INSTITUTE OF INFORMATION
AND COMMUNICATION TECHNOLOGIES
BULGARIAN ACADEMY OF SCIENCES
CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **26**, No 2
Sofia • 2026 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2026-0022

Supercomputer HEMUS: Benchmark Results and Performance Optimization

Emanouil Atanassov[1,2], *Mariya Durchova*[1], *Sofiya Ivanovska*[1,2], *Aneta Karaivanova*[1], *Aleksandar Kirilov*[1,2]

[1]*Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Acad. G. Bonchev str. 2, Bulgaria*
[2]*Centre of Excellence in Informatics and Information and Communication Technologies, 1113 Sofia, Bulgaria*
E-mails: *emanouil@parallel.bas.bg mabs@parallel.bas.bg sofia@parallel.bas.bg anet@parallel.bas.bg alex.kirilov@acad.bg*

***Abstract*:** *Supercomputers and advanced High-Performance Computing* (*HPC*) *systems are needed to address large scientific problems, AI models training or industrial/engineering tasks. One of the important steps during the installation, configuration and tuning stages, is to perform comprehensive benchmarking and obtain insights about the best ways of exploiting the systems capabilities. HEMUS is modern petascale heterogeneous HPC system. Benchmarking insights and optimization strategies from HEMUS are applicable to a broad class of contemporary HPC systems. In this paper we present results from general benchmarks, together with optimization strategies applied and the obtained conclusions. We also present benchmarks related to software simulation of quantum computing algorithms, as well as simulation using low-discrepancy sequences, as they cover a significant portion of the expected workload on the system. The results highlight the importance of careful tuning and optimization, demonstrating that significant performance gains can be achieved, with the proposed approaches broadly applicable across comparable HPC systems and workloads.*

***Keywords*:** *Supercomputing, High-Performance Computing* (*HPC*)*, Quantum simulators, MPI, Monte Carlo methods, Quasi-Monte Carlo methods.*

1. Introduction

Benchmarking is an important step within the system deployment process, as well as in the deployment of individual applications and libraries [2, 4, 11, 21]. Supercomputers are expensive, complex and specialized systems, whose main purpose is to perform large scientific, engineering and computational tasks in minimal time. Benchmarking is critical in the field of High-Performance Computing (HPC), as it provides an objective method for measuring, comparing and optimizing

their performance, taking into account requirements from users, user groups and applications. Comprehensive benchmarking is a necessary step in the overall process of installing and configuring a system, its fine-tuning and opening to the end-users.

If we consider the benchmarks to be a representative example or proxy for certain application domains, it is natural to attach high importance to widely used general benchmarks that stress multiple subsystems, and to try and extract as much as possible expertise and know-how that can be leveraged during the actual exploitation of the supercomputer [18].

In the next section we describe in detail the hardware and software organization of the supercomputer HEMUS, which is our target system.

The most famous benchmark in the field of supercomputing is the LINPACK [6], which measures the rate of floating-point operations that a supercomputer achieves while solving a dense system of linear equations using a certain algorithm. The results of LINPACK are used to compile the TOP500 global ranking - the main standard for ranking supercomputers [20].

Our approach in performing the LINPACK test with the goal to achieve the best results possible on our hardware is described in Section 2.1, where we also outline lessons learned and guidelines that are applicable for most of the applications developed or run on HEMUS.

Other benchmarks that stress different capabilities of the supercomputer and offer insight into its behavior under workloads that stress capabilities distinct from the pure double precision computation rate are presented in Section 2.2 and 2.3.

HEMUS has been designed and implemented as a multi-purpose system [1]. As such, it also allows for simulating quantum computing algorithms, following of course the usual restrictions and caveats. Due to the inherent complexity of the task, it is important to have a straightforward approach to evaluate and optimize different software implementations, capable of simulating quantum computing algorithms. Benchmarking results are presented and discussed in Section 3 [9, 10], with the goal of offering a recipe for optimizing this process on an HPC system of comparable capacity. Energy efficiency is an important part of this evaluation.

The conclusions from these benchmarks have wide applicability and cover most of the potential applications. However, our specific expertise in the domain of Monte Carlo and quasi-Monte Carlo methods calls for testing that can reveal useful optimization approaches and constraints when running such kinds of applications. In Section 4 we present and analyze tests which offer insight into possible issues and bottlenecks in this area.

As the architecture of HEMUS follows industry trends and thus is fairly popular, in the last section we try to summarize our experience with orientation towards potential users of similar HPC systems. We also describe problems and directions for future work, which naturally stem from the presented research.

2. Supercomputer HEMUS

IICT operates large and complex IT infrastructure, comprising two supercomputers, built on state-of-the-art hardware and software technologies, as well as a data storage

and management system. The infrastructure is deployed within a data center, equipped with the usual Uninterruptable Power Supplies (UPS), diesel-generator, Building Management System (BMS), etc.

It has a data storage subsystem with capacity of 6.72 petabytes (unformatted), which manages and serves data for multiple user groups, using the available InfiniBand links to the two supercomputers. The flagship system is the supercomputer HEMUS with petascale performance, while the older Avitohol supercomputer plays a supporting role. The typical data-management services like backup, metadata handling, version management, are provided, with the most performance-critical service being the parallel filesystem support.

HEMUS was acquired as part of the infrastructure procurement tasks of project BG05M2OP001-1.001-0003, financed by the Science and Education for Smart Growth Operational Programme and co-financed by the European Union through the European Structural and Investment funds. It was delivered in 2023 and represents the newest unifying high-performance computing component of the integrated infrastructure complex of IICT-BAS. Each of the three parts – two supercomputers and the system for storing and processing petabytes of data are organized to work both separately and in combined mode. The network interconnect between them is based on InfiniBand, with each supercomputer having a fully non-blocking topology. Uninterruptable operation of the facility in 24/7-hour mode is ensured through multiple UPSs and, in the last year, by a diesel generator. For easier detection, diagnosis and pro-active management of problems occurring in the server room, in accordance with the vendors-based specifications for operating the various hardware and service components, a building management system has been configured. The automatic control, carried out through sensors for temperature, humidity, water leak detections in the liquid cooling system, generates alarms of varying levels of importance.

From a hardware perspective, HEMUS is composed of two subsystems – CPU-based partition and GPU-accelerated partition, which can be used independently and jointly. The CPU-based partition consists of 128 HPE ProLiant XL220n Gen10 Plus servers as a compute node, situated in two server racks within 16 server chassis, each of which accommodates four servers. Each compute node is equipped with two CPUs Intel Xeon-Platinum 8352Y 2.2 GHz 32 cores / 64 HT (64 cores / 128 HT per node) with direct liquid cooling, 256 GB DDR4-3200 MHz RAM and two storage devices 480 GB NVMe Gen3 SSD type. The interconnectivity between each of them is based on non-blocking NDR InfiniBand with 200 Gbps bandwidth, regardless of their rack placement.

The GPU-accelerated partition consists of 20 HPE ProLiant XL675d Gen10 Plus servers with direct liquid cooling of the processors and the graphics accelerators, situated in four server racks, each of which accommodates five servers. Each compute node is equipped with eight NVIDIA A100 TENSOR CORE 40GB SXM GPUs with 64-bit floating point pick performance 19.5 TFLOPS, memory bandwidth 1555 GB/s and 600 GB/s NVLink interconnectivity within the node [12]. The other components are two AMD EPYC 7742 2.25 GHz 64 cores / 128 HT (128 cores / 256 HT per 1 node), 512 GB DDR4-3200 MHz RAM and storage devices as follows: two 480 GB

SATA SSD type and four 3.84 TB NVMe Gen3 SFF SC U.3 SSD type, offering in total approximately 16.30 TB per 1 node. The interconnectivity between each of them is based on non-blocking HDR InfiniBand with 200 Gbps bandwidth.

The system for storing and processing petabytes of data is capable to operate with 6.72 petabytes of data, sharing them between the supercomputers with low latency external connectivity, based on InfiniBand 200 Gbps and ethernet 10 Gbps interfaces to HEMUS and InfiniBand 100 Gbps and ethernet 1 Gbps to Avitohol respectively. The system is built through 8 data servers, each with capacity to process with up to 24 TB of operating memory. The data located in the system is accessible through five double-redundant management servers with Lustre and HPE Ezmeral, IBM Storage Scale (GPFS) file systems to meet the diverse requirements of the scientific user communities, as well as make use of the heterogeneous hardware.

The operating systems installed on the whole hardware complex are intended to facilitate easy administration while supporting wide range of use-cases. Avitohol and on the storage system are based on Red Hat and HEMUS operates through SUSE HPC Linux. For the specific needs of users, various software and packages such as Intel oneAPI – for massively parallel applications, NVIDIA CUDA – for machine learning tasks or intensive computations, and other have been purchased and installed, enabling and facilitating high-performance computing across the entire infrastructure.

2.1. Results from LINPACK test and discussion

The **LINPACK benchmark**, originally introduced by Dongarra, Luszczek and Petitet [6], remains the most popular tool for ranking supercomputers, while it offers interesting insights when evaluating any type of High-Performance Computing (HPC) system, even individual workstations [8]. A parallel version of the LINPACK benchmark, including instructions for execution, is available at: **http://www.netlib.org/benchmark/hpl/**. However, due to the importance of this test in deciding which systems can enter the TOP500 list of supercomputers [20], hardware vendors strive to produce optimized applications that can extract every last bit of performance of the system. LINPACK is mostly oriented towards measuring the raw compute power and provides a good stress test for the whole system. Since the systems in the TOP500 list are typically homogeneous, we had to select the GPU-based partition of HEMUS as the one that would run the benchmark. Because most of the raw computational power comes from the NVIDIA A100 accelerators, we decided to use the optimized executable, provided by NVIDIA in an enroot-based container.

By observing the way, they have packaged the executable and testing some of the subcomponents, like the MPI library, we concluded that using this container provides obvious performance benefits, compared to using the libraries that came with the operating system. This observation seemed to hold in many other areas, and thus we draw the conclusion that for many workloads it is advisable to run them inside such containers, for example starting from the NVIDIA HPC container or NVIDIA machine learning container [16].

The GPU-based partition of HEMUS has 20 servers with a total of 160 GPUs. The LINPACK executable required their placement in a grid. Our previous experience when running these tests on the Avitohol supercomputer has been that the grid should be as square as possible. As Avitohol has 150 servers, this means that a 12×12 configuration is preferable. We even obtained higher results using 144 servers in a square grid instead of using all the 150 servers in a 15×10 grid. Substantially different situation was observed on HEMUS, where the best configuration was the natural placement in an 8×20 grid, which achieved 2.51 PFLOPS, compared to 2.16 PFLOPS achieved via 16×10 grid. Since 8 GPUs are on the same server and thus can communicate much faster, using the NVLink interconnect, it is logical for them to be grouped together. We concluded that such an arrangement is advisable for applications that need to organize their GPU-to-GPU communications in a hierarchically structured way.

Many other parameters can be tuned in the Linpack input file, but they are specific to the task of solving a dense linear system using Gaussian elimination. We tried to maximize the use of available RAM by increasing the matrix size, as this gives the best scalability, and we sampled several combinations of the other parameters, as our time for achieving the test results was limited. We were also able to employ the CPUs to perform a small percentage of the computation, since such an option was present. This approach was useful in the case of LINPACK, as it allowed us to cross the 2.5 PFLOPS limit, but is not advisable in general, since it increases the complexity of the software substantially for a limited gain.

Table 1. LINPACK performance of the systems compared with their theoretical performance

Supercomputer	LINPACK	Theoretical	Efficiency
Avitohol	0.26	0.41	63%
Discoverer	4.52	5.94	76%
HEMUS	2.53	3.21	79%

Table 1 summarizes the LINPACK performance of three Bulgarian supercomputers: Avitohol, Discoverer, and HEMUS. An efficiency metric, calculated as the ratio between the achieved LINPACK performance and the system's theoretical peak performance, is also shown. It is notable that the HEMUS system achieves the highest efficiency. In terms of energy efficiency, HEMUS achieved over 2.5 PFLOP/s with a lower overall power draw compared to Avitohol, which is understandable given that Avitohol uses substantially older technology. The main advantage of HEMUS are the more efficient GPU accelerators. The execution achieved workload balancing, making full use of all available GPUs, with the CPUs providing mostly coordination. This approach is particularly necessary in heterogeneous systems. During the LINPACK tests that used nearly all the available RAM, the power usage of each of the GPUs was nearly 400W. This is an important metric to observe, as we have seen many applications that achieve 100% load of the GPU (as reported by the nvidia-smi tool) but are far from maximum energy usage, thus are not fully utilizing the capabilities of the hardware.

Although the LINPACK test is not universal, its importance has increased lately due to the prevalence of AI-based workloads, where efficiency of matrix operations drives the performance. It is obvious that GPU-enabled systems, when properly

configured, outperform traditional CPU-based systems not only in raw performance but also in efficiency and energy consumption.

2.2. HPCG Test on HEMUS

The LINPACK test gives a good idea of the maximum performance that a supercomputer can provide, but many workloads stress memory bandwidth or network bandwidth and especially network latency in a different way. The perceived need for a widely recognized benchmark that would provide different view of the system's capabilities has been addressed via the introduction of the High-Performance Conjugate Gradient (HPCG) benchmark [7]. Unlike LINPACK, which emphasizes peak floating-point performance, HPCG mostly performs sparse linear algebra operations, which are common in many real-world scientific and engineering applications and involve lots of communications with memory. HPCG has limited configurability, compared to LINPACK, which makes performance tuning more challenging. Since performance is measured in the same way as in LINPACK, i.e., in number of floating-point operations per second, one can see that the reported speeds in HPCG are many times lower than those for LINPACK. Perhaps this disheartening observation has led to less systems submitting results for HPCG in the public rankings.

After we submitted our results, HEMUS was ranked 108th in the HPCG list, with a measured performance of 36.42 TFLOP/s. This result is considerably lower than the LINPACK result, not only because of the accent on memory operations but also because the GPUs that provide most of its compute power are more suited for dense matrix operations.

As it is the norm that computational power increases faster than memory bandwidth, it is difficult to achieve high rates of floating-point operations per second in memory-bound workloads. In order to better accommodate for the needs of such applications, HEMUS has been designed with a non-blocking InfiniBand interconnection, which ensures that the network bandwidth is fully available to applications, without bottlenecks happening at the switches. For application developers it is advisable to perform careful evaluation of the various kinds of memory transfers and interactions between the different levels of memory and cache, in order to mitigate the architectural limitations in the current systems. Software prefetching can help, but is not always beneficial so it is advisable to benchmark on critical use cases and determine if it is useful.

2.3. Tests of mixed precision performance and micro-benchmarks

The HPL-MxP (Mixed Precision) benchmark, available at **https://hpl-mxp.org** [5] tests the performance of the system when using mixed precision operations. Algorithms that work with varying precisions for the different operations and data are de-facto standard in the domain of AI training and inference. The advantage of these algorithms is that usually the computing elements (CPU, GPU or others) have much higher performance when using operations with less precision.

The test itself uses a technique that combines most of the computations performed at lower precision (e.g., single precision), with high-precision correction

steps (refinement). Unlike the LINPACK standard of always using double-precision (FP64) floating-point operations, HPL-MxP attempts to emulate scenarios where lower precision operations dominate, although the underlying test problem is similar to LINPACK. In our evaluation, we used the NVIDIA-provided HPL-AI binary, which uses mixed precision and is optimized for GPUs like the A100, available at HEMUS.

Two configurations were tested and the results were as follows:

Table 2. Execution time and performance of HPL-MxP with and without refinement

Algorithm	Time (s)	Performance (PFLOP/s)
Without refinement	9.12	15.37
With refinement	19.76	10.52

One can see that the initial part of the algorithm, which was fully performed in low precision, results in much higher rates of operations, compared with the LINPACK numbers from the previous section. The refinement part results in slower overall rates, but we still see an overall performance increase of approximately four times.

The NVIDIA A100 GPU has particular advantages in AI-centric mixed-precision workloads, with 78 TFLOPS in Half Precision (FP16) and 39 TFLOPS in Brain Floating Point (BF16). The results serve as motivation for careful staging of operations and even developing mixed-precision algorithms for tasks that are outside of the AI domain.

In addition to the benchmarks described above, we also conducted multiple additional performance tests, usually called micro-benchmarks because they stress some particular features of the system. The communication performance was tested via the OSU Micro-Benchmarks (OMB) suite, with our emphasis being on the point-to-point latency and bandwidth tests [14]. These benchmarks measure the efficiency of the MPI communication layer over different message sizes between two processes. The OSU MPI Latency Test is a very popular micro-benchmark that measures the round-trip communication time for messages of varying sizes. One can see in Fig. 1, how the latency remains relatively low for small message sizes ($\leq$ 1 KB), ranging from approximately 0.5 μs for 1-byte messages to around 1.9 μs for 4 KB messages. Afterwards, latency increases gradually with the increase in message size, reaching 414 μs at the largest tested size of 4 MB. In order to improve this numbers, we tested multiple variants of the MPI library and communication options.

The other side of the communication problem relates to bandwidth. The most representative micro-benchmark test is the OSU MPI Bandwidth Test, which evaluates the data transfer rates between two processes residing on two physical servers. As shown in Fig. 2, the bandwidth increases sharply with message size, peaking at approximately 14.3 GB/s (14293 MB/s) for messages around 128 KB. Beyond this point, the bandwidth stabilizes in the range of 11-12 GB/s, even as the message size continues to increase. This issue is an artefact of the particular way the MPI calls are implemented and may be alleviated by switching to other MPI library or changing some of the runtime options. Such a tuning process should always be performed for an application where a benchmarking task can be identified successfully.

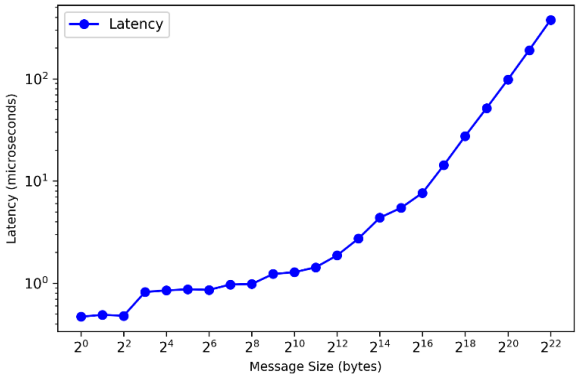

Fig. 1. OSU MPI Latency benchmark

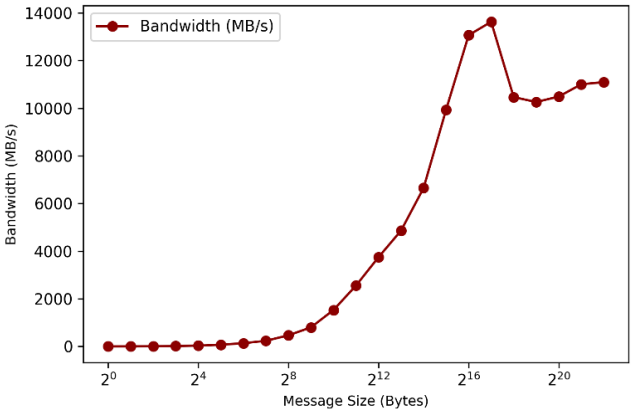

Fig. 2. OSU MPI bandwidth benchmark

Another popular micro-benchmark, this time related to memory bandwidth, is the STREAM benchmark [15]. It measures typical memory access patterns, highlighting four main operation types: Copy, Scale, Add and Triad. As these operations are obviously memory-bound rather than compute-bound, this benchmark provides insight about the potential for efficient execution of memory-bound workloads.

On HEMUS we used the implementation NVIDIA-STREAM (Version 25.4.0). The STREAM benchmark was executed on the GPU using FP32 (single precision) operations. When varying the number of elements in the tested arrays from 2^{23} up to 2^{28}, the benchmark reports the average memory transfer rate (in MB/s) for all four operations (Fig. 3).

The results show that in all types of operations we see some initial increase in performance as the array sizes grow, until a saturation point is reached. Triad consistently achieves the highest memory bandwidth, but the difference is more pronounced for the smaller array sizes. For all operations, the peak performance reaches approximately 1.3 up to 1.4 TB/s. For smaller array sizes ($< 2^{24}$), performance is much lower due to cache effects and insufficient workload to fully utilize the

memory subsystem capabilities. Beyond 2^{26} elements, the performance tends to plateau, indicating that the maximum sustainable bandwidth of the system is reached.

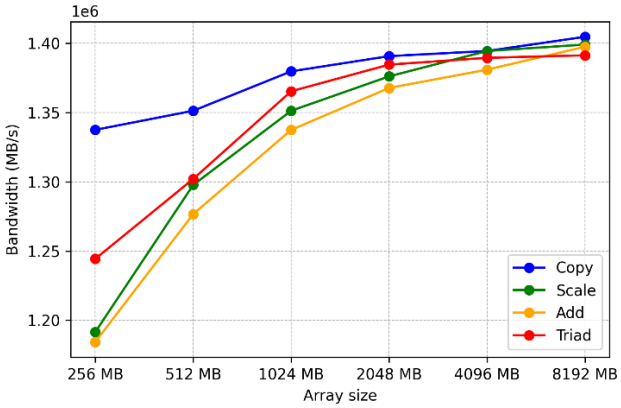

Fig. 3. STREAM bandwidth results (in MB/s) for different FP32 operations on the NVIDIA A100

The STREAM benchmark results are mostly an indication of the systems architecture limitations. On paper the RAM and GPU RAM bandwidths are high, but when multicore applications are run, the memory easily becomes a bottleneck. That is why one is expected to take advantage of the possibility for asynchronous operations and try and interleave communications and computations. It is also notable that the architecture of HEMUS has been optimized from point of view of GPU-to-GPU communications, providing NVLink for intranode and InfiniBand (with one port for each GPU card on each server) for internode data transfers. The idea of this arrangement has been to enable efficient execution of applications that span multiple nodes with high amounts of data transfers. Developers are expected to prioritize the use GPU-to-GPU data transfers even between different nodes as way to reduce bottlenecks and maximize the use of the available hardware resources.

3. Quantum computing simulations

Quantum computing has had a series of breakthrough discoveries and demonstrations that make it increasingly probable that quantum computing devices with capabilities to solve practical problems will become available in the next 5 to 10 years [3]. In order for this to become a reality, it is important to solve the noise problem and also to scale the implementations to much larger number of qubits. The software simulation of the execution of quantum computing algorithms is an intermediate approach, which enables certain quantum computing algorithms to be tested successfully on classic hardware. Despite its limitations, it has advantages in terms of availability, as most HPC systems can perform such kinds of simulations. The main limitation seems to be the total available memory of the system, having in mind that required memory scales exponentially in the number of qubits. Since the workload is also computationally heavy, it is advisable to have also GPU accelerators, as in the GPU-enabled partition of HEMUS. Since it is possible to distribute

computations and data (statevectors) across servers, one can successfully use HEMUS to simulate up to 40 qubits using the statevector method.

Quantum algorithms can be successfully implemented in python scripts using Qiskit toolkit from IBM [19]. When statevector simulation is to be used, one can employ the so-called Aer simulator, enabling GPU acceleration or even MPI for cross-server runs. Python modules like psutil, and pynvml can be used to monitor hardware load during quantum simulations. In this way one gets simultaneous measurement of CPU, RAM, and GPU utilization, as well as energy consumption, while running Quantum Volume tests with varying numbers of qubits and depths. System metrics can be collected in real time through multi-threaded monitoring and subsequently statistically analyzed. The obtained data provide an objective assessment of the performance and energy efficiency of GPU-based quantum simulations. Data collection is implemented by a parallel stream (thread), which periodically reports values with an interval of 0.05 s. After the simulation is completed, the average and maximum values are statistically calculated and recorded in a structured format.

For our testing, quantum circuits were generated using the QuantumVolume class from qiskit.circuit.library, which is a stochastic test circuit with high complexity [2]. Simulations are performed using AerSimulator with the statevector method and the device = "GPU" parameter, which ensures the use of the graphics accelerator for linear algebra operations.

For each combination of parameters – number of qubits n (n = 32-34), depth of the circuit d (d = 20, 200, 1000) and number of shots (shots = 100-10,000) a separate execution is performed. During each simulation, the parallel stream records the system metrics, which are then summarized. The energy consumption $E = P_{\text{avg}}t$ is also calculated, where P_{avg} is the average power, and t is the simulation time.

The obtained experimental results show a pronounced exponential dependence of the simulation time and the load on the computational resources relative to the number of qubits, which is in accordance with the theoretical complexity of statevector simulations.

With small numbers of qubits (up to 8-10), the simulations are performed almost instantly, with minimal load on the CPU and GPU. However, with an increase in the number of qubits above 12-14, a sharp increase in the execution time (up to several seconds and minutes) and a progressive increase in GPU activity are observed. Simulations with over 20 qubits show significant load on hardware, while GPU activity increases to 100%, which indicates that the computational burden is transferred mainly to the GPU, which begins to be used to its full potential.

Fig. 4 shows the relationship between the average GPU utilization and the number of qubits at a fixed quantum scheme depth d (d = 200) and different number of shots (shots = 100, 1000, 10,000, 20,000).

The graph clearly demonstrates three modes of behavior.

1. Low load (2-20 qubits). In this range, the GPU load remains below 10%, regardless of the number of shots. This indicates that the computational tasks are too small to fully utilize the GPU's compute power, and the simulation is mostly performed on the CPU.

2. Transition zone (22-26 qubits). After about 22 qubits, the GPU load begins to increase sharply. This transition corresponds to a moment when the size of the statevector 2^n becomes large enough to fully engage the GPUs resources.

3. Full activation (28-34 qubits). At more than 28 qubits, the GPU load reaches between 80% and 100%, which means that the simulator uses the graphics accelerator to the maximum. The curves for different numbers of shots merge, indicating that the number of shots have little impact on GPU utilization – the main factor is the number of qubits (the size of the quantum state). This is to be expected because of the type of simulation used.

The figure confirms that the GPU workload grows following the exponential complexity of the statevector simulations.

The transition point (~22-24 qubits) can be considered as the efficiency boundary of GPU acceleration, at which the computations switch from CPU-dominated to GPU-dominated mode.

At high dimensionality (above 30 qubits) the GPU reaches full utilization, which indicates that the hardware is being used optimally, but also that further increase in qubits will require scaling to more GPU devices or even using multiple servers (at more than 35-36 qubits).

Examining the graph of the simulation, we obtain the following approximation:

$$f(x) = a \times \exp(b \times x),$$

which shows exponential depency. The obtained parameters are a=0.0007 and b=0.5632, with R-square of 0.9789 and RMSE = 0.9409.

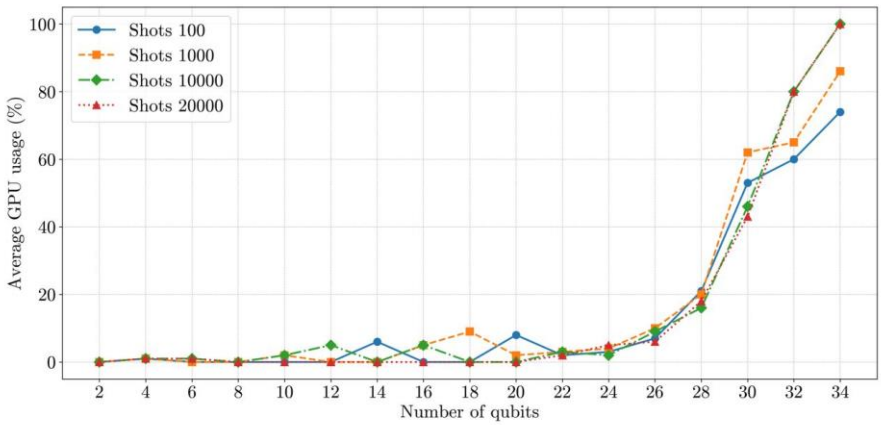

Fig. 4. Average GPU usage (%) for the different qubits

The simulation times needed for varying number of shots and number of qubits in the range of 2 to 34 are shown on Fig. 5. Up to approximately 20-22 qubits, the execution times are low. The simulations run with low CPU/GPU usage. After passing the 24 qubit limit, we observe an increase in the simulation times.

When reaching 30-34 qubits, the simulation times increase rapidly, while the use the system resources reaches almost full capacity.

There is another dependence that is the subject of our study. The dependence between the average GPU power consumption and the size of the quantum register, measured by the number of qubits, at different values of the number of measurements (shots). This dependence is key for estimating the computational complexity and hardware requirements in classical simulation of quantum circuits. In Fig. 6 it can be seen that in the range up to approximately 26 qubits the GPU power remains almost constant, varying in a narrow interval around 100-120 W. The computational operations remain within the cache and memory limitations of the GPU. At 28 qubits a distinct transition is observed, characterized by a sharp increase In GPU power. In the range of 30-34 qubits GPU power reaches values between 230 and 260 W, which is close to the maximum capacity of modern graphics accelerators.

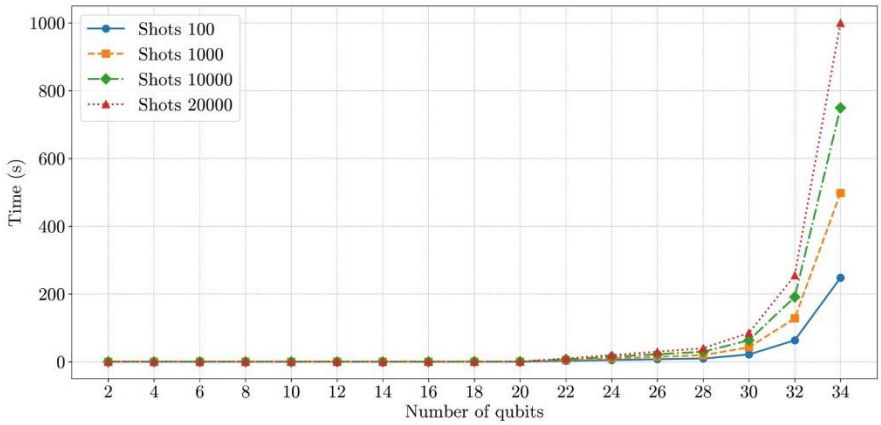

Fig. 5. Simulation time for the different qubits

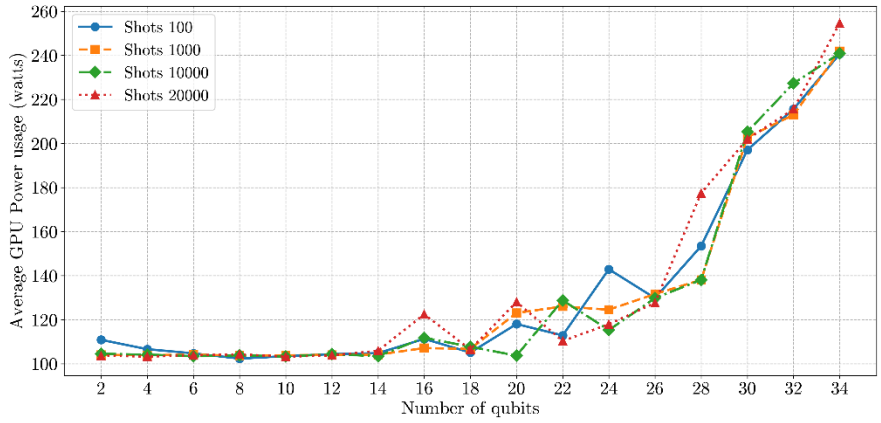

Fig. 6. Average GPU usage (watts) for the different qubits

4. On the performance of a class of quasi-Monte Carlo algorithms on diverse HPC Systems

Monte Carlo algorithms form an essential part of the workloads run at supercomputers and advanced HPC systems [17]. They have inherent scalability advantage, as they usually have obvious approaches to parallelisation and great amount of independency between the different threads of execution. Because of the relatively modest requirements for memory and network bandwidth they successfully leverage the available compute power to increase their accuracy. They are also natural candidates for porting to GPU-enabled systems, although some care is required when different threads have diverging execution times, since the current generation of GPUs make substantial use of the SIMD model of parallel execution.
Monte Carlo algorithms usually employ random number generators in order to sample the needed random variables. Although there are physical devices that can produce such numbers, the performance requirements of HPC simulations lead to strong preference for software implementations, i.e., pseudo-random number generators. There is an important trade-off to be considered, between quality of the generator (in terms of how different it is from "real" random numbers), and speed of generation. As the generation of the pseudorandom numbers takes only part of the execution, this trade-off is to be evaluated by the developers. In cases when fast generation is to be prioritized, simpler generators are to be preferred. In the case of HEMUS it is advisable to use the random number generators provided as part of the curand library, which is standard part of the CUDA SDK deployment [13]. One can also use custom implementations for comparison or slightly better performance.

Another approach for increasing the accuracy of Monte Carlo algorithm is based on the use of specially designed, so-called low-discrepancy sequences, which can be used instead of pseudo-random numbers and may achieve faster error convergence, depending on the algorithm. In such cases we speak of quasi-Monte Carlo methods and algorithms. The Sobol sequences are perhaps the most popular low-discrepancy sequences, widely used in Financial Mathematics for example. A standard implementation of these sequences is also available in the curand library. Although the Sobol sequences have relatively straightforward definition and efficient software implementation, it is well known that they achieve slower generation speeds, measured in coordinates per second. An additional process, called scrambling, is frequently used in order to provide error estimation or even, in the case of the so-called Owen scrambling, to improve the theoretical convergence rates. Unfortunately, scrambling can lead to slower generation speeds. Our benchmarking efforts were thus oriented towards comparing library and custom implementations of pseudorandom generators and Sobol sequences, with and without scrambling.

In Fig. 7 one can see the impact of number of threads on the speed of generation (measured in millions of numbers of points per second). It seems that there is some register pressure, leading to 64 and 128 threads being fastest for all algorithms. As the number of threads may be dictated by other operations in the algorithm, we should be content with seeing good performance in the range of 64 to 512 threads.

The next table presents the generation speeds for the different algorithms, when number of points to be generated is varied. One can see that in some cases the custom implementations of pseudorandom generators can outperform the library implementations, although the difference is small and will be relevant in very rare cases. The Sobol sequences appear to be significantly slower, which means that their use is justified when number of points used is high enough to produce better errors, or when the cost of generation is amortized due to many other complex calculations in the algorithm. Owen scrambling also slows down generation but remains as an option. Simple XOR-based scrambling is practically without additional cost, so it is advisable to always use it, even just for the fact that it will avoid problematic values like when coordinates are all zeroes or all 0.5.

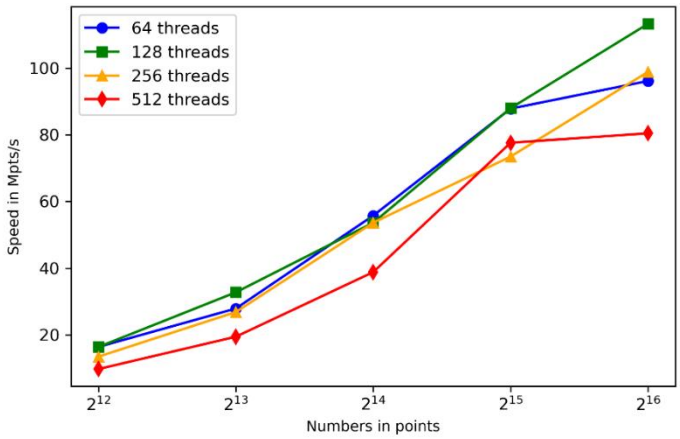

Fig. 7. Speeds of generation of the same algorithm (random number generator Philox32 from curand library), with varying number of points, dimension 4096 and different number of threads in the kernel launch

Table 3. Speeds of generation when using different algorithms (random, Sobol with and without scrambling, library or custom), with varying number of points and dimension 4096

Generator/Number of points	4096	8192	16,384	32,768	65,536
Sobol, custom	0.71	1.24	2.43	4.38	8.65
Sobol, custom with Owen scrambling	0.11	0.21	0.42	0.68	0.98
Sobol, cuRAND	1.24	1.24	1.24	1.24	1.24
Philox32 (custom RNG)	16.35	32.74	53.63	87.99	113.27
cuRAND Philox	27.78	55.56	79.87	113.07	147.69

In conclusion the generation of Sobol sequence and implementing scrambling can benefit from custom implementation, but for most use cases the library-provided versions are suitable to use. When generation is critical for the performance pseudo-random number generations provide better trade-offs, while users that need best accuracy should consider the use of low-discrepancy sequences, even the Owen-scrambled Sobol sequences. Indeed, for sufficiently high number of points, the speed of generation of the highly optimized custom implementation of the Owen scrambling achieves similar speeds like the library implementation without any scrambling.

5. Conclusion and directions for future work

This study presented a comprehensive benchmarking and performance analysis of the HEMUS supercomputer, put in context with the older Bulgarian supercomputers like Avitohol. Each of the benchmarks could be considered as representative of different types of workloads and thus the potential users or application developers could take hints towards performing their own optimization or tuning. Because HEMUS is built on a popular generic platform (NVIDIA DGX for the GPU-based partition), our observations and suggestions are applicable on a wide range of similar systems. Although some of the benchmarks revealed certain architectural limitations and potential bottlenecks, overall, the system appears to be balanced in terms of computational, memory, network capabilities and can execute diverse workloads from different scientific or industrial areas.

The two main considerations before developers or users of HEMUS should be to make full use of the GPU-enabled servers, as they provide huge compute capabilities, and also to fine-tune the memory and network accesses to avoid bottlenecks. The mixed-precision computations are a must for AI tuning/inference tasks and offer significant performance gains for developers that can introduce mixed-precision algorithms as part of their workflows. Once more data about usage patterns is gathered, it will be interesting to be able to map the performance of the applications to relevant benchmarking results.

***Acknowledgments*:** The work was partially supported by the Centre of Excellence in Informatics and ICT under the Grant No BG16RFPR002-1.014-0018, financed by the Research, Innovation and Digitalization for Smart Transformation Programme 2021-2027 and co-financed by the European Union. The research that led to these results was carried out using the infrastructure purchased under the National Roadmap for RI, financially coordinated by the MES of the Republic of Bulgaria (Grant No D01-98/26.06.2025).

References

1. Atanassov, E., A. Karaivanova, A. Kirilov, S. Yordanov. Data Services at IICT HPC Systems. – Digital Presentation and Preservation of Cultural and Scientific Heritage, Vol. **15**, 2025, pp. 323-330. DOI: 10.55630/dipp.2025.15.31.
2. Boehme, C., L. van Niekerk, D. Kumar, A. K. Sharma, T. Meisel, M. L. Paleico. A Comparison of HPC-Based Quantum Computing Simulators Using Quantum Volume. – In: Proc. of GI Quantum Computing Workshop, Informatik, 2024. Bonn: Gesellschaft für Informatik e.V. Wiesbaden, 24-26. September 2024. pp. 579-589. DOI: 10.18420/inf2024_46.
3. Cicero, A., M. A. Maleki, M. W. Azhar, A. F. Kockum, P. Trancoso. Simulation of Quantum Computers: Review and Acceleration Opportunities. – In: ACM Transactions on Quantum Computing, September 2025. DOI: 10.1145/3762672.
4. Crouzet, Y., K. Kanoun. System Dependability: Characterization and Benchmarking. – Adv. Comput., Vol. **84**, 2012, pp. 93-139.
5. Dongarra, J. J., P. Luszczek. HPL-MxP Benchmark: Mixed-Precision Algorithms, Iterative Refinement, and Scalable Data Generation. – The International Journal of High Performance Computing Applications, Vol. **40**, 2025, pp. 52-62.
6. Dongarra, J., P. Luszczek, A. Petitet. The LINPACK Benchmark: Past, Present, and Future. – Concurrency and Computation: Practice and Experience, Vol. **15**, 2003, No 9, pp. 803-820. DOI: 10.1002/cpe.728.

7. D o n g a r r a, J., M. A. H e r o u x, P. L u s z c z e k. A New Metric for Ranking High-Performance Computing Systems. – National Science Review, Vol. **3**, 2016, No 1, pp. 30-35. DOI: 10.1093/nsr/nwv084.
8. D o n g a r r a, J., M. A. H e r o u x. Toward a New Metric for Ranking High-Performance Computing Systems. Sandia National Laboratories, Technical Report No SAND2013-4744, 2013.
9. G a n g a p u r a m, A. J., A. M. L ä u c h l i, C. H e m p e l. Benchmarking Quantum Computer Simulation Software Packages: State Vector Simulators. – SciPost Phys. Core, Vol. **7**, 2024, 75.
10. G r a t s e a, K., V. K a s p e r, M. L e w e n s t e i n. Storage Properties of a Quantum Perceptron. – Physical Review. E, Vol. **110**, 2021, No 2-1, 024127.
11. H e r t e n, A., O. P e a r c e, F. S. M. G u i m a r a e s. An HPC Benchmark Survey and Taxonomy for Characterization. – arXiv, 2025.
12. L i, A., et al. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch, and GPUDirect. – IEEE Transactions on Parallel and Distributed Systems, Vol. **31**, 2019, No 1, pp. 94-107.
13. L i u, H., J. H. S e o, R. M i t t a l, H. H. H u a n g. GPU-Accelerated Scalable Solver for Banded Linear Systems. – In: Proc. of IEEE International Conference on Cluster Computing (CLUSTER'13), 2013, pp. 1-8.
14. L i u, M., D. K. P a n d a. High Performance MPI Micro-Benchmark Suite: OSU Micro-Benchmarks. Ohio State University Technical Report, 2003.
15. M c C a l p i n, J. D. STREAM: Sustainable Memory Bandwidth in High Performance Computers. Technical Report, University of Virginia, 1995.
16. NVIDIA Corporation. NVIDIA HPC SDK: Compilers and Tools for HPC Applications. NVIDIA Developer Resources, 2023.
17. A t a n a s s o v, E., T. G u r o v, T. I v a n o v s k a, S. K a r a i v a n o v a, A. T. S i m c h e v. On the Parallel Implementation of Quasi-Monte Carlo Algorithms. – Lecture Notes in Computer Science, 10665. Springer International Publishing, 2018, pp. 258-265. DOI:10.1007/978-3-319-73441-5_27.
18. P a p u g a, J., A. K a ľ a v s k ý, M. L u t o v i n o v, I. V í z k o v á, S. P a r m a, M. N e s l á d e k. Evaluation of Data Sets Usable for Validating Multiaxial Fatigue Strength Criteria. – International Journal of Fatigue, Vol. **145**, 2021, 106093.
19. Qiskit Team. Qiskit: An Open-Source Quantum Computing Framework. – IBM Quantum, 2026. **https://qiskit.org**
20. TOP500. TOP500 Supercomputer Sites – November 2025. **https://www.top500.org/lists/2025/11/**
21. X i n g, F., H. Y o u, C. L u. HPC Benchmark Assessment with Statistical Analysis. – Procedia Computer Science, Vol. **29**, 2014, pp. 210-219. DOI: 10.1016/j.procs.2014.05.019.

Received: 07.03.2026, Accepted: 18.04.2026