, that resulted in four being the most energy efficient number of CPU threads for that specific workload. The idle power use of the GPUs has been removed from the measurements for this scenario to emulate a machine that is purely CPU based. The other four setups try to utilize the GPUs in different ways, relevant for the use in a batch system. The second setup (Default) simulates the exclusive use that is the default for the use of GPUs with HTCondor. It assigns only one job to each GPU at most, generally resulting in only one process running on that GPU. The third (Simple sharing) is an example of a GPU that is allowed to run multiple jobs at once without any additional supporting setups like MIG or MPS. As in the first benchmark, up to twelve instances of the workload fit onto the GPU at the same time before it runs out of VRAM. The fourth scenario (MPS) is the same as the third, with the exception, that MPS is also enabled to better support the concurrent execution. The fifth and final setup (MIG) is a rerun of the one from the 2022 contribution, in which each GPU was split into the maximum number of independent GPU parts. As a result, a maximum of seven workload instances can be run concurrently per GPU. The results of the benchmark, including run time of the individual workloads, power usage, power efficiency and the number of concurrent benchmark instances on the machine, can be found in table 1.

It can be seen that the default setup is highly inefficient compared to all other setups, including the CPU one. All other GPU scenarios show better efficiency due to the higher number of concurrent instances when compared to the default case and the faster runtime when compared to the CPU case. The simple sharing scenario is less than half as efficient as both optimized cases with MPS or MIG.

4.3 HTCondor scaling benchmark

As a final benchmark before deployment, the same workload as the scaling benchmark from the 2022 contribution was repeated with additional benchmark scenarios. A total of 414 jobs with similar workloads to the ones of the above benchmarks were sent to run on the benchmark machine through an HTCondor batch system. The four scenarios examined here are the ones from the previous benchmark without the

Table 1: Results of the energy efficiency benchmark. Scenarios as explained in section 4.2. The energy per training is calculated as the power draw multiplied with the average runtime of one instance divided by the number of concurrent instances.

Benchmark scenarios	CPU	Default	Simple sharing	MPS	MIG
# of concurrent instances	64	8	96	96	56
Average runtime [min]	82.07	10.56	41.43	16.23	9.41
Total power draw [W]	664.80	748.22	1202.58	1396.29	1327.55
Energy per training [kJ]	51.95	59.25	31.15	14.17	13.39

Table 2: Results of the HTCondor scaling benchmark. Scenarios as explained in section 4.3. Total throughput given as epochs per time to account for a varying number of epochs per job.

Benchmark scenarios	Default	Simple sharing	MPS	MIG
Total Throughput [Epochs/s]	1.94	10.06	18.16	14.14

CPU case. It should be noted that the hardware resources are not as intensively utilized as in the bare metal benchmarks above. Some dead time between the end of one job exists before the next job starts and towards the tail end of the benchmark, fewer jobs are running at the same time. While the dead time between jobs is an expected feature when working with batch systems, the decreasing concurrency towards the end of the benchmark is not. A high throughput system is intended to run a consistent stream of jobs and as such, the batch system is not meant to run out of jobs in general. To take these discrepancies into consideration, a cut was applied to ensure that only the part of the benchmark is considered, during which the benchmark machine is fully utilized. The throughput shown in table 2 is in the form of epochs per second to account for an inherent randomness in the number of epochs that are necessary for the individual trainings to converge.

It can be observed that the default case, with only one job allowed per GPU, has the least throughput. The approach to share a GPU without actually deploying any supporting software or hardware features is a significant improvement. The best performance is shown by the MIG and to an even greater degree the MPS setup. The main difference in the performance between the benchmark scenarios is the number of available job slots. The throughput for an individual job is nearly equal between all setups except for the *simple sharing* one, as that one is bottle-necked by the time slicing behavior explained in section 3.

5 Deployment in production for an HTCondor batch system

The setup with MPS has been rolled out for a number of machines on the TOpAS cluster. The machines in use are not the same as during the described tests, with the main difference being that the setup was applied to three machines that utilize NVIDIA V100S GPUs instead of the A100 GPUs for the benchmark. The setup consist of an MPS daemon process running in the background, an HTCondor configuration to manage the resource allocation and a script to manage the supervision through MPS for the running jobs. The variable `CUDA_MPS_PIPE_DIRECTORY` of the MPS service is set to a non-default value in order to have MPS only affect processes that actively set this variable to the same value in their environments. All other processes that run on the GPU are unaffected by MPS and subject to the normal time slicing behavior. The configuration file sets up one partitionable slot and a VRAM pool for each of the eight physical GPUs. In addition, the existing GPUs are advertised eight times each and can therefore be assigned to up to eight different jobs. An incoming job with a requirement for a GPU and an amount of VRAM can be assigned the required resources from one of the partitionable slots. With this setup, jobs can be placed on an individual GPU as long as the required VRAM is available. This setup does not allow for multi-GPU jobs as it would require resources from multiple different partitionable slots, which is generally not supported. Finally, the script modifies the initial environmental settings to steer the supervising MPS service, including the `CUDA_MPS_PIPE_DIRECTORY` variable. In addition, it sets the `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT` variable, preventing processes from occupying more VRAM than requested.

This setup has been in deployment since May 2024 with no major issues. During peak usage, between four and five jobs have been executed on each of the available physical GPUs concurrently. This is a considerable improvement to the default setup, where only one job is allowed per GPU.

6 Comparison to MIG

Both MIG and MPS can provide significant improvements to the throughput of GPU jobs in a batch system, but they also have their respective up- and downsides. MPS is available for a number of graphic cards that do not support MIG, such as the NVIDIA V100 GPUs that are part of TOpAS. MPS is generally easier to set up and allows for greater flexibility due to its software based nature. This has allowed it to be included as part of the TOpAS batch system without complex changes to the underlying HTCondor installation. The combination of multiple processes through MPS complicates the distinction between jobs for the purpose of monitoring, as only a single MPS process is attributed all the work that is performed on the GPU. With some extra effort, assigned VRAM can still be monitored, but GPU utilization can only be roughly estimated. Although MPS offers tools to allocate hardware resources and enforce limits, these boundaries can sometimes be breached, resulting in overutilization until the policy violation is detected. Additionally, problematic processes can affect other processes supervised by MPS if the error is sufficiently severe to affect the entire device. Finally, GPU processes spawned by non-CUDA libraries are not supported by MPS, leading to errors or unexpected behavior.

Due to its hardware based nature, MIG is more difficult to set up and its flexibility is limited. In summary, this makes it difficult to use MIG dynamically in a batch system, as only the preset configuration of GPUs would be available. The size of those pre-configured GPU pieces might not fit the requirements of the requested jobs. On the other hand, this rigidity leads to great isolation between the individual GPU pieces and enables access to the monitoring data of each of them. CUDA-external GPU libraries show the same behavior as with full GPUs, as the supervision is not directly CUDA related, making it safer to use with those libraries than the CUDA based MPS.

TOpAS provides a good testing ground for comparisons of this nature, and has lead to additional insights concerning the usage of different GPU models. The A100 GPUs are usually requested for larger scale applications that require the full VRAM available, while the smaller V100 GPUs tend to be occupied by small scale jobs that only require few GPU resources. As such, it is counterproductive to remove A100 GPU resources from the batch system to split them into smaller GPUs, as long as the V100 GPUs could take on those jobs. As MIG is not available for the V100 GPUs, the only way to improve their utilization is through MPS. The variety of workflows with a wide array of hardware requirements should be taken into consideration when acquiring hardware for future HEP compute cluster projects.

Over all, MIG would be the more suitable solution in many cases due to the higher degree of security, but the requirement of newer hardware limits the technology significantly. Even for hardware that is capable of using it, the complex and rigid setup currently makes it difficult to use in a batch system environment.

7 Conclusion

The Multi-Process Service (MPS) allows for multiple CUDA contexts to be fused into a single one. Due to the limitations of the default time slicing scheduling, this allows for greater performance when compared with only placing multiple contexts on the same NVIDIA GPU.

The performance of MPS was benchmarked on bare metal and in the context of an HTCondor batch system, using the TOpAS cluster. The viability of its use as part of a batch system was tested and the benefits and risks have been described. The benchmarks have shown that in an ideal scenario, the addition of MPS can lead to speedups of close to five when compared to the non MPS scenario. The benchmark scenario with MPS has shown itself to be more than twice as efficient as the one without MPS, and closely behind the MIG approach. When tested in the context of a real batch system, MPS has shown a speedup over the current default usage of more than nine, although a simpler approach of sharing that does not involve MPS has also shown a speedup of five over the default non-sharing approach. In addition, the findings from the 2022 ACAT contribution were reproduced.

The introduction of MPS into the TOpAS batch system has led to an improved throughput of four to five for those nodes during peak usage, and the setup has been described. The technologies MPS and MIG have been compared and the topics of performance, security, ease of use and flexibility have been covered.

References

[1] Evans L, Bryant P. *LHC Machine.* Journal of Instrumentation. 2008 aug;3(08):S08001. Available from: `https://dx.doi.org/10.1088/1748-0221/3/08/S08001`.

[2] The CMS Collaboration. *The CMS experiment at the CERN LHC.* Journal of Instrumentation. 2008 aug;3(08):S08004. Available from: `https://dx.doi.org/10.1088/1748-0221/3/08/S08004`.

[3] CMS Offline Software and Computing. *CMS Phase-2 Computing Model: Update Document*. Geneva: CERN; 2022. Available from: `https://cds.cern.ch/record/2815292`.

[4] Caspart, René, Fischer, Max, Giffels, Manuel, von Cube, Ralf Florian, Heidecker, Christoph, Kuehn, Eileen, et al. *Setup and commissioning of a high-throughput analysis cluster.* EPJ Web Conf. 2020;245:07007. Available from: `https://doi.org/10.1051/epjconf/202024507007`.

[5] NVIDIA Corporation. *NVIDIA Tesla V100 GPU Architecture*;. [Internet]; 2017. WP-08608-001_V1.1. [Accessed 22th July 2024]. Available from: `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

[6] NVIDIA Corporation. *NVIDIA A100 Tensor Core GPU Architecture*;. [Internet]; 2020. [Accessed 22th July 2024]. Available from: `https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf`.

[7] Voigtländer T, Giffels M, Quast G, Schnepf M, Wolf R. *Optimized GPU usage in high energy physics applications [Manuscript submitted for publication].* Journal of Physics: Conference Series. 2024.

[8] NVIDIA Corporation. *NVIDIA Multi-Instance GPU*;. `https://www.nvidia.com/en-us/technologies/multi-instance-gpu`. [Accessed 22th July 2024].

[9] NVIDIA Corporation. *Multi-Process Service*;. `https://docs.nvidia.com/deploy/mps/index.html`. [Accessed 22th July 2024].

[10] NVIDIA Corporation. *CUDA Toolkit*;. `https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html`. [Accessed 22th July 2024].

[11] HTCondor Team. *HTCondor.* Zenodo; 2024. Available from: `https://doi.org/10.5281/zenodo.11397217`.

[12] Stone JE, Gohara D, Shi G. *OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems.* Computing in Science & Engineering. 2010;12(3):66-73. Available from: `https://doi.org/10.1109/mcse.2010.69`.

[13] TensorFlow Developers. *TensorFlow.* Zenodo; 2024. Available from: `https://doi.org/10.5281/zenodo.12726004`.

[14] Laurie D, Styblik Z, Amelkin A. *ipmitool*; 2023. [Software]. Github. `https://github.com/ipmitool/ipmitool`.

[15] NVIDIA Corporation. *System Management Interface SMI*;. `https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf`. [Accessed 22th July 2024].