Simulating quantum dynamics: Evolution of algorithms in the HPC context

I. Meyerov,[1, *] A. Liniov,[2, †] M. Ivanchenko,[3, ‡] and S. Denisov[4, §]

[1]*Department of Software and Supercomputing Technologies, Lobachevsky University, 603950, Nizhny Novgorod, Russia*

[2]*Department of Software Engineering, Lobachevsky University, 603950, Nizhny Novgorod, Russia*

[3]*Department of Applied Mathematics, Lobachevsky University, 603950, Nizhny Novgorod, Russia*

[4]*Department of Computer Science, Oslo Metropolitan University, N-0130, Oslo, Norway*

Due to complexity of the systems and processes it addresses, the development of computational quantum physics is influenced by the progress in computing technology. Here we overview the evolution, from the late 1980s to the current year 2020, of the algorithms used to simulate dynamics of quantum systems. We put the emphasis on implementation aspects and computational resource scaling with the model size and propagation time. Our mini-review is based on a literature survey and our experience in implementing different types of algorithms.

I. INTRODUCTION

The agenda of computational quantum physics (CQP) is to provide researchers with tools to model quantum systems on computers. Since most of the problems in quantum mechanics cannot be solved analytically, numerical methods were always in demand and played an important role in the development of quantum mechanics. In the period between the late 1990s and early 2010s, the activity on the CQP field was boosted by several waves of advances in experimental physics, such as the appearance of quantum optics of ultracold matter (marked by the creation of the Bose–Einstein condensate in a lab [1]) and fast progress in superconducting microwave technologies (resulted in the creation of the first generation of quantum computer prototypes [2]). Almost instantly, CQP turned to be not only a branch of theoretical quantum physics that assists the latter in gaining new knowledge but also a toolbox of methods to design new experiments and blueprint quantum devices.

The new status of CQP strengthened the ties between quantum physics and high-performance computing (HPC) and changed the character of the research activity on the field. Starting from the 2010s, a familiarity with cutting-edge computing technologies and knowledge of how to use them to handle larger and more complex models became important elements of the professional expertise. By now, CQP represents a synergetic combination of quantum physics, applied mathematics, and HPC, in which the last component is no less important than the first two.

In this paper, we overview the evolution [3] of the algorithms used for digital simulations of the dynamics of quantum systems. We, therefore, do not discuss different diagonalization, renormalization, and variational techniques used to find ground-state or/and first excited states (unless the corresponding technique is a part of the discussed simulation algorithm). We put the emphasis on such computational aspects as the resource scaling, cluster implementation, and parallelization, and try to address them in the context of the HPC technology development. The evolution is illustrated (Fig. 1) by adapting the idea of the Gartner Hype Cycle for Emerging Technologies [4].

[*]E-mail: meerov@vmk.unn.ru

[†]E-mail: alin@unn.ru

[‡]E-mail: ivanchenko.mv@gmail.com

[§]E-mail: sergiyde@oslomet.no

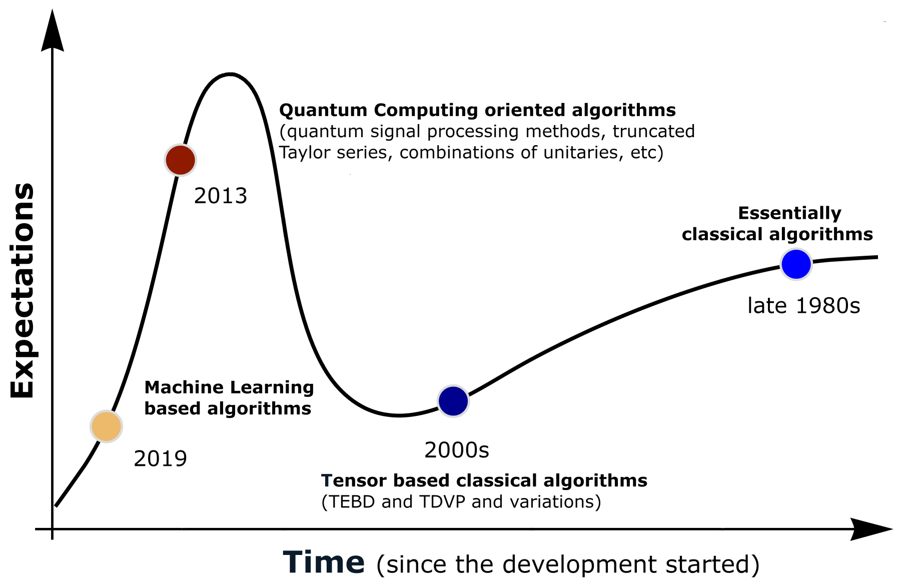

FIG. 1: Evolution curve of algorithms to simulate quantum dynamics.

Our overview is partially based on the literature survey. The analysis of the publications revealed that it is very seldom that the information on the details of algorithm implementations and computational resources is provided, even in additional materials and appendices. Therefore we supplement the consideration by describing our experience in implementing different simulation algorithms on supercomputers "Lobachevsky" (at Lobachevsky State University of Nizhny Novgorod) and "Lomonosov 2" [5] (at Moscow State University).

The remainder of the paper is organized as follows. In Section 2 we discuss the evolution of the algorithms sketched in Figure 1. In Section 3 we address the first two generations of algorithms designed for implementations on classical computers. In Section 4 we discuss algorithms developed for digital quantum simulation (i.e., for implementations on digital quantum computers [6]). In Section 5 we outline recent advances related to the Machine Learning approach to QCP. Finally, in Section 6 we summarize the consideration and discuss future perspectives and possible trends.

II. EVOLUTION CURVE OF SIMULATION ALGORITHMS

To model the dynamics of a quantum system, evolving in a Hilbert space $\mathcal{H}$ of dimension $dim\ \mathcal{H} = N$, we have to integrate numerically the initial value problem for one of the two operator-valued differential equations. In the case of coherent unitary evolution, it is the Schrödinger equation,

$$i\hbar|\dot{\varphi}(t)\rangle = \hat{H}(t)|\varphi(t)\rangle, \tag{1}$$

with the initial wave function $|\varphi(0)\rangle = |\varphi_0\rangle$. The Hilbert space can be spanned with a basis, $\{|\psi_j\rangle\}$, $j = 1, 2, ..., N$. The expansion over the basis allows transforming the wave function into an N-dimensional complex column vector, $|\varphi(t)\rangle = \sum c_j(t)|\psi_j\rangle \ \Rightarrow [c_1(t), c_2(t), ..., c_N(t)]^T$, and the Hamiltonian operator $\hat{H}(t)$ into an N-dimensional time-dependent Hermitian matrix. In the case of open evolution, when, e.g., the system is coupled to its environment, the state of the system is described with the density operator $\varrho(t)$. The evolution of this operator is governed by a quantum Liouvillian $\mathcal{L}(t)$ (most often of the so-called Gorini–Kossakowski–Sudarshan–Lindblad

(GKSL) form [7] but some other forms [8] are also used),

$$\dot{\varrho}(t) = \mathcal{L}(t)\varrho(t), \tag{2}$$

with the initial density operator ϱ_0. Often the density operator is also vectorized and transformed into N^2 (or $N^2 - 1$) complex vector and the Liouvillian is then recast in the form of an N^2 (or $N^2 - 1$)-dimensional time-dependent matrix.

We adapt the idea of the Gartner Hype Cycle of Emerging Technologies [4] to illustrate the evolution of algorithms developed to integrate Eqs. (1-2); see Fig. (1). The evolution consists of several stages and reflects the overall attitude of the CQP community towards an algorithm. At first, a new promising algorithm is recognized as an "Innovation Trigger". If there is growing confidence in the community that the algorithm has a potential and can be used to solve standing problems, the activity and number of publications on the algorithm are going up and this brings the latter to the "Peak of Inflated Expectation". The peak is usually succeeded by the phase of sobering and realization of the algorithm limitations, and the community attitude slides down to the "Trough of Disillusionment". Finally, if the algorithm survives this phase, it enters the "Plateau of Productivity" and becomes a part of the conventional CQP toolbox. We address four different 'species', or (with a slight abuse of the terminology), *generations* of the algorithms.

The first generation was remaining the only one until the late 1980s. At that time, the overwhelming majority of the considered models describe single-particle systems and an implementation of one of the standard schemes used to integrate differential equations (though tailored to the symplectic structure of the Schrödinger equation) such as different variants of split-step, pseudo-spectral, and Krylov subspace methods [9], was enough in most cases. These algorithms remain an important part of the CQP toolbox and they are routinely used, e.g., in computational quantum chemistry. During this initial stage of the evolution, computational aspects of algorithm implementations were not considered as important and such questions as resource scalings and parallelization were out of the focus.

The situation changed with the rise of many-body quantum physics in the late 1990s. This was the time when the computational quantum community faced the "Curse of Dimensionality" [10], which in this case turned to be an exponential growth of the amount of data, needed to be stored, and computational resources needed to process this data, to simulate the dynamics of a many-body model, with the number of 'bodies' (spins, qubits, photons, etc) the model is built of.

The appearance, at the beginning of the 2000s, of a new generation of algorithms [11–13], based on the low-rank tensor approximation ideas [14], was a substantial advance in the fight with the Curse. The algorithms turned to be very successful when used to simulate many-body systems of linear topology characterized by a short-range entanglement and were a decisive factor in the development of new research fields such as many-body localization and transport in quantum disordered systems. These algorithms [posses a substantial potential for parallelization and their appearance initiated the development of the HPC component in CQP. It has also become clear to non-experts (though experts were aware of this limitation from the beginning) that the propagation of a generic model with unbounded growth of entanglement is limited by a certain – often a very short – time horizon and so these algorithms have a limited scope and should be used with care [15, 16]. On the evolution curve, we place these algorithms (rather subjectively) to the "Trough of Disillusionment" region.

First two generations are *classical*, i.e., the corresponding algorithms were on purpose designed to simulate quantum models on classical digital computers [11]. There are two new CQP trends that are catalyzed by recent technological advances. First, it is the appearance of quantum computer prototypes on the IT market and their accessibility to the QCP researchers [17]. Even though the devices manufactured by Rigetti, IBM, Google, and Microsoft belongs to the generation of the so-called 'Noisy Intermediate-Scale Quantum (NISQ)' processors [18], their advent boosted the development of a new generation of simulation algorithms [19–21] designed for implementations on future digital quantum computers. These algorithms go beyond the generic Trotterization ideology [22] and can be used to simulate dynamics of quantum models of different types, not only systems of qubits or spins.

Finally, the rise of Machine Learning (which currently is modifying the whole paradigm of computational physics [23]) brought new perspectives. Machine learning strategies and artificial neural networks (ANNs) were proposed as means to compactify the descriptions of many-body states and recognize different quantum phases [23]. It was realized that Matrix Product States (MPS), which serve a foundation for the tensor-based simulation algorithms, represent a particular class of the ANN states. There are several works appeared recently in which ANNs were suggested as the tool to simulate the dynamics of many-body models [24–26], even beyond the time scales that can be reached with the most advanced tensor-based algorithms [26].

III. CLASSICAL ALGORITHMS

A. Performance optimization and parallelization: Why it is important?

Modern computing systems are becoming more and more complex and heterogeneous. This requires new methods for performance optimization and parallelization of scientific software. Such methods are supposed to utilize resources of cluster systems with multi- and many-core CPUs at the following levels:

- Computational nodes on distributed memory: reducing the overhead of data transfers, load balancing;
- Computational cores on shared memory: Non-uniform memory access (NUMA)-aware memory usage, cache optimizations, load balancing;
- Single instruction multiple data (SIMD) units in computational cores: code vectorization.

In this regard, we often need to develop a hybrid parallel algorithm by using MPI + OpenMP/TBB/. . . technologies with efficient use of SIMD instructions. When using graphic processing units (GPUs), we can potentially make computations faster if the task is tidied up to a limited amount of GPU's internal memory but the code development often becomes more intricate.

When implementing already existing algorithms, the problems of optimizing performance and efficient parallelization comes to the fore. Digital simulations of quantum model by using an algorithm belonging to the first generation is computationally a very expensive task. To simulate the dynamics of an open quantum system, described by the master equation, Eq. (2), we have to operate with matrices of the size $N^2 \times N^2$. This can be problematic already starting $N \sim 10^3$. It is often necessary to repeat simulations many times, for example, in order to analyze the behavior of a model system at a large number of points in the parameter space.

In this regard, there is a need to develop and use custom data structures and parallel algorithms that allow solving the problem on a cluster in an acceptable time and fit into the imposed memory limits. To date, several software toolkits to simulate dynamics of quantum systems on classical computers have been developed. For example, an open source software QuTiP (Quantum Toolbox in Python) [27] allows one to model dynamics of open quantum systems. Being developed in Python, QuTiP is based on the high-performance libraries Numpy, Scipy, and Cython, which take care of performance. There is also a TBTK package [28], " an open-source C++ framework for modeling and solving problems formulated using the language of second quantization". Another software, WavePacket, is the MATLAB code, designed to simulate dynamics of coherent [29] and open [30] quantum models. The versatility and variability of methods implemented in these and other software toolkits are an undoubted advantage; however, achieving high efficiency of using parallel supercomputers requires taking into account the features of the specific task and the model used. In this regard, we consider some examples demonstrating how the development of custom data structures and tailored parallel algorithms can lead to a significant performance gain.

B. Essentially classical algorithms

When realizing some of our CQP projects, we designed and implemented two types of algorithms belonging to the first generation, one to simulate unitary dynamics of quantum models with an explicitly time-dependent Hamiltonians [31] and another one to simulate dynamics of open models with time-dependent Liouvillians [32, 33].

In [31] we considered the problem of parallelization of an algorithm to calculate eigenstates of large quantum models with Hamiltonians that are modulated periodically in time. This demanded numerical propagation of many initial states up to the time equal to the period of modulations. The main part of the algorithm combines the Magnus expansion of the time-dependent system Hamiltonian with the Chebyshev expansion of an operator exponent [38]. In this method, the computations are based on linear algebra operations, in particular, on the dense matrix-vector multiplications. Parallelization of calculations on distributed memory is performed trivially with an ideal load balancing by dividing the vectors of initial conditions among the processes with subsequent propagation and collection of the results and their relatively simple processing on one master node. The main optimization applicable in the calculations at each node of the cluster is the simultaneous propagation of all vectors of initial conditions, which allows switching from matrix-vector multiplications to matrix-matrix multiplications. This optimization speeds up calculations by an order of magnitude due to more efficient work with memory hierarchy with the same number of floating-point operations. Vectorization of computations is achieved through the use of high-performance BLAS implementations, for example, from the Intel Math Kernel Library. Note that it is a common approach: the switching to the third level BLAS and the use of highly-optimized mathematical software usually allows achieving near optimal performance. In this problem, the code can be easily ported to GPUs by switching to cuBLAS with a significant reduction in computation time.

Similar methods are considered in the paper [36], which explains how to efficiently implement the Magnus integrator with Leja interpolation. The paper explores the commutator free method, which replaces computationally intensive matrix multiplications with matrix-vector multiplications thus reducing the number of floating point operations. The authors state that such their method is better from the numerical analysis point of view, but it does not fit the architecture of modern parallel CPUs and GPUs, which eliminates its advantages. The authors also demonstrated that GPUs can speed up calculations by an order of magnitude in a dense testbed problem.

Another example of the effective use of specific problem features is given in [32] and [33]. The implemented algorithm transforms the Lindblad equation into a system of linear ordinary differential equations with real coefficients by using the generalized Gell-Mann matrices as a basis [32]. Such a system can be propagated forward in time with one of the standard high-order integration methods. It was demonstrated [32], that a naive method of computing this expansion requires enormous computation time and memory, while the construction of the specific data structures and methods for their computing, based on counting the only nonzero elements, can significantly reduce both computational complexity and memory usage. In this method, sorting algorithms and dense & sparse BLAS operations are the main mathematical kernels. Once again, we can profit from high-performance mathematical software libraries. In [33] it was shown that these algorithms can also be parallelized for cluster-based implementations. The main goal of parallelization in this case is not to reduce the computation time, but to reduce the memory costs per each cluster node. As a result, it was possible to simulate the model with $N = 200$ states (dense case) and $N = 2000$ states (sparse case) using 25 nodes of a cluster with 64 GB RAM per node.

In this section, we would like also mention the Krylov subspace method [39, 40] which remains popular in the CQP community. The method proved to be very efficient, from the point of memory size, when the average number of Krylov vectors on every propagation step is much smaller than N. The corresponding algorithms are part of an open-source package [41] and the results of its performance analysis can be found in [42]. A parallel supercomputer implementation was reported in [43].

C. Tensor-based classical simulation algorithms

The total length L of the description (the number of complex coefficients required to specify a quantum state) of a model system consisting of M elements, each one with d degrees of freedom, scales as $L(M) \sim N = d^M$. To specify an *arbitrary* state of a system of 50 qubits, we need $2^{50} \approx 10^{15}$ complex numbers. With the double-precision format, this exceeds the memory capacity of the supercomputer "Lomonosov 2" [5]. For an open quantum model, the complexity squares: to specify a density operator we need $L(M) \sim d^{2M}$ real-valued parameters.

Singular Value Decomposition (SVD) is one of the best tools to reduce the amount of stored data when dealing with large matrices [44]. Its generalization to tensors, so-called Tensor-Train (TT) decomposition [14], also turned to be very effective when dealing with tensors. In the physical literature, this decomposition is commonly referred to as Matrix Product State (MPS; in the case of pure states) or Matrix Product Operator (MPO, in the case of mixed states) representation [45]. While two names are used simultaneously (though in different fields), the underlying mathematical structure is basically the same [13]. The MPS/MPO/TT approach allows us to reduce the growth of the description of *some* many-body states to a linear scaling $L(M) \sim M$ [14].

The MPS/MPO representation can also be used for an effective propagation of quantum many-body models. Following the Time-Evolving Block Decimation (TEBD) method [11], the description of the state, obtained after every propagation step, is reduced to a fixed length L_{cut}. The accuracy of the propagation is therefore controlled by the value of L_{cut}: If the information is thrown out after the reduction is substantial, the propagation is bad and leads to a wrong result. Otherwise, it is good. Some many-body systems 'behave' well during the TEBD propagation and so the amount of neglected information is tolerable (we are not going to discuss physical properties underlying such a 'good behavior' and refer the reader to the extensive literature on the subject; see. e.g., Ref. [45]).

In our recent work [35], we tried to reproduce the results reported in [34], where a disordered chain of M spins, with a next-neighbor coupling and two "thermal reservoirs", acting on the two end spins, was used as the model. The transport of the spin charge through the chain in the stationary regime was considered and the spin current scaling with the chain length M was analyzed. The results for $M = 400$ were reported; see Fig. 2. This is an unprecedented size for a many-body open quantum model (to the best of our knowledge at the time). The complexity of the computational experiments was increased by the fact that the model had to be propagated over a long time in order to reach the stationary state. Finally, to obtain scaling dependencies, an averaging over many disorder realizations was performed. At the same time, the work reports no details of numerical simulations (even the value of such an important parameter as the bond

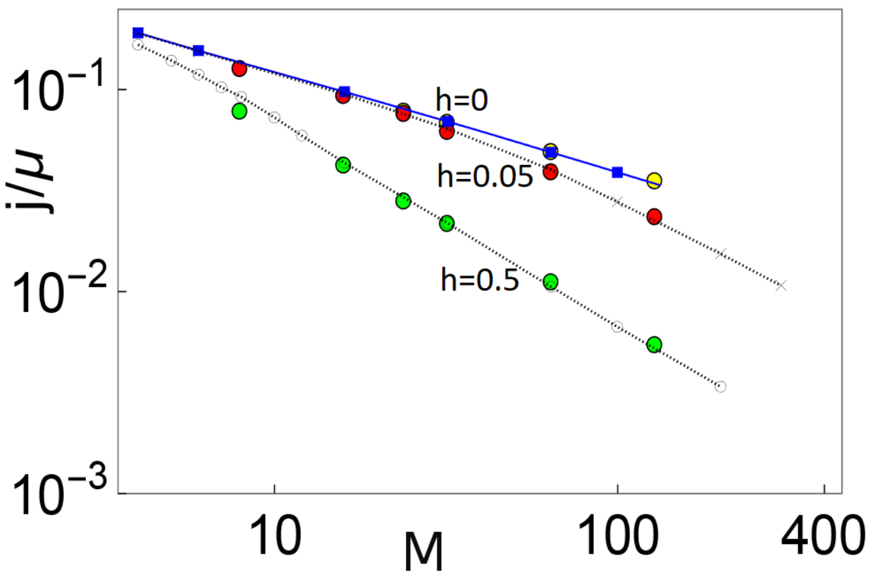

FIG. 2: Scaling of the spin current through a disordered spin chain with M spins for different values of disorder strength h. Our results [35] (big filled circles) are plotted on top of the results reported in [34].

dimension R was not specified).

We implemented a parallel version of the TEBD algorithm by using the MPI technology and the standard master-worker scheme. The computational experiments have been performed on the "Lobachevsky" cluster, with a 2×8-core Intel Xeon CPU E5-2660, 2.20 GHz, 64 GB RAM, Infiniband QDR interconnect. The code was compiled with the Intel C++ Compiler, Intel Math Kernel Library and Intel MPI from the Intel Parallel Studio XE suite of development tools and the Armadillo library. We were able to reach reproduce the results for $M = 128$ spins (see Fig. 2), by running our code on four nodes of the cluster (one MPI-process per CPU core, 64 MPI processes overall). Total computation time for a single disorder realization was 143 s.

The Suzuki–Trotter decomposition [22] is one of the key ingredients of the TEBD scheme. In [37], the authors investigated the use of CPUs and GPUs to optimize the performance of one round of the decomposition. It was demonstrated that it is possible to obtain a significant performance gain through code vectorization and optimization of memory usage patterns. By using GPU calculations in some model problems it was possible to reduce the computation time by one order of magnitude. As a subsequent development, a massively parallel version of the solver was presented in [46] and its functionality was further expanded in [47].

There is a family of more advanced algorithms realizing the so-called Time-Dependent Variational Principle (TDVP) [12, 13]. The main difference from the TEBD family is that the model evolution is confined to the subspace of the MPS (MPO) states and therefore there is no need to perform the truncation after every steep (or a fixed number of steps). This does not mean, however, that the TDVP propagation is numerically exact. In fact, a model system could leave the MPS/MP subspace when evolving in its Hilbert space; in other words, the confinement is imposed not by the physics of the model but by the algorithm. Currently, the TDVP-based algorithms are considered to be the most advanced simulation algorithms in the CQP community working with many-body models.

In [49], a detailed analysis of the single-node performance of TEBD and TDVP implementations is presented. Both implementations were tested on "a single core of a Xeon E5-2630 v4 with 64 GB of RAM" and so the parallelization potential was not addressed. In a very recent work [48], a cluster implementation of the TDVP scheme to simulate spin chain models with long-range interactions was presented. It is reported that the code "scales well up to 32 processes, with parallel efficiencies as high as 86 percent" and results of simulations for 201-site Heisenberg XXX spin chain with quadratically decaying interactions are presented.

Finally, there is a generalization of the MPS concept to higher-dimensional models, the so-called Projected Entangled Pair States (PEPS) [50]. The thermodynamic limit is addressed with infinite-PEPS (iPEPS) [51] and there are several algorithms implementing the iPEPS representation to simulate the dynamics of infinite-size lattice models. Here we mention a relatively recent work, in which some computational aspects of iPEPS-based simulations are discussed [52]. However, no information on the computation resources used for simulations is provided. We guess that the simulations were performed on a single core.

IV. ALGORITHMS FOR DIGITAL QUANTUM SIMULATIONS

The qubit architecture of the quantum processors manufactured by Rigetti, IBM, Google, and Microsoft, gives hope that Feynman's idea [53] of modeling quantum systems on quantum computers will be realized in near future. However, future full-fledged digital quantum computers require new algorithms whose logics is principally different from the logics of algorithms currently used in computational quantum physics – simply because the latter are designed to be implemented on classical (super)computers.

There is a new QC-oriented paradigm that already brought several algorithms to simulate the dynamics of quantum systems, of different nature and genesis, on quantum processors of the qubit architecture; see, e.g., Refs. [20, 21]. The corresponding research activity, as well as the expectations placed on these algorithms, are going up steadily, being heated by the fast progress on the front of QC-technologies and strong competition between the main players on the QC market.

Similar to their classical predecessors, quantum simulation algorithms can be characterized by scalings "number of operations (gates) vs time of propagation and/or size of the model and/or accuracy". The first step in this direction was reported in a recent work [54]. A method to 'compile', i.e., to minimize the number of gates, was tested with several existing algorithms and a one-dimensional nearest-neighbor Heisenberg system was used as the model. The scaling "number of gates vs the size of the model (number of spins)" was addressed. It is not clear though whether actual *simulations* (e.g., by using a classical emulator) were performed.

There is a need for more quantitative results and evaluation tests of the algorithm performances on the intermediate scale, 30 – 60 qubits, ideal (i.e., noise-free) quantum processors [55]. Simulations of different models, ranging from textbook examples to technology-relevant systems, could bridge quantum physics and quantum computing in a new way, by allowing, for example, to describe properties of quantum models in terms of their *quantum* computational scalability. Presently, such a research program can only be realized on classical computers. This immediately leads us to the question of the efficiency of cluster-based emulators.

Modern supercomputers allow simulating quantum systems consisting of 38 [57], 42 [65], 45 [70], 49 [71], possible 53 [72], and in special cases up to 64 qubits [73]. There is a variety of open-source packages [60] that can be used to emulate quantum circuits [61–64], execute quantum algorithms [57, 65–67], and even support a complete development cycle from describing an algorithm to mapping it onto specific quantum computer architecture, including optimization, verification, and performance evaluation [68]. Due to the various characteristics [69], there is an option to choose an emulator most optimal for a specific task and available computing resources. We choose QuEST [56], a recently released "open source, hybrid multi-threaded and distributed, GPU accelerated simulator of universal quantum circuits" [57]. Even though the results of the performance analysis of QuEST are reported in Ref. [57], we decided to do it independently.

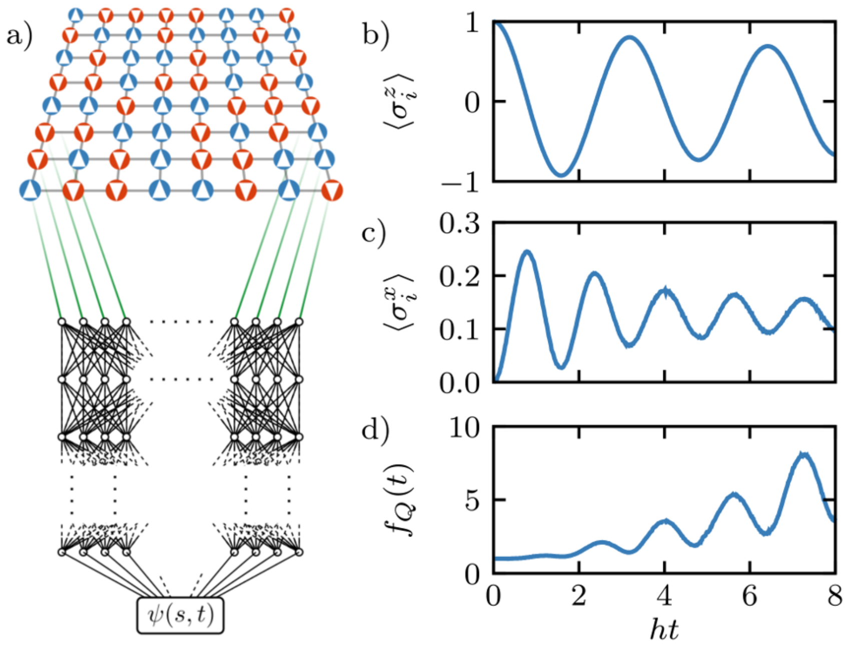

FIG. 3: (a) Sketch of the propagation of a many-body model, the transverse-field square-lattice Ising system, whose state is encoded by a deep convolution neural network. (b - c) Oscillations of the ferromagnetic order induced by the quench of the magnetic field at $t = 0$. (d) The so-called quantum Fisher information reveals the development of the multipartite entanglement in the system in the course of the propagation. Courtesy of Markus Heyl.

Our experiments using the QuEST package show that simulations of circuits consisting of 30–36 qubits are possible on a single node, with an evident limiting factor that is the amount of memory. For 30 qubits, about 18 GB is required, for 36 – about 1.1 TB. The distributed version allowed us to use the computing power and memory of several nodes, for example, we were able to simulate (with a speed-up factor $20 - 30$ due to the MPI/Open MP use) up to 38 and 40 qubits on the supercomputers ”Lobachevsky” and “Lomonosov-2”, respectively. In our opinion, it is enough to emulate digital quantum simulations with different scalable models of non-qubit(spin) nature.

V. ARTIFICIAL NEURAL NETWORK SIMULATION ALGORITHMS

There is an emerging trend in the CQP field initiated by the rise of the Machine Learning technologies [23]. Among many other interesting findings, it was found that artificial neural networks (ANNs) can be used to store highly entangled states of many-body systems [23, 58]. It was also found that many-body states which belong to the MPS/MPO (one-dimensional lattices) or PEPS (two-dimensional lattices) classes represent sub-sets of the ANN states [58, 59].

There is an immediate question: Can we make the next step and use a trained ANN to propagate the state of a quantum model forward in time? This idea was put forward and tested in the work by Carleo and Troer [24]. They used a popular model, a square-lattice transversed-field Ising system, and a special type of ANN, the so-called Restricted Boltzmann Machine [74]. By quenching the magnetic field, a strongly non-equilibrium evolution was initiated. It was demonstrated that a trained Reduced Boltzmann Machine is able to propagate the model with high accuracy (so that, in terms of observable the difference from the numerically exact unitary propagation remained small) up to a time sufficient to see the quench-induced oscillations in the ferromagnetic order.

In a recent work [26], a deep convolutional neural network (a broader class than Restricted Boltzmann Machines) was used. The authors demonstrated that with it is possible to simulate the dynamics of the $2d$ transversed-field Ising model with higher accuracy and for a time longer than can be achieved with a tensor-based algorithm [52] (presumably, by using the same computation resources). The simulations were performed on an Intel Xeon E5-2680 server by using the standard MPI/OpenMP technology. An alternative implementation on a GPU VIDIA V100 was tested and turned to be “30 times faster than our OpenMP implementation on 20 cores and still 15 times faster than an ideal parallelization on 20 CPU cores” [26]. Similar results, by using the same model (but of a smaller size) and a slightly different ANN approach, were also reported recently in [25].

VI. CONCLUSION

We tried to overview the evolution of the numerical algorithms developed to simulate dynamics of quantum systems and discussed this evolution in the context of the HPC development. Evidently, there are algorithms that remained outside the classification presented with Fig. 1 and which therefore were not addressed.

For example, there is the so-called ”quantum trajectories“ (QT) or ”quantum jump“ method (also known as the ”Monte Carlo wave function“ method) [7], which allows transforming the numerical integration of the master equation, Eq. (2), into a task of statistical sampling over an ensemble of quantum trajectories. Thus, we could deal with an N-dimensional vector instead of an N-dimensional matrix. The price to be paid for this reduction is that we have to sample over many realizations. However, the sampling is an embarrassingly parallel problem and thus we can benefit substantially from the use of a cluster. In Refs. [75, 76] we consider the problem of reaching the asymptotic state of a non-equilibrium open quantum model by using the QT-algorithm. We demonstrated that on a comparatively small cluster it is possible to propagate models with $N = 2000$ states. The parallelization for both shared and distributed memory is straightforward, which is typical for Monte Carlo methods. The main difficulty there lies in the efficient use of the memory hierarchy, since in a straightforward implementation, not very effective matrix-vector multiplications are the dominant operations. It was shown in Ref. [76] that by using a specially developed algorithm, it was possible to group many matrix-vector multiplications into equivalent matrix multiplications, which led to a 17-fold acceleration.

As respect to the third generation, the question ”What the term ’parallelization’ could mean in this case?” is of interest. Are there simulation algorithms which can be effectively implemented on several digital quantum processors, that are wired in a *classical* way? (it might be that this question was already addressed in the literature and we are simply ignorant of this fact). Another interesting direction is the development of ’quantum compilers’ and ’quantum software’ [77, 78]. We hope that the researchers working on quantum software will soon address parallelization aspects.

Finally, we would like to discuss possible future trends related to the last, fourth generation. Even though it is hard to gauge the potential of the ML paradigm in the CQP context, the first results look very promising and inspiring. At this point the questions on how to use supercomputers for training ANNs and the subsequent high-performance inference become relevant. Currently, systems based on GPUs are mainly to train ANNs. The GPU architecture is optimal for a large number of the same computationally intensive operations on different data sets that occur during training. At the same time, for ANN high-performance inference, we can effectively use also CPUs, whose instruction set have recently extended by the Intel Deep Learning Boost instructions including Vector Neural Network Instructions (VNNI) that enable INT8 deep learning inference support [79]. Along with traditional architectures, new technological developments are constantly emerging. The initiative of Intel Corporation, which plans to release a new GPU optimized for

AI and high-performance computing [80], and the development of the Graphcore company, which released the Graphcore IPU (Intelligence Processing Unit) designed for AI algorithms [81] seem to be very relevant in this respect. We are confident that during the next decade we will witness the rapid development of hardware and software tools focusing on both high-performance scientific computing and the use of Machine Learning technologies. Hopefully, this progress will also affect the research activity in the QCP field.

Acknowledgments

The work is supported by the Russian Science Foundation via Grant No. 19-72-20086. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [5] and the Lobachevsky supercomputer at Lobachevsky University of Nizhny Novgorod.

---

[1] K. B. Davis et al., "Bose-Einstein condensation in a gas of sodium atoms," Phys. Rev. Lett. **75**, 3969–3973 (1995).

[2] R. Barendes et al., "Coherent Josephson qubit suitable for scalable quantum integrated circuits," Phys. Rev. Lett. **11**, 080502 (2013).

[3] We use this term in its biological sense, as a "cumulative inherited change in a population of organisms through time leading to the appearance of new forms" (Dictionary, Merriam-Webster. `https://www.merriam-webster.com/dictionary/evolution`).

[4] Gartner: Hype Cycle Research Methodology. `https://www.gartner.com/en/research/methodologies/gartner-hype-cycle`. Accessed 2020.

[5] V. I. Voevodin, A. Antonov, D. Nikitenko, P. Shvets, S. Sobolev, I. Sidorov, K. Stefanov, Vad. Voevodin, and S. Zhumatiy, "Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community,"Supercomputing Frontiers and Innovations **6**, 4–11 (2019).

[6] M. A. Nielsen, and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2010).

[7] H.-P. Breuer and F. Petruccione, *The Theory of Open Quantum Systems* (Oxford University Press, Oxford, 2002).

[8] S. Kohler, J. Lehmann, and P. Hänggi, "Driven quantum transport on the nanoscale," Phys. Rep. **406**, 379–443 (2005).

[9] R. Kosloff, "Propagation methods for quantum molecular dynamics," Annu. Rev. Phys. Chem. **45**, 145–78 (1994).

[10] R. Bellman, *Dynamic Programmings* (Princeton University Press, 1957).

[11] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," Phys. Rev. Lett. **91**, 147902 (2003).

[12] J. Haegeman et al., "Time-dependent variational principle for quantum lattices," Phys. Rev. Lett. **107**, 070601 (2011).

[13] J. Haegeman, C. Lubich, I. Oseledets, B. Vandereycken, and F. Verstraete, "Unifying time evolution and optimization with matrix product states," Phys. Rev. B **94**, 165116 (2016).

[14] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions," SIAM J. Sci. Comput. **31**, 3744–3759 (2009).

[15] B. Kloss, Y. Bar Lev, and D. Reichman, "Time-dependent variational principle in matrix-product state manifolds: Pitfalls and potential," Phys. Rev. B **97**, 024307 (2018).

[16] S. Goto, and I. Danshita, "Performance of the time-dependent variational principle for matrix product states in the long-time evolution of a pure state," Phys. Rev. B **99**, 054307 (2019).

[17] IBM Q Experience. `https://www.ibm.com/quantum-computing/technology/experience/`. Accessed 2020.

[18] J. Preskill, "Quantum Computing in the NISQ era and beyond," Quantum **2**, 79 (2018).

[19] D. W. Berry et al., "Simulating Hamiltonian dynamics with a truncated Taylor series," Phys. Rev. Lett. **114**, 090502 (2015).
[20] G. H. Low and I. L. Chuang, "Optimal Hamiltonian simulation by quantum signal processing," Phys. Rev. Lett. **118**, 010501 (2017).
[21] G. H. Low and I. L. Chuang, "Hamiltonian simulation by qubitization," Quantum **3**, 163 (2019).
[22] S. Lloyd, "Universal quantum simulators," Science **273**, 1073 (1996).
[23] G. Carleo et al., "Machine learning and the physical sciences," Rev. Mod. Phys. **91**, 045002 (2019).
[24] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," Science **355**, 602-606 (2017).
[25] I. López-Gutiérrez and C. B. Mendl, "Real time evolution with neural-network quantum states," arXiv:1912.08831 (2019).
[26] M. Scmitt and M. Heyl, "Quantum many-body dynamics in two dimensions with artificial neural networks," arXiv:1912.08828 (2019).
[27] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP 2: A Python framework for the dynamics of open quantum systems," Comp. Phys. Comm. **184**, 1234–1240 (2013).
[28] K. Björnson, "TBTK: A quantum mechanics software development kit," SoftwareX **9**, 205–210 (2019).
[29] B. Schmidt and U. Lorenz, "WavePacket: A Matlab package for numerical quantum dynamics. I: Closed quantum systems and discrete variable representations," Comp. Phys. Comm. **213**, 223–234 (2017).
[30] B. Schmidt and C. Hartmann, "WavePacket: A Matlab package for numerical quantum dynamics. II: Open quantum systems, optimal control, and model reduction," Comp. Phys. Comm. **228**, 229–244 (2018).
[31] T. V. Laptyeva et al., "Calculating Floquet states of large quantum systems: A parallelization strategy and its cluster implementation," Comp. Phys. Comm. **201**, 85-94 (2016).
[32] A. Liniov et al., "Unfolding a quantum master equation into a system of real-valued equations: Computationally effective expansion over the basis of SU (N) generators," Phys. Rev. E **100**, 053305 (2019).
[33] I. Meyerov et al., "Transforming the Lindblad equation into a system of linear equations: Performance optimization and parallelization," arXiv: 1912.01491 (2019).
[34] M. Ẑnidarič, A. Scardicchio, and V. K. Varma, "Diffusive and subdiffusive spin transport in the ergodic phase of a many-body localizable system," Phys. Rev. Lett. **117**, 040601 (2016).
[35] V. Volokitin et al., "Propagating large open quantum systems towards their asymptotic states: cluster implementation of the time-evolving block decimation scheme," J. of Phys.: Conf. Series **1392**(1), 012061 (2019).
[36] N. Auer et al., "Magnus integrators on multicore CPUs and GPUs," Comp. Phys. Comm. **228**, 115–122 (2018).
[37] C. S. Bederián and A. D. Dente, "Boosting quantum evolutions using Trotter-Suzuki algorithms on GPUs," In: Proceedings of HPCLatAm-11, 4th High-Performance Computing Symposium, Cordoba, Argentina (2011).
[38] S. Blanes et al., "The Magnus expansion and some of its applications," Phys. Rep. **470**(5-6), 151–238 (2009).
[39] C. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," SIAM Rev. **45**, 3–49 (2003).
[40] Y. Saad, "Analysis of Some Krylov Subspace Approximations to the Matrix Exponential Operator," SIAM J. Numer. Anal. **29** (1), 209–228 (1992).
[41] D. Jaschke, M. L. Wall, and L. D. Carr, "Open source Matrix Product States: Opening ways to simulate entangled many-body quantum systems in one dimension," Comp. Phys. Comm. **225**, 59–91 (2018).
[42] D. Jaschke and L. D. Carr, "Open source matrix product states: exact diagonalization and other entanglement-accurate methods revisited in quantum systems," J. Phys. A: Math. and Theor. **51**, 465302 (2018).
[43] M. Brenes et al., "Massively parallel implementation and approaches to simulate quantum dynamics using Krylov subspace techniques," Comput. Phys. Commun. **235**, 477–488 (2019).
[44] H. Samet, *Foundations of Multidimensional and Metric Data Structures* (Morgan Kaufmann, 2006).
[45] U. Schollwoeck, "The density-matrix renormalization group in the age of matrix product states," Ann. of Phys. **326**, 96 (2011).
[46] P. Wittek and F. M. Cucchietti, "A second-order distributed Trotter-Suzuki solver with a hybrid CPU-GPU kernel," Comp. Phys. Comm. **184**, 1165–1171 (2013).

[47] P. Wittek and L. Calderaro, "Extended computational kernels in a massively parallel implementation of the Trotter-Suzuki approximation," Comp. Phys. Comm. **197**, 339–340 (2015).

[48] P. Secular, N. Gourianov, M. Lubasch, S. Dolgov, S. R. Clark, and D. Jaksch, "Parallel time-dependent variational principle algorithm for matrix product states," arXiv:1912.06127 (2019).

[49] S. Paeckel et al., "Time-evolution methods for matrix-product states," Ann. of Phys. **411**, 167998 (2019).

[50] V. Murg, F. Verstraete, and J. I. Cirac, "Variational study of hard-core bosons in a two-dimensional optical lattice using projected entangled pair states," Phys. Rev. A **75**, 033605 (2007).

[51] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac, "Classical simulation of infinite-size quantum lattice systems in two spatial dimensions," Phys. Rev. Lett. **101**, 250602 (2008).

[52] Ho N. Phien et al., "Infinite projected entangled pair states algorithm improved: Fast full update and gauge fixing," Phys. Rev. B **92**, 035142 (2015).

[53] R. P. Feynman, "Simulating physics with computers," Int. J. Theor. Phys. **21**, 467-488 (1982).

[54] A. M. Childs et al., "Toward the first quantum simulation with quantum speedup," PNAS **115**, 9456 (2019).

[55] We do not address here the issue of fault tolerant quantum computations and different error mitigation techniques, which are subjects of active research; see, e.g., recent works S. Endo, S. C. Benjamin, and Ying Li, "Practical quantum error mitigation for near-future applications," Phys. Rev. X **8**, 031027 (2018) and Chao Song, Jing Cui, H. Wang, J. Hao, H. Feng, and Ying Li, "Quantum computation with universal error mitigation on a superconducting quantum processor," Sci. Adv. **5**, eaaw5686 (2019).

[56] QuEST – Quantum Exact Simulation Toolkit. `https://quest.qtechtheory.org/`. Accessed 2020.

[57] T. Jones, A. Brown, I. Bush, and S. C. Benjami, "QuEST and high performance simulation of Quantum Computer," Sci. Rep. **9**, 1073 (2019).

[58] Zhih-Ahn Jia et al., "Quantum neural network states: A brief review of methods and applications," Adv. Quantum Technol., 1800077 (2019).

[59] I. Glasser, N. Pancotti, M. August, I. D. Rodriguez, and J. I. Cirac, "Neural-network quantum states, string-bond states, and chiral topological states," Phys. Rev. X **8**, 011006 (2018).

[60] List of QC simulators. `https://quantiki.org/wiki/list-qc-simulators`. Accessed April 2020.

[61] A. S. Green et al., "Quipper: a scalable quantum programming language," In: Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, 333-342 (2013).

[62] A. W. Cross et al., "Open quantum assembly language," arXiv:1707.03429 (2017).

[63] K. Svore et al., "Q# Enabling scalable quantum computing and development with a high-level DSL," In: Proceedings of the Real World Domain Specific Languages Workshop 2018, 1-10 (2018).

[64] A. J. Abhari et al., "Scaffold: Quantum programming language," TR-934-12 (2012).

[65] G. Guerreschi et al., "Intel Quantum Simulator: A cloud-ready high-performance simulator of quantum circuits," arXiv:2001.10554 (2020).

[66] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, "qHiPSTER: the quantum high performance software testing environment," arXiv:1601.07195 (2016).

[67] G. Aleksandrowicz et al., "Qiskit: An open-source framework for quantum computing," `https://zenodo.org/record/2562111`. Accessed April 2020.

[68] M. Amy and V. Gheorghiu, "staq–A full-stack quantum processing toolkit," arXiv:1912.06070 (2019).

[69] A. B. de Avila et al., "State-of-the-art quantum computing simulators: Features, optimizations, and improvements for D-GM," Neurocomputing (2019).

[70] T. Häner and D. S. Steiger, "0.5 petabyte simulation of a 45-qubit quantum circuit," In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 1-10 (2017).

[71] E. Pednault et al., "Breaking the 49-qubit barrier in the simulation of quantum circuits," arXiv:1710.05867 (2017).

[72] E. Pednault et al., "On "Quantum Supremacy"," `https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/`. Accessed 2020.

[73] Z. Y. Chen, et al., "64-qubit quantum circuit simulation," Science Bull. **63**(15), 964-971 (2018).

[74] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (The MIT Press, 2016).

[75] V. Volokitin, A. Liniov, I. Meyerov, M. Hartmann, M. Ivanchenko, P. Hänggi, and S. Denisov, "Computation of the asymptotic states of modulated open quantum systems with a numerically exact realization of the quantum trajectory method ," Phys. Rev. E **96**, 053313 (2017).

[76] A. Liniov, V. Volokitin, I. Meyerov, M. Ivanchenko, and S. Denisov, “Increasing performance of the quantum trajectory method by grouping trajectories,” Comm. Comput. and Inform. Sci. **793**, 136 (2017).
[77] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, “A software methodology for compiling quantum programs, ” Quantum Sci. Technol. **3**, 020501 (2018).
[78] S. Khatri et al., “Quantum-assisted quantum compiling,” Quantum **3**, 140 (2019).
[79] Increasing AI Performance and Efficiency with Intel DL Boost. `https://www.intel.ai/increasing-ai-performance-intel-dlboost/#gs.117qh4`. Accessed 2020.
[80] Intel Unveils New GPU Architecture with High-Performance Computing and AI Acceleration. `https://newsroom.intel.com/news-releases/intel-unveils-new-gpu-architecture-optimized-for-hpc-ai-oneapi/#gs.11dtfx`. Accessed 2020.
[81] Graphcore. `https://www.graphcore.ai/`. Accessed 2020.