QuTree - A Tree Tensor Network Package

Roman Ellerbrock,[1,2,3] K. Grace Johnson,[1,2] Stefan Seritan,[1,2] Hannes Hoppe,[3] J. H. Zhang,[1,2]
Tim Lenzen, Thomas Weike,[3] Uwe Manthe,[3] and Todd J. Martínez[1,2,*]

[1]*Department of Chemistry and The PULSE Institute, Stanford University, Stanford, CA USA 94305*
[2]*SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA USA 94025*
[3]*Bielefeld University, Universitätsstraße 25, 33615 Bielefeld, Germany*

[*]corresponding author
email: toddjmartinez@gmail.com

Abstract: We present QuTree, a C++ library for tree tensor networks approaches. QuTree provides class structures for tensors, tensor trees and related linear algebra functions that facilitate the fast development of tree tensor network approaches such as the multilayer multiconfigurational time dependent Hartree (MCTDH) approach or the density matrix renormalization group (DMRG) approach and its various extensions. We investigate the efficiency of relevant tensor and tensor network operations and show that the overhead for managing the network structure is negligible, even in cases with a million leaves and small tensors. QuTree focuses on providing simple, high-level routines while retaining easy access to the backend to facilitate novel developments. We demonstrate the capabilities of the package by computing eigenstates of coupled harmonic oscillator Hamiltonians and performing random circuit simulations on a virtual quantum computer.

I. Introduction

The accurate quantum mechanical description of multi-particle systems is a fundamental challenge in chemical physics and theoretical chemistry. In the past decades, the multiconfigurational time-dependent Hartree (MCTDH) approach[1-2] and its multilayer extension[3-4] have emerged as important methods for quantum dynamics. In the multilayer MCTDH approach, a quantum state is represented by tree tensor networks and the Dirac-Frenkel variation principle provides the corresponding equations of motion. Detailed MCTDH calculations have been carried out to study reactive processes in the gas phase or spectra of isolated molecules, simulating the quantum dynamics using accurate *ab initio* potential energy surfaces (PESs). Benchmark examples of high-dimensional MCTDH calculations employing accurate *ab initio* PESs include the computation of thermal rate constants[5-11] and state-selected reaction probabilities for the $H + CH_4 \rightarrow H_2 + CH_3$ reaction[12-18] and its isotopically substituted analogues,[18-20] the photo-detachment spectrum of FCH_4^-,[21] the infrared spectrum of the Zundel cation,[22-25] and the tunneling splitting in the ground[26-30] and vibrationally excited states[27, 30-31] of malonaldehyde. Photoinduced dynamics in vibronically coupled systems have been simulated using realistic, ab initio-based model Hamiltonians.[32-35] MCTDH has also been applied to various models describing condensed matter systems.[36-37] Very recently, the multilayer MCTDH framework was extended to study large-scale circuit-based quantum computing.[38] For a more comprehensive overview of applications we refer to reviews on MCTDH.[39-40]

Independently, the density matrix renormalization group (DMRG) approach[41-42] and related extensions have been developed by the physics community using similar tensor network ideas to represent the wavefunction. DMRG was originally used to study time-independent properties of 1D quantum lattices and was later extended to study time-dependent problems.[43] Prominent benchmark examples of DMRG include studying static properties of strongly correlated Hamiltonians such as the Heisenberg, *t-J* and Hubbard models and dynamical properties of eigenstates, such as dynamical structure functions or frequency-dependent conductivities.[44-47] The DMRG approach was extended to employ more general tensor networks which facilitate simulations beyond 1D lattices with nearest neighbor interactions. The correlation of 2D lattices is more efficiently described by the projected entangled-pair states (PEPS) approach which incorporates 2D connectivity into the wavefunction ansatz.[48] Tree tensor networks provide a hierarchical wavefunction ansatz that has been used extensively to deal with 1D critical systems

and 2D systems.[49] The multiscale entanglement renormalization ansatz (MERA) adds so-called distanglers, which are unitary operators that account for entanglement between neighboring sites.[50] For a comprehensive overview of tensor network approaches developed in the physics community including recent applications to machine learning, see recent reviews on tensor network states.[51]

Quantum lattice Hamiltonians are naturally expressed as a compact sum-of-products or matrix product operator. Unfortunately, few-term decompositions of this form are not always obviously possible for accurate and realistic Hamiltonians used in rigorous nuclear dynamics simulations. Nonlinear internal coordinate systems are employed to decouple vibrational from rotational motion and to minimize correlation between coordinates. The resulting kinetic energy operators often consist of hundreds of terms, even for small molecules.[13, 52] Constructing precise, high-dimensional potential energy surfaces (PESs) is even more difficult and evolved as a separate field of research. PESs of low-dimensional or rigid systems can be represented using specifically chosen ansätzes. Accurately describing large amplitude motion in high-dimensional systems, however, requires more sophisticated methods. The permutationally invariant polynomial approach exploits the symmetry of the system.[53] Alternatively, modified Shepard interpolation which interpolates between local second order Taylor expansions has also been used successfully.[54] Recently, PESs have been constructed using artificial neural networks.[55]

The correlation discrete variable representation (CDVR) provides integrals over PESs for tree tensor network states.[4, 56-57] The CDVR is a quadrature that uses tree-structured grids and a correction scheme that is reminiscent of Smolyak's quadrature.[58] This scheme has enabled various state-of-the art calculations on accurate PESs.[39] Recently, a nonhierarchical CDVR approach with drastically improved scaling and comparable accuracy has been reported.[59] An alternative to the quadrature is provided by the POTFIT algorithm which allows casting arbitrary functions into a tree-structured operator.[60-61] Recently, the use of collocation schemes has been explored as an alternative to quadrature.[62] In this approach, the wavefunction is precisely known at certain grid points and the Schrödinger equation is solved only there. While this approach avoids the need for integrals, only some properties of the system are accessible, e.g. the energy of the wavefunction.

The CDVR approach is an excellent example of how extending tensor networks to try new ideas can break preexisting structures in equations and their implementation. CDVR requires a series of new backend routines that become the bottleneck of the tensor network calculation. Some structures of typical tensor network implementations (like the graph and use of tensors) can be

preserved in the CDVR approach. However, others (like the kind of tensor operations and the order in which nodes in the network are accessed) must be changed. Arriving at a working implementation of such ideas can take months, even for highly skilled numerical programmers. The commitment, time and effort required for implementing new ideas poses a huge barrier that limits further developments. QuTree is designed to reduce this barrier and accelerate future developments. It provides reusable template class structures that can be employed or excluded as needed to access only the required functionality. Its backend routines can be easily accessed and extended within the same programming language and codebase. Furthermore, QuTree is comparably small, and comes with documentation and tutorials.

There already exist tensor network libraries in both the chemistry and physics communities. In the chemistry community, Heidelberg's MCTDH package[63] and MCTDH-X[64] (among others) provide implementations of the equations of motion of MCTDH and efficient integrators. They also provide schemes that facilitate simulation of systems provided as potential energy surfaces, such as POTFIT[65] and multigrid POTFIT[61] or CDVR.[4, 56, 59] Such schemes are needed for the efficient evaluation of matrix elements, but often comprise a bottleneck for chemical systems. The MCTDH packages also provide various underlying utilities like orthogonal polynomial basis sets, FFT grids, and simultaneous diagonalization schemes. In the physics community, software for DMRG, TTNS, MERA, PEPS, and other tensor network types of wavefunctions includes iTensor,[66] Google's tensor network library,[67] and ExaTN.[68] These codes often provide solvers for eigenstates and propagating wavepackets of lattice Hamiltonians, simulating quantum circuits, and various utilities for system-specific observables.

There also exist general tensor libraries that provide APIs for tensors which we categorize in four layers. The first layer contains pure hardware-optimized libraries like OpenBLAS,[69] MKL,[70] cuBLAS,[71] and ScaLAPACK[72] that are written in low-level languages such as assembler, FORTRAN or CUDA. The second layer includes libraries written in C++ that provide APIs around layer 1 libraries and add universal function signatures for differing types and hardware agnosticism. Examples include Magma,[73] BLAS++/LAPACK++,[74-75] and the TiledArray framework.[76] Some libraries also add tensor classes like pytorch's underlying libtensor.[77] Layer 3 consists of python libraries that operate on a higher level of abstraction and are more accessible to users via built-in package managers. Internally, these libraries map to layer 1 or 2 libraries to execute time-sensitive tasks. Popular examples include pytorch,[77] tensorflow,[78] and jax.[79] Layer 4

unifies layer 3 APIs into a single framework that allows to use layer 1-3 solutions via the same API. To the best of our knowledge, at this time Tensorly[80] is the only library that falls into this category.

QuTree is a unique platform for developments on the interface between quantum nuclei dynamics and quantum computing. It provides native tensor APIs and C++ classes and functions for managing tree tensor networks with minimal overhead. As we demonstrate in this paper, this enables simulations of extremely large networks (up to a million leaves). QuTree delivers TTN classes that are as performant as desired for layer 2 solutions of tensor interfaces. In addition, it provides a wide range of algorithms related to TTNs that allow simulating both chemical systems and quantum computing applications. It provides equations of motion solvers for MCTDH, the CMF2 integrator,[81] a state-averaged eigenstate finder, the nonhierarchical-CDVR approach with grid-optimized SPFs,[59, 82] time-dependent discrete variable representation (TDDVR),[56] along with the required weighted-simultaneous diagonalization algorithm (WSD),[59] various base-integrators, basis types, and various potential energy surface interfaces and sum-of-product operators for chemical systems. The nonhierarchical-CDVR and grid-optimized SPFs are, to the best of our knowledge, exclusively implemented in QuTree; other algorithms have for the first time been implemented in C++ (TDDVR, WSD). Beside the wide range of algorithms required for describing the quantum dynamics of nuclei, QuTree facilitates simulations of quantum circuits.[83] Having a package like QuTree that provides functionalities for describing nuclear quantum dynamics and quantum computing will be important for future developments at the interface between the fields.

The work is structured as follows: in Sec. II, we review fundamentals of tree tensor networks. In Secs. III and IV, we discuss the implementation and performance of QuTree, respectively. In Sec. V we provide an example application, simulating 2D random quantum circuits. We close with concluding remarks and an outlook for future developments.

II. Theory

We begin with a review of the fundamental theory of tensor operations and tensor networks. The expressions in this Section were derived in the context of MCTDH, DMRG and their extensions. While the multilayer MCTDH community exclusively uses equations to represent expressions, the DMRG and tensor network state community has integrated so-called tensor network diagrams to facilitate expressions. Recently, Larsson started to unify the notation and

diagrammatic representation used in the DMRG and MCTDH communities.[84] We will continue this effort and review both formalisms simultaneously. In order to keep the paper general and concise, we will introduce tensor networks without mapping them to physical, chemical or machine learning applications.

II.1 Tensors

Let **A** be a d^{th} order tensor with the set of dimensions $\mathbf{n} = (n_1, \dots, n_d)$. The tensor elements will be addressed using the multidimensional index $I = (i_1, \dots, i_d)$ and will be denoted by $A_I = A_{i_1,\dots,i_d}$. Vectors and matrices are special cases of first and second order tensors, respectively. Tensors are diagrammatically depicted by a vertex with attached edges, one for each of its indices. The dimensions are sometimes written next to the corresponding edge. Fig. 1a shows exemplary diagrammatic representations of tensors of different order.

If the indices of two tensors share the same upper bound, a summation over the shared index can be performed. This is diagrammatically depicted by connecting the edges of tensors that correspond to the chosen indices (see, e.g., Fig. 1b for the tensor network depiction of a matrix vector product). Thus, they are named tensor network diagrams. This opens a plethora of possible tensor operations of which some take a special role in tensor network theory. We will review them in the following.

We define a tensor hole contraction between two tensors, **A** and **B** as

$$h_{i_k j_k} = \sum_{i_1,\dots,i_{k-1},i_{k+1}\dots,i_d} A^*_{i_1\dots i_{k-1} i_k i_{k+1}\dots i_d} \cdot B_{i_1\dots i_{k-1} j_k i_{k+1}\dots i_d} \tag{1}$$

where **A** and **B** are both d^{th} order tensors and summation is performed over all indices but the hole-index *k*. The asterisk (*) indicates complex conjugation. We use the brief notation $\boldsymbol{h} = (\boldsymbol{A}, \boldsymbol{B})_{(k)}$ where *k* indicates the hole-index. Here and in the following, bold upper-case letters will denote tensors and bold lower-case letters will denote matrices. The corresponding diagrammatic representation of this product for an example where **A** and **B** are 3rd order tensors is shown in Fig. 1c.

Another important product is the matrix tensor product

$$B_{i_1\dots i_d} = \sum_{j_k} h_{i_k j_k} \cdot A_{i_1\dots i_{k-1} j_k i_{k+1}\dots i_d} \tag{2}$$

which we denote as $\boldsymbol{B} = \boldsymbol{h^k} \cdot \boldsymbol{A}$. The corresponding diagram is depicted in Fig. 1d for an example where A is a 3rd order tensor.

In applications, it is often desirable to reshape tensors into vectors, matrices, or tensors with different dimensions. We formalize this by introducing *compound indices*. Let $I = (i_1, \dots, i_d)$ be a compound index with upper bounds $N = (n_1, \dots, n_d)$. We can address tensor elements as a vector via A_I or as a tensor via $A_{i_1 \dots i_d}$. We now choose a different set of indices $J = (j_1, \dots j_e)$ with upper bounds $M = (m_1, \dots, m_e)$. If the upper bound of both indices match ($\sum_i m_i = \sum_i n_i$), then we can address the same tensor elements of **A** via $A_{j_1 \dots j_e}$. The most common use of this is to reshape high-order tensors into vectors or matrices (or vice versa).

Using tensor reshaping, we can generalize matrix and vector operations to tensors of arbitrary order by reshaping $\boldsymbol{A}$ into a matrix (or vector), performing the operation and reshaping the compound indices back to the original ones. We can, for example, extend rank-revealing matrix factorizations to tensors via $A_{i_1 \dots i_d} = A_{j_1 j_2}$ where $j_1 = (i_{k_1}, i_{k_2}, \dots)$ is some compound index and $j_2 = I \setminus j_1 = (i_{m_1}, i_{m_2}, \dots)$ is the compound index of the remaining indices. Then, we factorize the matrix and map the indices back: $A_{j_1 j_2} = \sum_l^n U_{j_1 l} \cdot V_{j_2 l} = \sum_l^n U_{i_{k_1} i_{k_2} \dots l} \cdot V_{i_{m_1} i_{m_2} \dots l}$. Common operations that appear in the context of tensor networks are tensor-tensor products (Eqs. 1 and 2 are examples of such products), tensor transposes, and tensorized factorizations like the QR-, LU-, and SVD-decompositions. A diagrammatic representation of a factorization of a 3rd order tensor is depicted in Fig. 1e. While tensor products replace two tensors in a tensor network by one, tensor factorizations do the opposite. Performing factorizations and tensor reshapes recursively on a tensor results in a tensor tree with a tree topology that depends on the order of reshapes and factorizations.

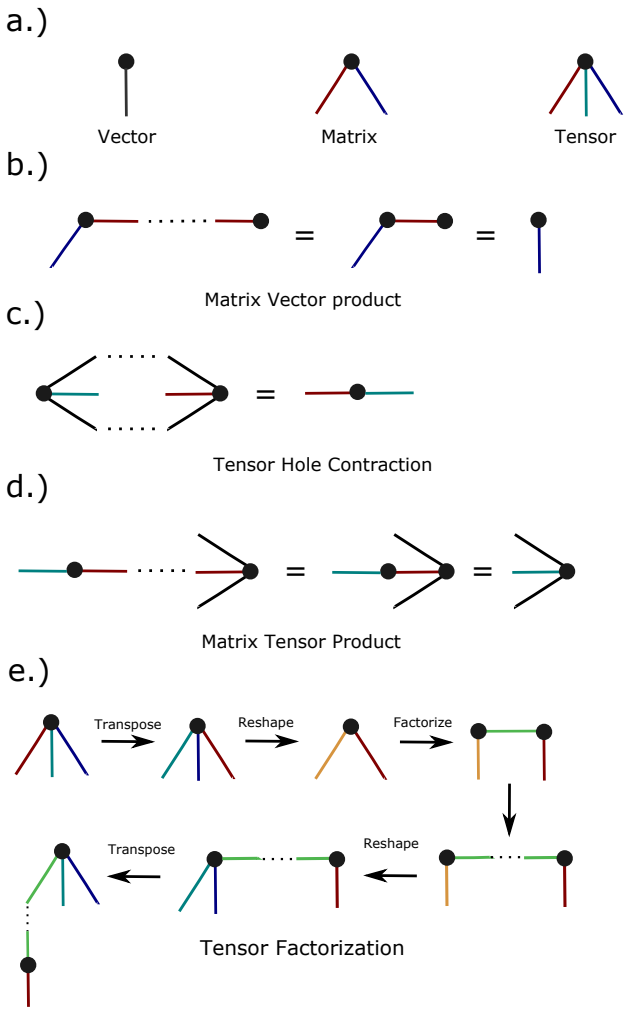

Fig. 1: Graphical representation of vectors, matrices, and tensors (a) and some important examples of tensor operations (b-d), including matrix vector product (b), tensor hole contraction (c), matrix tensor product (d), and tensor factorization (e). Note that we use 3rd order tensors but these can be replaced by any order.

II.2 Tree Tensor Network States

We will now define tree tensor network states (TTNS) which are a class of tensor network representations of quantum states or, more generally, of a vector in a high-dimensional vector space. We will start with an exemplary system to present the idea before introducing the general notation for arbitrary trees.

Consider a system with four distinguishable particles, described by primitive basis functions $|e_{i_1}^1\rangle, |e_{i_2}^2\rangle, |e_{i_3}^3\rangle, |e_{i_4}^4\rangle$, respectively. The standard wavefunction ansatz for the system reads

$$|\Psi\rangle = \sum_{i_1,i_2,i_3,i_4}^{N_1,N_2,N_3,N_4} A_{i_1 i_2 i_3 i_4} \cdot \left|e_{i_1}^1\right\rangle \cdot \left|e_{i_2}^2\right\rangle \cdot \left|e_{i_3}^3\right\rangle \cdot \left|e_{i_4}^4\right\rangle, \tag{3}$$

where N_1, N_2, N_3, and N_4 primitive basis functions are required for each particle. If we extend this system to f particles, this ansatz scales exponentially as N^f regardless of the amount of information in the expansion tensor $\boldsymbol{A}$. We will divide the system into two subsystems: system A contains particles 1 and 2, and system B contains particles 3 and 4. We introduce optimized functions for subsystems A and B via

$$\left|\varphi_{i_A}^A\right\rangle = \sum_{i_1,i_2}^{n_A} A_{i_A i_1 i_2}^A \cdot \left|e_{i_1}^1\right\rangle \cdot \left|e_{i_2}^2\right\rangle, \qquad \left|\varphi_{i_B}^B\right\rangle = \sum_{i_3 i_4}^{n_B} A_{i_B i_3 i_4}^B \cdot \left|e_{i_4}^3\right\rangle \cdot \left|e_{i_4}^4\right\rangle. \tag{4}$$

The number of required functions n_A and n_B is between 1 and $N_1 \cdot N_2$ or $N_3 \cdot N_4$, respectively. The total wavefunction is expressed as a direct product of the optimized functions

$$\begin{aligned} \left|\Psi^{4-\text{particle}}\right\rangle &= \sum_{i_A i_B}^{n_A n_B} A_{i_A i_B}^C \cdot \left|\varphi_{i_A}^A\right\rangle \cdot \left|\varphi_{i_B}^B\right\rangle \\ &= \sum_{i_A i_B} A_{i_A i_B}^C \sum_{i_1 i_2}^{N_1,N_2} A_{i_A i_1 i_2}^A \cdot \left|e_{i_1}^1\right\rangle \cdot \left|e_{i_2}^2\right\rangle \cdot \sum_{i_3}^{N_3 N_4} A_{i_B i_3 i_4}^B \left|e_{i_4}^3\right\rangle \cdot \left|e_{i_4}^4\right\rangle . \end{aligned} \tag{5}$$

The tensors $\boldsymbol{A}^A, \boldsymbol{A}^B, \boldsymbol{A}^C$ can be obtained by performing a factorization on $\boldsymbol{A}$ as described above. We can extend this idea to describe twice as many particles by reinterpreting the 4-particle wavefunction as partial basis functions and by adding an index i_C to the tensor that addresses the basis functions $\left|\varphi_{i_C}^C\right\rangle = \sum_{i_A i_B}^{n} A_{i_A i_B i_C}^C \cdot \left|\varphi_{i_A}^A\right\rangle \cdot \left|\varphi_{i_B}^B\right\rangle$. The functions describe what is now the subsystem comprising the first four particles. We introduce analogous functions $\left|\varphi_{i_D}^D\right\rangle$ for the next four particles and then combine these four-particle functions into an 8-particle wavefunction $\left|\Psi^{8-\text{particle}}\right\rangle = \sum_{i_C i_D}^{n} A_{i_C i_D}^E \cdot \left|\varphi_{i_C}^C\right\rangle \left|\varphi_{i_D}^D\right\rangle$.

This divide and conquer approach of representing the wavefunction is shown by the tree in Fig. 2, where each of the expansion tensors $\boldsymbol{A}^A, \boldsymbol{A}^B, \boldsymbol{A}^C$, ... corresponds to one node in the tree. If the dimensions of the tensors scale moderately as the system size is increased, this approach can break the exponential scaling of the standard wavefunction ansatz. In the following, we will derive expressions for general trees and introduce a notation to address tensors locally, i.e., by only using local connectivity of a node or edge.

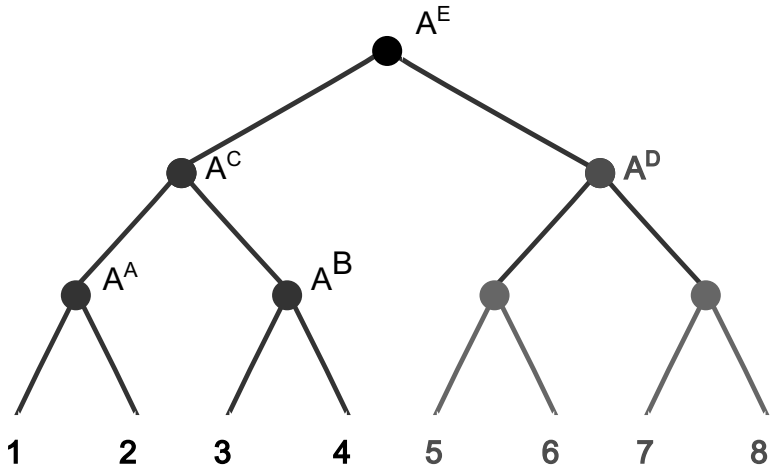

Fig. 2: Diagrammatic representation of an eight-particle system described by a tree tensor network.

Let $\boldsymbol{t} = \{v, e\}$ be a tree graph with nodes v and directed edges e, and let $p \in \boldsymbol{v}$ be an exemplary node. Fig. 3 shows a diagram for an exemplary tree with associated names of some vertices and edges depicting a new notation. We denote the root node of the tree as r and the k$^{\text{th}}$ neighbor of node p as p, k. By convention, $p, 0$ will correspond to the parent of a node and all other p, k to the children of that node. The edge pointing from p to the k^{th} neighbor will be denoted by $p \rightarrow k$ and the edge pointing in the opposite direction will be denoted as $p \leftarrow k$. Note that edges can be addressed relative to two different nodes. Let $q = p, k$ be a child of p, then $p \rightarrow k = q \leftarrow 0$ and $p \leftarrow k = q \rightarrow 0$. Fig. 3a shows the notation for a global tree and Fig. 3b shows it for a local node.

By defining one node as the root, we can sort the nodes into layers by their distance to the root. In the context of multilayer MCTDH, the root node has historically been depicted at the top. Thus, the root is also referred to as the top layer and nodes adjacent to leaves are referred to as the bottom layer. Most equations that include tree tensor networks require sweeping over nodes in the tree in a specific order. We call a sweep in which we proceed from the root node towards the leaves as a top-down sweep and the inverse sweep as a bottom-up sweep.

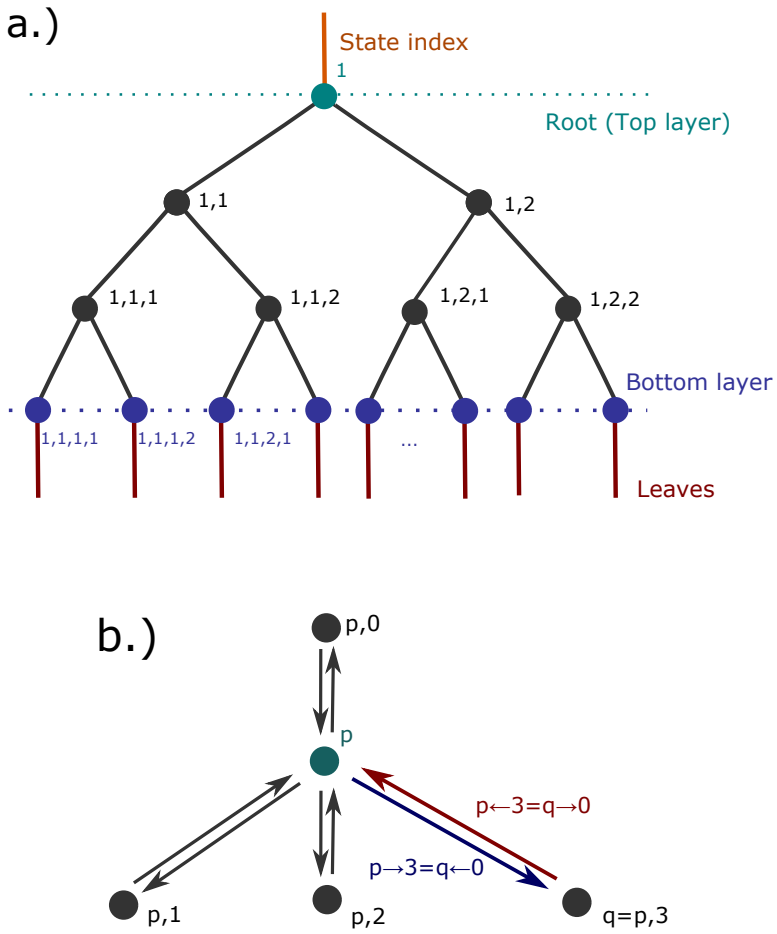

Fig. 3: Tensor network diagram notation. Panel a) shows nodes addressed globally. Undirected edges are shown for simplicity. Leaves are marked in red, bottom layer nodes in blue and the top (root) node in green. Panel b) shows adjacent nodes and directed edges addressed locally for an exemplary node p.

For a given tree ***t***, we define the tree tensor network representation of a set of quantum states $|\Psi_{i_0}\rangle$ by

$$|\Psi_{i_0}\rangle = \sum_{i_1 \ldots i_d} A^1_{i_0,\ldots,i_d} \cdot |\varphi^{1\leftarrow 1}_{i_1}\rangle \cdot \ldots \cdot |\varphi^{1\leftarrow d}_{i_d}\rangle \tag{6}$$

where $|\varphi^{1\leftarrow k}_{i_k}\rangle$ are basis functions for each child of the root node. They can alternatively be referenced with respect to the child node via $|\varphi^{1\leftarrow k}_{i_k}\rangle = |\varphi^{(1,k)\rightarrow 0}_{i_0}\rangle$. The basis functions are then defined recursively via

$$|\varphi^{p\rightarrow 0}_{i_0}\rangle = \sum_{i_1 \ldots i_d} A^p_{i_0,\ldots,i_d} \cdot |\varphi^{p\leftarrow 1}_{i_1}\rangle \cdot \ldots \cdot |\varphi^{p\leftarrow d}_{i_d}\rangle. \tag{7}$$

For $p = 1$ we obtain the definition for the first layer of functions shown in Eq. 6. The number of children d can, in principle, differ between different nodes in which case d is replaced by d_p. In this section, we restrict the discussion to a constant d to keep the notation simple. This recursion ends at the leaves where the recursive basis functions are replaced by primitive basis functions

$$|\varphi_n^l\rangle = |e_n^l\rangle, \tag{8}$$

where $|e_n^l\rangle$ are primitive basis functions. The primitive basis functions are chosen according to the application, e.g. harmonic oscillator eigenfunctions, Legendre polynomials or equidistant grid points (so-called FFT grids) in quantum dynamics applications with multilayer MCTDH. Spin eigenfunctions are used as primitive basis functions in many physical model systems as well as quantum computing and machine learning applications.

The set of all tensors used in the wavefunction expansion is called a tree tensor network, or simply a tensor tree. If a tree tensor network is used to represent a quantum state, this state is called a tree tensor network state (TTNS). The recursive basis functions $\left|\varphi_n^{p\leftarrow k}\right\rangle$ (Eq. 7) are called single-particle functions (SPFs) in the MCTDH literature. A set of SPFs is orthonormal if $\left\langle\varphi_i^{p\leftarrow k}\middle|\varphi_j^{p\leftarrow k}\right\rangle = \delta_{ij}$. In the physics literature, the corresponding tensors are called isometric. Each tensor $\boldsymbol{A}^p$ in the tree has dimensions $\boldsymbol{n}^p = (n_0^p, \dots n_d^p)$ and we call the set of all dimensions in the tree the *tensor tree shape*. In the context of DMRG and its extensions, the specific dimensions are called *bond dimensions*. In the MCTDH literature, these are typically called the *number of SPFs* and depicted as numbers next to edges in tree tensor network diagrams. The first tensor index, i_0, is reserved for the edge pointing to the parent.

The SPFs are contracted basis functions that are attributes of upward edges. Analogous functions corresponding to downward edges are single hole functions (SHFs), which read

$$\left|\psi_{i_k}^{p\to k}\right\rangle = \sum_{I(k)} A_I^p \cdot \left|\psi_{i_0}^{p\leftarrow 0}\right\rangle \cdot \left|\varphi_{i_1}^{p\leftarrow 1}\right\rangle \cdot \dots \left|\varphi_{i_{k-1}}^{p\leftarrow [k-1]}\right\rangle \left|\varphi_{i_{k+1}}^{p\leftarrow [k+1]}\right\rangle \cdot \left|\varphi_{i_d}^{p\leftarrow d}\right\rangle. \tag{9}$$

Like SPFs, the SHFs are defined recursively but in a top-down recursion. At the root node, the SHFs vanish from the equation. SHFs are different from SPFs in two major ways: they are attributes of downward edges, and they are, in general, not orthonormal when defined as in Eq. 9. Instead, the overlap of two SHFs results in the element of a reduced density matrix $\left\langle\psi_{i_k}^{p\to k}\middle|\psi_{j_k}^{p\to k}\right\rangle = \rho_{i_k j_k}^{p\to k}$ of the subtree. SHFs can be normalized by performing a tensorized QR-decomposition of the tensor in Eq. 9. Since normalized SHFs are effectively equivalent to the

downward-defined SPFs, we denote them by $\left|\varphi_{i_k}^{p\to k}\right\rangle$ and they obey $\left|\psi_{i_k}^{p\to k}\right\rangle = \sum_{j_k} \rho_{i_k j_k}^{p\to k,\frac{1}{2}} \left|\varphi_{j_k}^{p\to k}\right\rangle$. If a different node in the tree is assigned to be the root, some upward edges become downward edges and vice versa. Consequently, the corresponding SHFs and SPFs get swapped. If both are normalized in the same way, the wavefunction is invariant under this transformation. This has implications for the convergence of MCTDH.[85]

Using SPFs and SHFs, we finally arrive at two central ways to express a TTNS: an expansion from an arbitrary node and an expansion from an arbitrary edge. The expansion from a node p reads

$$|\Psi\rangle = \sum_I A_I^p \cdot \left|\psi_{i_0}^{p\leftarrow 0}\right\rangle \left|\varphi_{i_1}^{p\leftarrow 1}\right\rangle \cdot \ldots \cdot \left|\varphi_{i_d}^{p\leftarrow d}\right\rangle = \sum_I \tilde{A}_I^p \cdot \left|\varphi_{i_0}^{p\leftarrow 0}\right\rangle \cdot \ldots \cdot \left|\varphi_{i_d}^{p\leftarrow d}\right\rangle. \tag{10}$$

Here, $\left|\varphi_{i_0}^{p\leftarrow 0}\right\rangle$ is a normalized SHF and is only distinguishable from SPFs by being associated to downward edges. We call $\tilde{A}_I^p = \sum_{j_k} \rho_{i_k j_k}^{p\leftarrow k,\frac{1}{2}} A_{i_0 \ldots i_{k-1} i_k i_{k+1} \ldots i_d}^{p\leftarrow k}$ density weighted tensors. Note that, given a density-weighted tensor, the choice of normalized SHF is not unique if the reduced density matrix is singular. While the choice does not change the quality of the representation of the wavefunction, it will determine the search space in objective functions like the choice of unoccupied SPFs. Using normalized and unnormalized SHFs, the expansion from an arbitrary edge reads

$$|\Psi\rangle = \sum_{i_k} \left|\varphi_{i_k}^{p\leftarrow k}\right\rangle \cdot \left|\psi_{i_k}^{p\to k}\right\rangle = \sum_{i_k j_k} \rho_{i_k j_k}^{p\to k,\frac{1}{2}} \left|\varphi_{i_k}^{p\leftarrow k}\right\rangle \cdot \left|\varphi_{j_k}^{p\to k}\right\rangle. \tag{11}$$

In Fig. 4 we show how a wavefunction can be split into SPFs and SHFs for an edge in an exemplary tree. SPFs and SHFs correspond to complementary subtrees.

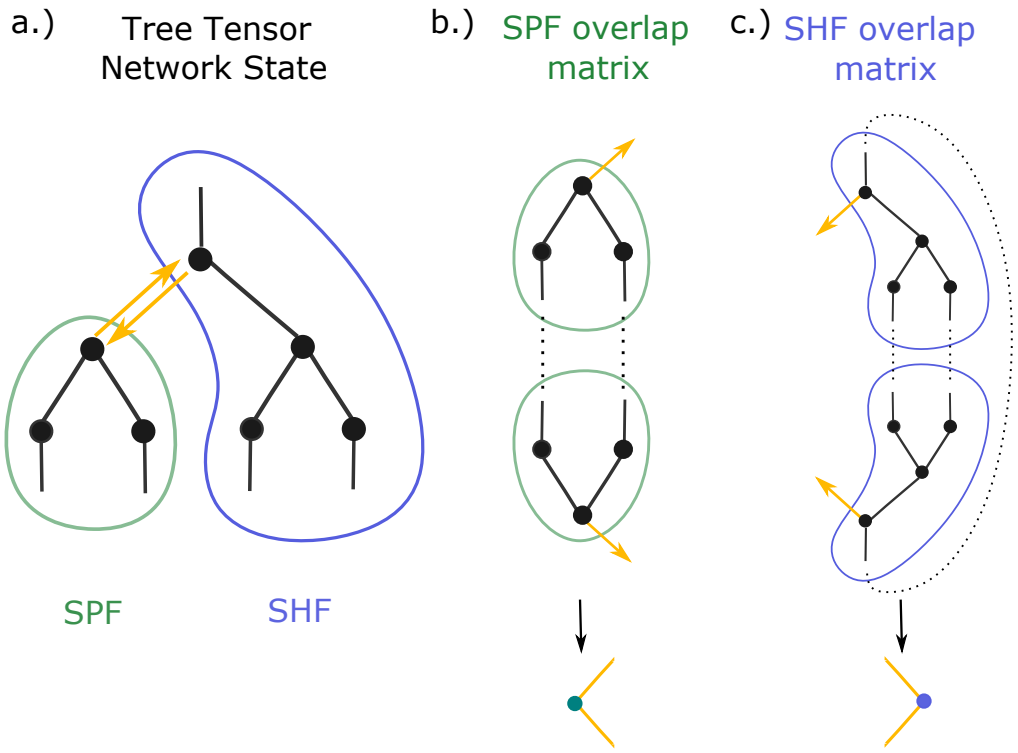

Fig. 4: Panel a) shows splitting a tree tensor network into SPFs and SHFs at a highlighted edge. Panels b) and c) show how SPF overlap matrices (green) and reduced density matrices (blue) are calculated, respectively.

The tensor network ansatz is typically applied to a given problem by choosing a suitable objective function. The objective function requires applying an operator (e.g. a Hamiltonian) to the TTNS, thus the operator must be represented in TTNS form. Stationary points of these objective functions are found using the variational principle or gradient methods. This typically results in in expressions including dot-products between tree tensor network wavefunctions and dot-products between operator representations. In the following, we will review frequently occurring expressions: dot products, reduced density matrices, and matrix- and mean-field matrix representations of operators.

The dot-product between two tree tensor network states $|\Psi\rangle$ and $|\tilde{\Psi}\rangle$ is evaluated via

$$\langle\Psi|\tilde{\Psi}\rangle = \sum_{I,J} A_I^{1*} \cdot \langle\varphi_{i_1}^{1\leftarrow 1}|\tilde{\varphi}_{j_1}^{1\leftarrow 1}\rangle \cdot \ldots \cdot \langle\varphi_{i_1}^{1\leftarrow d}|\tilde{\varphi}_{j_1}^{1\leftarrow d}\rangle \cdot \tilde{A}_J^1, \tag{12}$$

where the SPF overlaps $\left\langle\varphi_{i_k}^{(p,0)\leftarrow k}\middle|\tilde{\varphi}_{j_k}^{(p,0)\leftarrow k}\right\rangle = \left\langle\varphi_{i_0}^{q\rightarrow 0}\middle|\tilde{\varphi}_{j_0}^{q\rightarrow 0}\right\rangle$ are calculated recursively

$$\left\langle \varphi_{i_0}^{p\to 0} \middle| \tilde{\varphi}_{j_0}^{p\to 0} \right\rangle = \sum_{I(0),J(0)} \tilde{A}_I^{p*} \cdot \left\langle \varphi_{i_1}^{p\leftarrow 1} \middle| \tilde{\varphi}_{j_1}^{p\leftarrow 1} \right\rangle \cdot \ldots \cdot \left\langle \varphi_{i_d}^{p\leftarrow d} \middle| \tilde{\varphi}_{j_d}^{p\leftarrow d} \right\rangle \cdot \tilde{A}_J^p, \tag{13}$$

where $I(k) = i_0 \ldots i_{k-1} i_{k+1} \ldots i_d$ is the compound index with hole in k. The equation is solved by a sequence of matrix tensor products, Eq. 2, followed by a tensor hole contraction, Eq. 1. The set of all local dot-products $\left\langle \varphi_{i_k}^{q\leftarrow k} \middle| \tilde{\varphi}_{j_k}^{q\leftarrow k} \right\rangle$ in $\boldsymbol{t}$ form a *matrix tree,* which has the same tree structure as tree tensor network states but stores matrices on edges instead of tensors at nodes. The tensor network diagram corresponding to the overlaps of SPFs, Eq. 13, is depicted in Fig. 5. The dot product between tree tensor network states is evaluated in a bottom-up swipe.

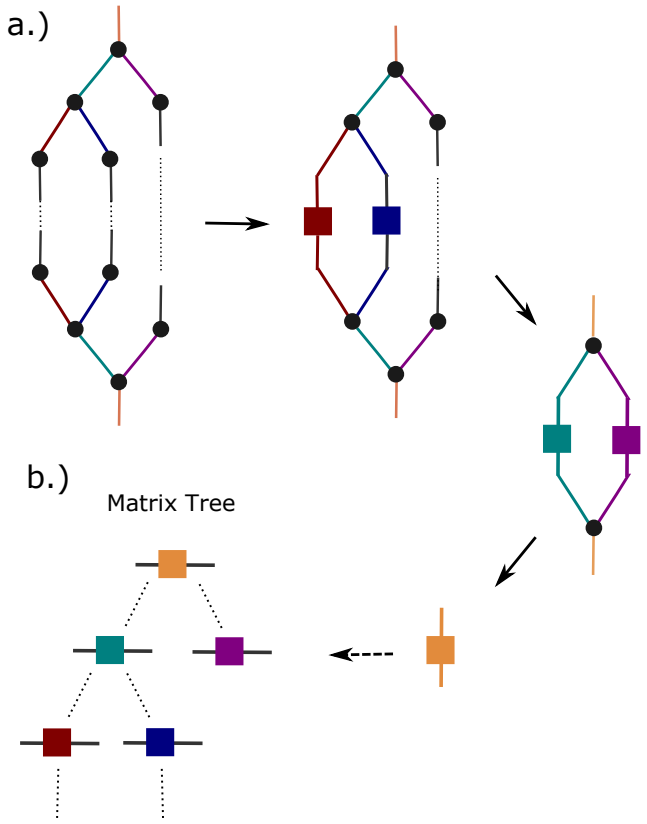

Fig. 5: Tensor diagrams depicting the steps required to calculate the dot-product between two tree tensor network states (Eqs. 12-13). Panel a) shows the desired dot-product and the required steps which each yield SPF overlap matrices (Eq. 13), represented here as colored squares. Panel b) depicts all matrices resulting from the workflow organized as a matrix tree.

The overlaps of SHFs read

$$\langle \psi_{i_k}^{p\to k} | \tilde{\psi}_{j_k}^{p\to k} \rangle = \sum_{I(k)} A_I^{*p} \cdot \tilde{B}^p_{i_0 \dots i_{k-1} j_k i_{k+1} \dots i_d} \tag{14a}$$

with

$$B_I = \sum_{J(k)} \langle \psi_{i_0}^{p\leftarrow 0} | \tilde{\psi}_{j_0}^{p\leftarrow 0} \rangle \cdot \ldots \cdot \langle \psi_{i_{k-1}}^{p\leftarrow [k-1]} | \tilde{\psi}_{j_{k-1}}^{p\leftarrow [k-1]} \rangle \cdot \langle \psi_{i_{k+1}}^{p\leftarrow [k+1]} | \tilde{\psi}_{j_{k+1}}^{p\leftarrow [k+1]} \rangle \cdot \ldots \cdot \langle \psi_{i_d}^{p\leftarrow d} | \tilde{\psi}_{j_d}^{p\leftarrow d} \rangle \cdot \tilde{A}_J \tag{14b}$$

First, the overlap of SPFs, Eq. 13, are calculated before Eq. 14 is solved in a top-down sweep. The overlaps of SHFs can, like those of SPFs, be attributed to edges and are solved analogously to SPF overlaps. Note that setting $k = 0$ and using normalized functions in Eq. 14 provides an expression that is equivalent to Eq. 13.

Hamiltonians like lattice models or kinetic energy operators can often be expressed in sum-of-product form, $\widehat{H} = \sum_m c_m \cdot h_m^1 \cdot \ldots \cdot h_m^d$ where $\hat{h}_m^k$ is an operator that acts on a single lattice site or coordinate *k*. Matrix representations of these operators often occur in working equations. To represent the operators, we consider just a single summand $\widehat{M} = h_m^1 \cdot \ldots \cdot h_m^d$ to simplify the notation. The matrix elements of $\widehat{M}$ read

$$\left\langle \varphi_{i_0}^{p\to 0} \middle| \widehat{M} \middle| \tilde{\varphi}_{j_0}^{p\to 0} \right\rangle = \sum_{I(0),J(0)} \tilde{A}_I^{p*} \cdot \left\langle \varphi_{i_1}^{p\leftarrow 1} \middle| \hat{g}^{p\leftarrow 1} \middle| \tilde{\varphi}_{j_1}^{p\leftarrow 1} \right\rangle \cdot \ldots \cdot \left\langle \varphi_{i_d}^{p\leftarrow d} \middle| \hat{g}^{p\leftarrow d} \middle| \tilde{\varphi}_{j_d}^{p\leftarrow d} \right\rangle \cdot \tilde{A}_J^p, \tag{15}$$

where $\hat{g}^{p\leftarrow x} = \hat{h}^{r_1} \cdot \ldots \cdot \hat{h}^{r_x}$ is the product the operators acting only on the subtree below $p \leftarrow x$. The matrices are evaluated recursively, in a bottom-up sweep analogous to Eq. 13. For a leaf *l*, $\left\langle e_i^{p\leftarrow l} \middle| \hat{h}^l \middle| e_j^{p\leftarrow l} \right\rangle$ are matrix elements resulting from representing the operators in the primitive basis and provide the entry point to the recursion. If normalized SPFs are used, the matrix elements of identity operators result in Kronecker deltas and can be omitted. This results in a sparse tree topology.

Working equations feature not only the representations of operators in the SPF basis but also in the SHF basis. If the SHFs are density-weighted, they are referred to as mean-field matrices. Analogous to the overlap of SHFs, we introduce contractions of the matrix representations of $\boldsymbol{g}^p$'s via

$$\left\langle \psi_{i_k}^{p\to k} \middle| \hat{g}^{p\to k} \middle| \tilde{\psi}_{j_k}^{p\to k} \right\rangle = \sum_{I(k)} A_I^{*p} \cdot \tilde{C}^p_{i_0 \dots i_{k-1} j_k i_{k+1} \dots i_d}, \tag{16}$$

with

$$\begin{aligned} &\tilde{C}^p_{i_0 \dots i_{k-1} j_k i_{k+1} \dots i_d} \\ &= \sum_{I(k)} \left\langle \varphi_{i_0}^{p\leftarrow 0} \middle| \hat{g}^{p\leftarrow 0} \middle| \tilde{\varphi}_{j_0}^{p\leftarrow 0} \right\rangle \cdot \dots \cdot \left\langle \varphi_{i_{k-1}}^{p\leftarrow [k-1]} \middle| \hat{g}^{p\leftarrow [k-1]} \middle| \tilde{\varphi}_{j_{k-1}}^{p\leftarrow [k-1]} \right\rangle \\ &\cdot \left\langle \varphi_{i_{k+1}}^{p\leftarrow [k+1]} \middle| \hat{g}^{p\leftarrow [k+1]} \middle| \tilde{\varphi}_{j_{k+1}}^{p\leftarrow [k+1]} \right\rangle \cdot \dots \cdot \left\langle \varphi_{i_d}^{p\leftarrow d} \middle| \hat{g}^{p\leftarrow d} \middle| \tilde{\varphi}_{j_d}^{p\leftarrow d} \right\rangle \cdot \tilde{A}_J^p. \end{aligned} \tag{17}$$

After evaluating the matrix representation in the SPF basis (Eq. 14), the matrix elements in the SHF basis are obtained in a top-down sweep via a sequence of matrix tensor operators followed by a tensor hole contraction. Representations of identity operators can be omitted and result in a sparse tree topology, following the same arguments as for SPFs.

III. Implementation

QuTree provides C++ classes and functions that enable performing common operations on tree tensor networks. The code comes with documentation that explains details of the implementation. Here we will give an overview of the relevant data structures and functions. Fig 6 shows a diagram of central classes in QuTree divided into four sub-modules.

The module *Core* contains classes for tensors, matrices, and vectors and manages corresponding basic linear algebra operations. The function *contraction* performs tensor hole contraction (Eq. 1) and the function *matrixTensor* performs matrix tensor product (Eq. 2). Fig. 7a shows exemplary code for generating tensors, accessing tensor elements, and performing operations involving tensors.

The module *TreeShape* handles classes that manage the graph structure of the tensor network. The class *Tree* manages the network structure and the tensor shapes. It provides bidirectional iterators for nodes and edges: a forward iteration performs a bottom-up sweep, and a backwards iteration performs a top-down sweep. Trees can be constructed a) via input by manually providing the tree structure, dimensions and leaf-specification or b) automatically, by providing the number of leaves (and additional specification). In the second case, either (close-to) balanced trees or maximally unbalanced trees (for matrix product states or tensor trains) can be generated and leaves

can be re-assigned afterwards. The *Leaf* class provides an abstract class interface that can be used to adapt the library to specific applications. The library comes with a set of interfaces for various orthogonal polynomial basis sets, equidistant (FFT) grids, spin basis sets and logic basis sets without further specification.

The module *TreeClasses* contains *TensorTree* and *MatrixTree* classes with both dense and sparse tree topology. The class *NodeAttribute* is a class template that can assign objects (e.g., matrices or tensors) to nodes and edges in the tree. It provides iterators and bracket operators that take a node as input and return the corresponding object. *SparseNodeAttribute* is a template class with similar functionality to *NodeAttribute* but for a subset of the nodes. The class provides sparse iterators that scale asymptotically as the number of nodes touched in the sweep. *TensorTree* represents a tree tensor network (Eqs. 6-7). *MatrixTree* represents corresponding overlap matrices of SPFs and SHFs (Eqs. 12-14). The namespace *TreeFunctions* provides functions to evaluate dot-products between tree tensor networks and calculate reduced density matrices.

The submodule *TreeOperators* provides functions which represent sum-of-product operators and related objects. The class *LeafOperator* represents an operator acting locally on a leaf, e.g. a single-particle operator or local lattice site operator. *MultiLeafOperator* (corresponding to the summands $\widehat{M}$ above) is a product of *LeafOperators*. The *SumOfProductOperator* (corresponding to the sum-of-product forms of $\widehat{H}$ in the previous section) is a sum of *MultiLeafOperator*s. *TreeFunctions::represent* evaluates the representation of an operator in SPF basis (Eq. 14) and *TreeFunctions::contract* calculates the representations in SHF basis (Eq. 15).

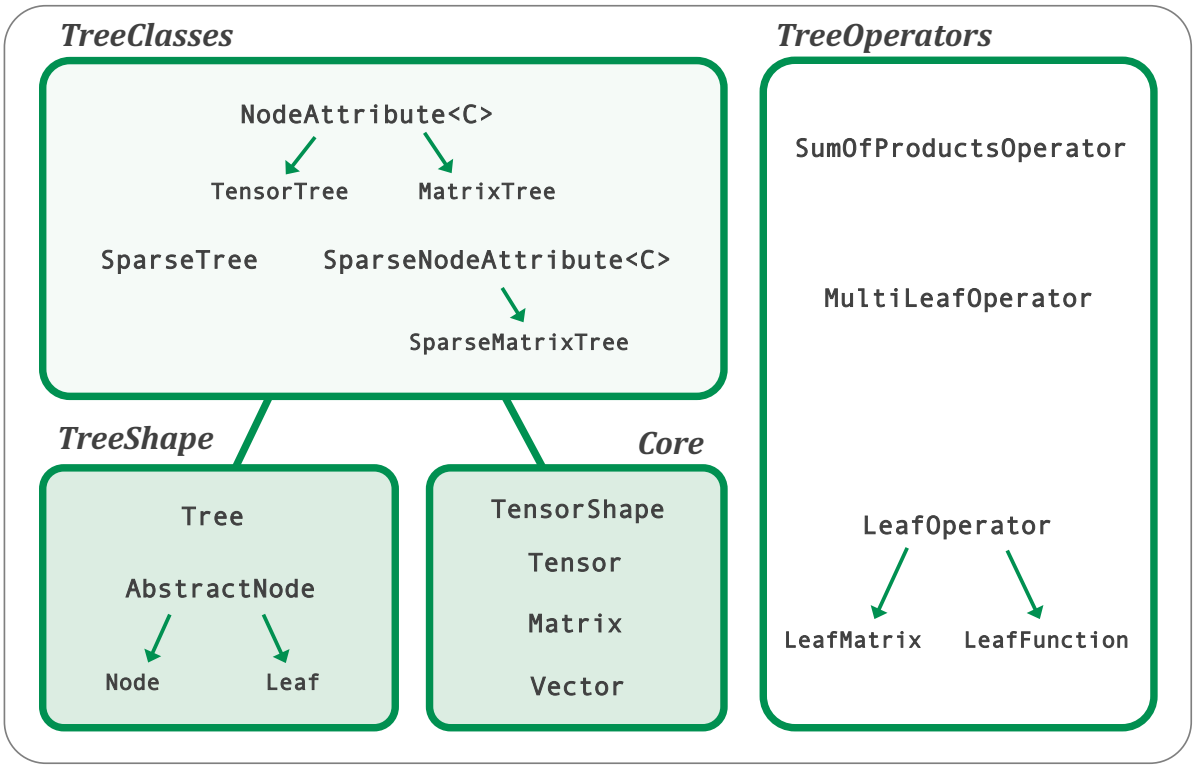

Fig. 6: Class diagram of the central classes in QuTree. Arrows indicate inheritance relationship and boxes show groupings into submodules. The classes shown in the modules Core, TreeShape and TreeOperators are type templates.

Fig. 7b shows exemplary code to generate a Tree with tensor dimensions corresponding to every node in the tree. The tree generator automatically generates a binary close-to balanced tree structure where the number of primitive basis functions, basis type and bond dimension are specified. Similar functions exist to generate a tensor train, or MPS. By default, the tree tensor network corresponding to the tree structures are generated and occupied using random values. Tensor tree dot products (Eq. 12) and tensor tree hole products (Eq. 16) are calculated.

If orthogonal tensor trees are used, calculating the corresponding density matrix tree and matrix tree representations of operators in SPF & SHF basis can be simplified. If a product operator acts on a subset of all leaves, the matrix representation is only required at nodes that connect the active leaves. On all other nodes, the matrix representation is the identity matrix and can be omitted. The active nodes are managed internally by the class *TreeMarker*. Fig. 7b shows exemplary code to generate a *MultiLeafOperator* and build its sparse matrix tree representation in SPF & SHF basis for an auxiliary tensor tree. QuTree enables calculating the matrix representation

at other subsets of nodes as well, including the inverse case where active and inactive nodes are swapped. This enables calculating the grid representation used in the context of nonhierarchical correlation discrete variable representation,[59] for example.

a)

```
// define dimensions of a 4th order tensor
Tensorshape shape({2, 3, 4, 5)};
Tensorcd A(shape); // allocate memory and set to zero
Tensorcd B(shape, false); // allocate memory without setting to zero

// loop over all tensor elements with compound index I
for (int I = 0; I < shape.totalDimension(); ++I) {
     // access tensor elements as array A_I; set value to 1
     A[I] = 1.;
     // map compound index to vector of indices I → (i_0, i_1, …, i_d)
     vector<size_t> idxs = indexMapping(I, shape);
     // access tensor via small indices B_{i_0…i_d} = 2 · i_0
     B(idxs) = 2. * idxs[0];
}

// performs Eq. (1), h_{i_0 j_0} = Σ_{I(0)} A*_{i_0 I(0)} · B_{j_0 I(0)} resulting in a 2x2 matrix
Matrixcd h = contraction(A, B, 0);
// performs Eq. (2), C_{i_0…i_d} = h_{i_0 j_0} · A_{j_0 I(0)}
Tensorcd C = matrixTensor(h, A, 0);

// Create tensor with eternal memory ("false" here declares that
// tensor will not take ownership of memory
complex<double>* mem = new complex<double>[120];
Tensorcd D(shape, mem, false);
// A tensor without ownership of memory will not free the memory upon
// destruction, therefore delete memory manually
delete mem;
```

b)

```
// Generate Tree and TensorTree
Tree tree = TreeFactory::balancedTree(12, 5, 2);
TensorTreecd Psi(tree);

// Operations on Tensor Trees
using namespace TreeFunctions;
MatrixTreecd W = dotProduct(Psi, Psi, tree);      // <Psi|Psi>_p
MatrixTreecd C = contraction(Psi, Psi, W, tree); // <Psi|Psi>_(p)
MatrixTreecd Rho = contraction(Psi, tree, true); // Assume orthogonal basis

// Represent Operators
LeafFuncd x = &LeafInterface::applyX;
MultiLeafOperator<complex<double>> M(x, 0);
auto Mrep = represent(M, Psi, tree);       // <Psi|M|Psi>_p
auto Mcon = contraction(Psi, Mrep, tree); // <Psi|M|Psi>_(p)
```

Fig. 7: a) Exemplary QuTree code for generating tensors, accessing tensor elements and performing matrix tensor operations. b) Exemplary QuTree code for generating a Tree and TensorTree and performing fundamental operations including dot-products, contractions and representation of operators.

We now demonstrate how QuTree facilitates writing complex code by calculating all matrix elements required to implement the equations of motion of the multilayer MCTDH approach. This is often considered to be the most complex part of a multilayer MCTDH implementation, as well as the most numerically demanding. Fig. 8 shows the relevant code for the implementation of the class *HamiltonianRepresentation*, which calculates and stores the matrix elements. The multilayer MCTDH wavefunction is represented as a *TensorTree*, the Hamiltonian is represented as a *SumOfProductOperator*, and the tree structure is managed by *Tree*. All required matrix elements are calculated by five QuTree functions. The code is prototypical for tree tensor network simulations; similar code is required to build required matrices for quantum computing applications and machine learning tasks.

The QuTree package comes with a more comprehensive set of usage examples, documentation and tutorials which demonstrate functionality not covered in this paper.

```
class HamiltonianRepresentation {
	// …
	void build(const SumOfProductOperatorcd& H, const TensorTreecd& Psi,
		const Tree& tree) {

		// Calculate density matrices
		TreeFunctions::contraction(rho_, Psi, tree, true);

		// Calculate spectral decompositions of density matrices
		rho_decomposition_.calculate(rho_, tree);

		// Calculate inverse density matrices
		rho_inverse_ = inverse(rho_, tree);

		// Calculate representations of summands in H
		TreeFunctions::represent(hMats_, H, Psi, Psi, tree);

		// Calculate corresponding contractions
		TreeFunctions::contraction(hContractions_, hMats_,
			Psi, Psi, tree);
	}
	// …

	MatrixTreecd rho_;
	SpectralDecompositionTreecd rho_decomposition_;
	MatrixTreecd rho_inverse_;
	SparseMatrixTreescd hMats_;
	SparseMatrixTreescd hContractions_;

};
```

Fig. 8: Relevant code for a class that calculates all objects required to evaluate the equations of motion of the multilayer MCTDH approach. All objects are represented using QuTree classes and calculated from functions provided by the present work.

IV. Performance

We study the performance of central operations in QuTree that were presented in Sec. II. In the following, we will consider complex double instantiations of class templates, the common datatype for time-dependent physical and chemical applications as well as quantum computing applications. First, we study the performance of the central tensor operations: the tensor hole contraction (Eq. 1, performed by *contraction*) and matrix tensor products (Eq. 2, performed by *matrixTensor*). The functions *contraction* and *matrixTensor* are implemented using the general matrix-matrix product function *Xgemm* (zgemm in the case of complex double) from Basic Linear Algebra Subprograms (BLAS). In the supporting information we show how these operations are mapped to *Xgemm* calls and discuss their arithmetic intensity.

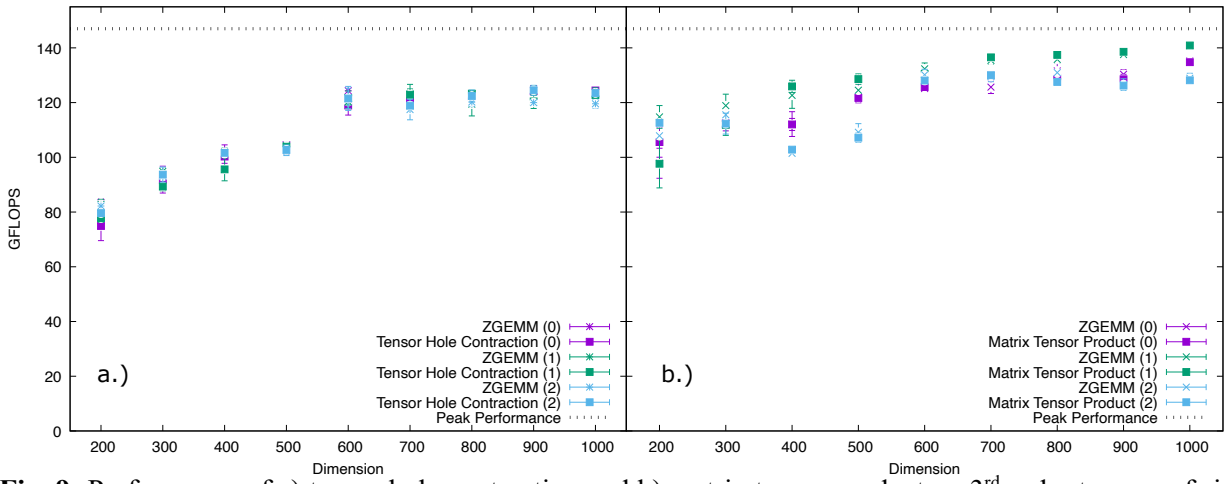

Fig. 9: Performance of a) tensor hole contraction and b) matrix tensor product on 3rd order tensors of size (m, m, m) in giga floating point operations per second (GFLOPS) as a function of the dimension m. The tensors are of type complex double. Squares show the performance of the tensor operation whereas crosses indicate the performance of the corresponding ZGEMMs. Peak performance of the machine used (145 GFLOPS) is indicated with a dotted line.

To benchmark the fundamental tensor operations, we create 3rd order tensors with dimensions $\boldsymbol{n} = (m, m, m)$ and perform the tensor hole products and matrix tensor products. Large 3rd order tensors are used in MPS (tensor train), and other binary tree calculations. The tensor hole contraction and matrix tensor product are arithmetically bound for the machine used in the present work for dimensions larger or equal to 18 or 27, respectively (see supporting material for more details). We therefore investigate the performance in floating point operations per second. Fig. 9 shows the performance of tensor hole contraction (Fig. 9a) and matrix tensor product (Fig. 9b) in giga floating point operations per second (GFLOPS) as a function of the dimension m. Both panels show the performance for operations targeting the first, middle, and last index of the tensor, labeled (0), (1), and (2), respectively. The peak performance of the present machine for a default ZGEMM on square matrices is 145 GFLOPS (see supporting material for more detailed information). The performance of the tensor hole contraction saturates at around 120-125 GFLOPS for dimensions of 600 or higher which corresponds to 83-86% of the peak square ZGEMM performance. For the matrix tensor product, the highest performance is 125-135 GFLOPS (again depending on which index is targeted) which corresponds to 86-93% of the peak performance. Since the arithmetic intensity of both operations is lower than that of a square ZGEMM, a slightly lower performance is expected. To provide a better estimate for the efficiency of the routines (i.e. the overhead beyond calling ZGEMM), we show the performance of the ZGEMMs called within the tensor hole

contraction and matrix tensor product in the same diagram (crosses in Fig. 9). The performance of the underlying ZGEMM matches the performance of the tensor operations with the exception of a few outliers which are likely due to inconsistencies of background tasks.

Next, we perform benchmarks of tree tensor network operations and show that the overhead of handling the tree topology is negligible. We perform tree tensor network operations using balanced, binary trees with 2^l leaves and local tensors with dimensions $(2,2,2)$. This is the smallest tensor dimension that results in non-trivial tree tensor network operations and allows us to investigate any potential overhead. Fig. 10 shows the time required to compute the tree dot product and tree hole contraction between two TTNS (Eqs. 13 and 14) as a function of the number of leaves. Furthermore, we show the time required to compute the sparse tree dot product and hole product corresponding to a product operator acting on the first and last leaves in the tree (i.e. the most distant leaves). In the largest benchmark, dense operations are performed on a tree with one million leaves (two million nodes) in 1365 ms (tree dot product) and 875 ms (tree hole contraction). This correponds to an average time of 0.65 µs per node (tree dot product), and 0.4 µs per node (tree hole contraction). The corresponding sparse operations only take between 30-40 µs since only $2 \cdot \log N_{leaves} - 1$ nodes are accessed. This results in an average operation time of 0.8-1.0 µs per node. Access times of a couple microseconds per node are much smaller than tensor operation times of practically relevant tensors and are therefore negligible. We note that for systems with more than 250 000 leaves, iterating over all nodes and only checking whether nodes are marked active in a calculation cycle will result in $O(N_{leaves})$ scaling even if only $O(\log(N_{leaves}))$ nodes are active. The *TreeMarker* class provides a sparse iterator that accesses only active nodes and achieves logarithmic scaling for arbitrary large trees. The trend lines show that the operations result in the expected scaling.

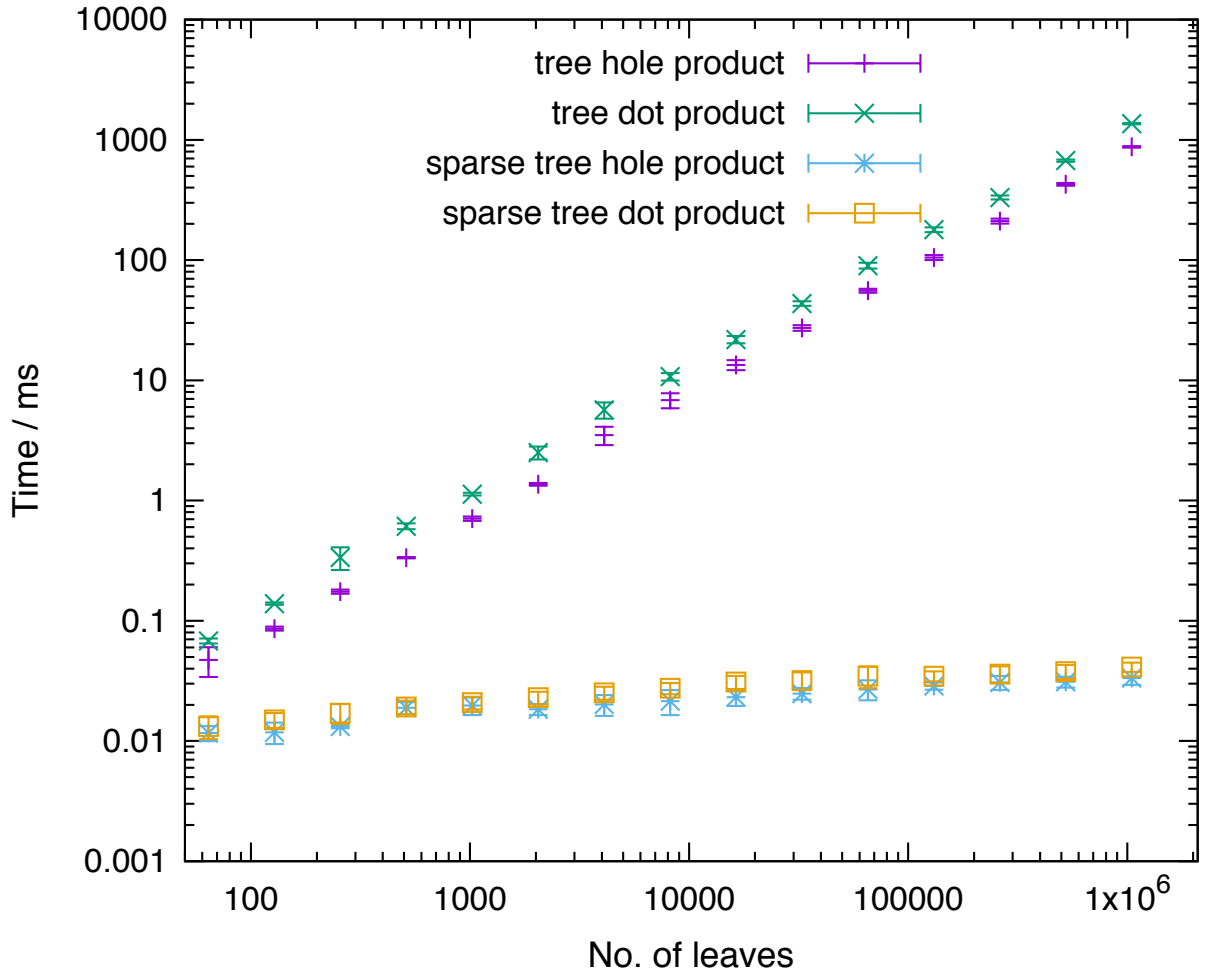

Fig. 10: Time required to perform tensor tree operations for binary, balanced trees as a function of the number of leaves. Tensors are of type complex double and have a dimension of two for each index. Sparse tree operations are performed for a path connecting two most-distant leaves.

V. A Quantum Computing Application: 2D Random Circuit Simulation

We demonstrate the capabilities of QuTree in a practical application by performing 2D random circuit simulations on a virtual quantum computer built using QuTree called QuTreeVM. Random circuits have become prominent benchmarks to compare quantum devices to classical computers and were used to demonstrate quantum supremacy on the 53 qubit Sycamore machine.[86-88] They mimic the discretized real-time dynamics of chaotic quantum systems and result in strong entanglement at low circuit depth, making them hard to simulate accurately. An introduction to quantum circuits can be found in recent textbooks.[83, 89]

Recently, the limits of classical computers in simulating random circuits below full accuracy have been explored using matrix product states simulations.[88] We performed simulations of 2D random circuits following the scheme described in Ref. 88. Random circuits consist of alternating layers of random single qubit and two qubit gate operations. In our simulations, we used the simplified random circuits from Ref. 88 in which C_Z operations were used to entangle neighboring qubits (instead of the iSWAP gates used in the original Google experiment[86-87]). We simulate the system by representing the state vector as a tensor network and applying single- or two-qubit operators using the scheme described in Ref. 38. Fig. 11 shows diagrams of the different tree tensor networks we tested.

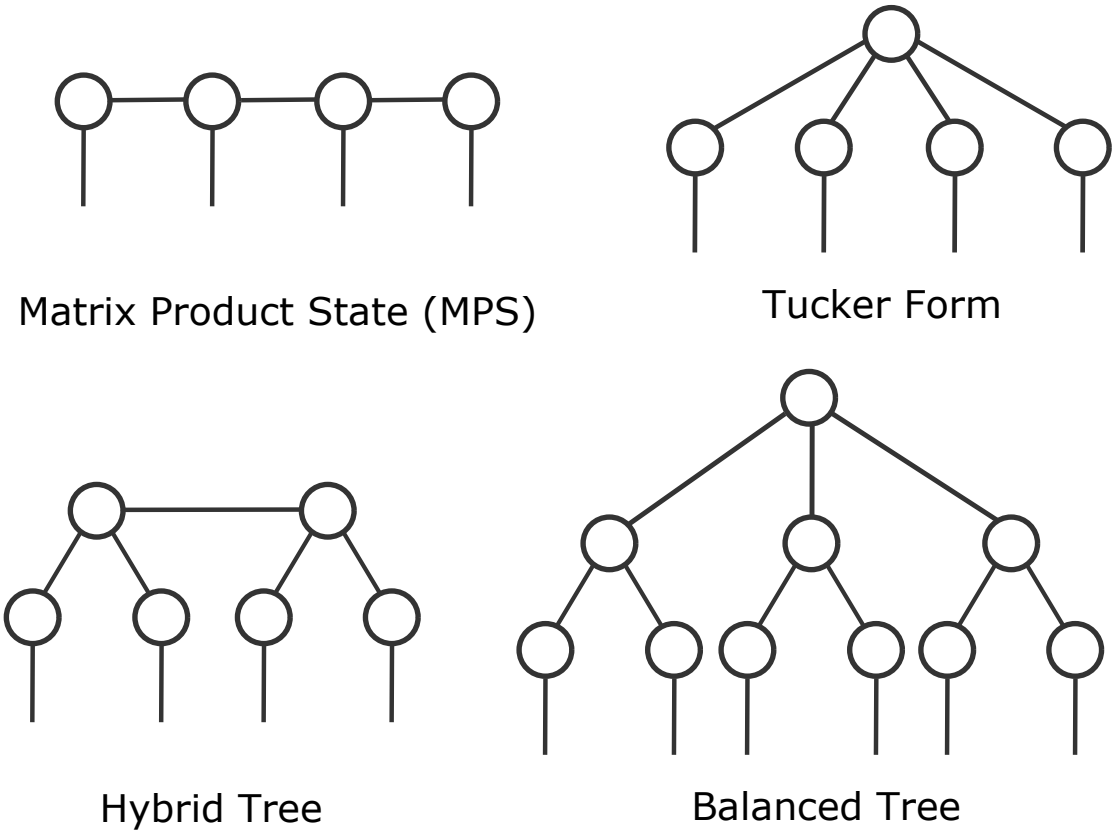

Fig. 11: Tree tensor network structures used in 2D random circuit simulations: a) a matrix product state, b) a Tucker form, c) a hybrid tree, and d) a three-layer balanced tree.

Different MPS structures were extensively studied in Ref. 88. Here, we compare different tree topologies: a matrix-product state form (MPS), a Tucker form, a hybrid of a Tucker and a matrix product (hybrid) form, and a balanced tree tensor network (see Fig. 11). Note that the hybrid model is equivalent to a balanced tree that connects the left and right pair blocks by an additional node – the additional node is redundant and can be dropped. Each model has its own strengths and flaws in describing entanglement. MPSs capture one-dimensional nearest neighbor entanglement most effectively, balanced trees capture recursively partitioned entanglement, and Tucker forms are preferable if multiple subsystems are strongly entangled inside themselves and strongly between each other. The entangling operations of 2D random circuits target neighboring qubits; thus, entanglement is created locally at low circuit depth and is then quickly distributed among the whole system as depth increases. It is therefore not obvious whether it is more important to describe short-range or long-range entanglement to obtain high fidelities.

Since different tree structures scale differently with the bond dimension, we investigate both resulting accuracy and computational time/memory requirements. Fig. 12 shows the computation time as a function of the (geometric) average error per gate operator, ϵ_{avg} for trees (a)-(d) as well as for Qiskit's MPS simulator, which we compare to for reference. As a quantum signature of chaos, the total fidelity $\mathcal{F}$ of a simulation to depth D can be assessed via[88]

$$\mathcal{F} \approx \left(1 - \epsilon_{avg}\right)^{D}. \tag{18}$$

In terms of the bond dimension χ, the most accurate results are obtained by MPS (a), followed by the Tucker form (b), the hybrid model (c) and, far off, by the binary tree (d). This shows that it is more important to describe local entanglement than global entanglement. Even though strong entanglement is created between all qubits, it does not pay off to sacrifice accuracy in describing short range entanglement for long range entanglement. The balanced tree (d) and hybrid models (c) incorporate further contractions and are therefore less accurate compared to (a) and (b) for the same computation time. The Tucker form outperforms MPS for computations taking 750s or less and is the most accurate representation in the limit of low bond dimension. The Qiskit MPS simulator provides a significantly less accurate description. The largest Qiskit MPS simulation provides an average error of 5.2% in 19 h. For comparison, QuTreeVM achieves an accuracy of 4.8% in 29s using the Tucker form. Qiskit's simulator does not allow grouping qubits into qudits and requires shifting qubits together before applying an operator, which typically imposes additional, approximate SVDs. This results in a much lower fidelity at the same computation time compared to the QuTree simulations.

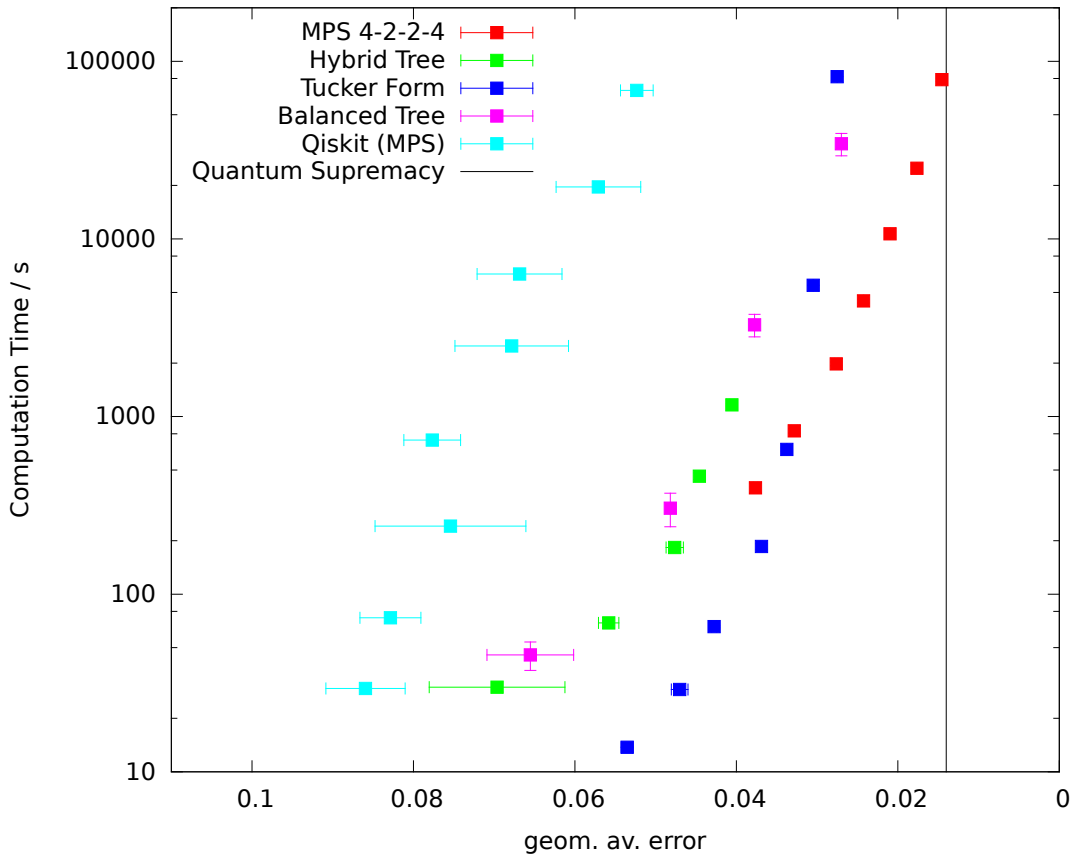

Fig. 12: Geometric average error per gate operator in a 2D random circuit simulation as a function of the computation time for different tree topologies using QuTree and Qiskit's MPS simulator. The tree diagrams are shown in Fig. 11. The "quantum supremacy" line corresponds to the geometric average error needed to match the fidelity observed in a recent Google experiment.[86-87]

Fig. 13 shows the memory required to store the tensor networks as a function of the average fidelity per gate operator. We find that the memory scales similarly to the computation time for the different TTNSs we tested. While deep trees can be highly efficient in describing weak entanglement, the more elaborate topology does not provide a more efficient representation in extremely strongly entangled systems like those resulting from random circuit simulations.

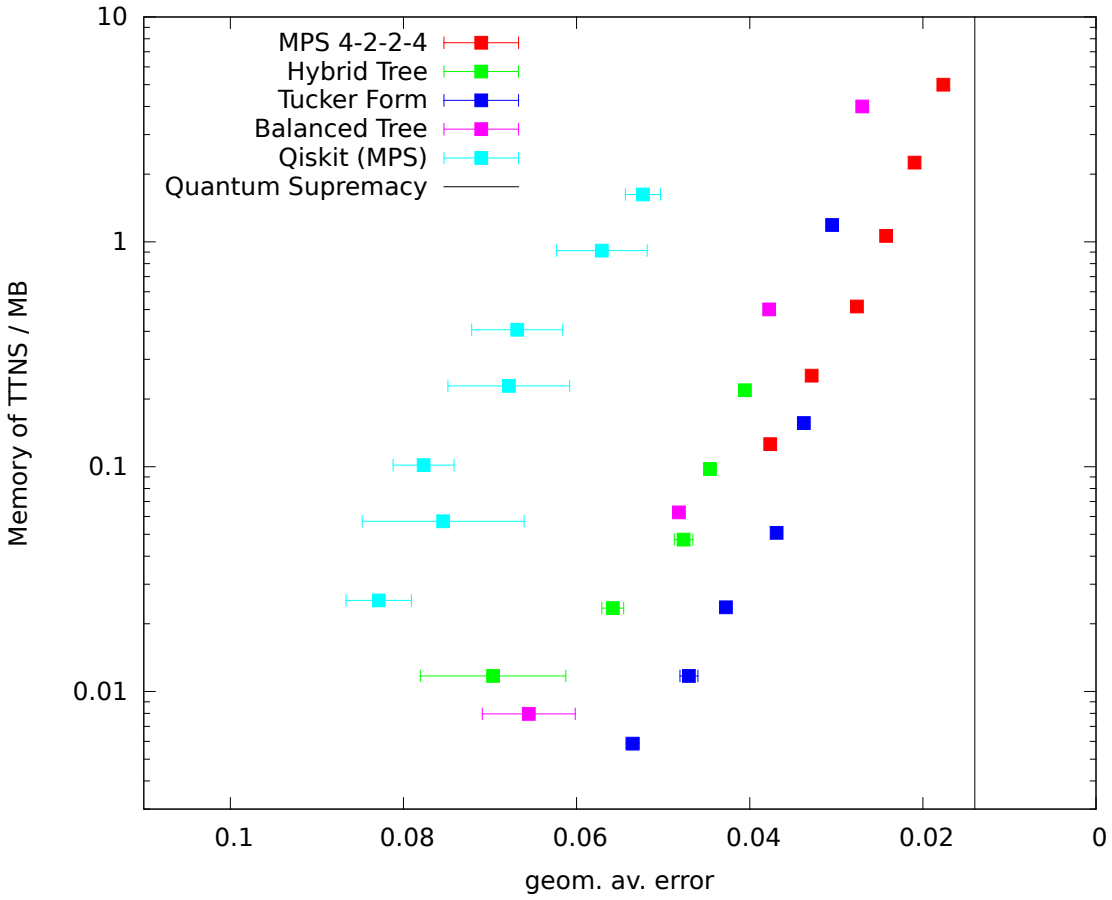

Fig. 13: Geometric average error per gate operator in a 2D random circuit simulation as a function of the number of the size of the tensor networks for different tree topologies. The corresponding tree diagrams are shown in Fig. 11.

VI. Calculating Eigenstates of Hamiltonians: Coupled Harmonic Oscillators

To demonstrate the eigen-solver provided by QuTree, we computed the eigenstates of coupled harmonic oscillators, varying the number of oscillators and basis. We chose a model Hamiltonian used in the context of multilayer MCTDH.[90] The model consists of n harmonic oscillators with frequency $\omega = 4000cm^{-1}$ and strong, linear, nearest neighbor couplings in 1D with coupling strengths of $\lambda = 2000cm^{-1}$. The Hamiltonian reads $\hat{H} = \sum_i \frac{d}{dx_i^2} + \frac{1}{2}\omega^2 \hat{x}_i^2 + \lambda \hat{x}_i \hat{x}_{i+1}$ where the last summand is coupled to the first oscillator (i.e. with periodic boundaries). The system is scalable, can be solved analytically, and shows nontrivial entanglement. We solved for the ground state of the system with 4, 8, 16, 32, 64, 128, 256, 512, and 1024 oscillators. Excited states can be obtained in QuTree via the state-averaged approach described by Manthe.[91] We used a binary tree topology which is often the default topology since every mode or particle has only a logarithmic

distance to every other mode or particle. For the leaves, we used 10 harmonic oscillator eigenfunctions with the natural frequency. Doubling the number of eigenfunctions changes the resulting energies by less than $10^{-14} cm^{-1}$. To converge the calculations, we used a virtual bond dimension of x at the bottom layer and increased it by y when ascending a layer towards the root, which we denote by the basis label $(x + y)$. We used the basis sets $(4 + 4)$, $(8 + 6)$, and $(10 + 8)$ to solve for the ground states using a self-consistent field approach for tree tensor network states[84] with 10 iterations per system size and basis set. The ground state energies and computation times per iteration are shown in Table 1. We find that even the (4+4) basis provides a good estimate of the ground state energy of a 1024 harmonic oscillator system with an error of 1.29 cm^{-1} or $6.4 \cdot 10^{-7}$ relative error in $67.6s$ per iteration. For the largest (10+8) basis, which requires $663.9s$ per iteration, the error is reduced to $0.000572\mathrm{cm}^{-1}$ which is $2.8 \cdot 10^{-10}$ relative to the exact ground state energy.

Table 1: Ground state energies and computation times per iteration for a system of coupled harmonic oscillators as a function of dimensionality and basis sets. The calculations converged in 5-10 iterations. See main text for detailed information.

		Basis Set					
		(4+4)		(8+6)		(10+8)	
System Size	Exact Energy	Error [cm^{-1}]	Time [s]	Error [cm^{-1}]	Time [s]	Error [cm^{-1}]	Time [s]
4	7863.703305	0.002611	0.004	0.000003	0.011	0.000000	0.0187
8	15733.47199	0.009887	0.0132	0.000025	0.0838	0.000001	0.149
16	31466.96482	0.019973	0.0658	0.000051	0.3329	0.000002	0.62
32	62933.92965	0.039993	0.2155	0.000112	0.9624	0.000010	2.60
64	125867.8593	0.080339	0.6436	0.000206	2.92	0.000007	10.02
128	251735.7186	0.160530	1.8874	0.000414	123.93	0.000063	25.09
256	503471.4372	0.321333	5.6871	0.000829	29.498	0.000134	73.69
512	1006942.874	0.642796	18.5994	0.001676	91.3516	0.000236	228.55
1024	2013885.749	1.287242	67.6008	0.003322	308.2981	0.000572	701.37

Vibrational states on general potential energy surfaces can be evaluated using the nonhierarchical correlation discrete variable representation.[59] Indeed, the exemplary numerical simulations presented in Ref. 59 were performed using the QuTree package.

VII. Conclusion

We presented QuTree, a novel Tree Tensor Network library written in C++, available for public use in a GitHub repository.[92] The library is designed to facilitate development of new methods based on tensor networks. We provided an overview of the central functionality of the library and its corresponding theoretical background. We discussed how equations are mapped to functions in C++ and how theoretical concepts like tensor networks and trees are represented by corresponding class structures. We demonstrated the efficiency of the library by benchmarking important backend routines for various cases that are relevant in typical applications. We showed that the library is efficient when either large tensors with various shapes are used or when many small tensors are used in trees with hundreds of thousands of leaves and nodes. We performed random circuit simulations on a virtual quantum computer and computed eigenstates of coupled harmonic oscillators to demonstrate that the library can be used to study state-of-the-art problems with little effort. In its current form, QuTree is limited to computations on a single CPU. Future work will extend QuTree to multiple nodes with both GPUs and CPUs by changing the backend to existing abstract linear algebra packages such as BLAS++/LAPACK++,[74-75] Magma,[73] or the TiledArray framework.[76] This will enable large-scale simulations and thereby facilitate applications to quantum circuits with much stronger entanglement.

VIII. Supplementary Material

Details of arithmetic complexity of tensor operations and memory aspects of performance, along with technical details for benchmarks.

VIV. Acknowledgments

This material is based upon work supported by the AMOS program of the U. S. Department of Energy, Office of Science, Basic Energy Sciences, Chemical Sciences, Geosciences and Biosciences Division and the DOE Q-Next project. Support from IBM through the Stanford

SystemX Alliance is also gratefully acknowledged. R.E. thanks the Deutsche Forschungsgemeinschaft for financial support.

Data Availability

The data that supports the findings of this study are available within the article and its supplementary material.

References

1. Meyer, H. D.; Manthe, U.; Cederbaum, L. S., "The multi-configurational time-dependent Hartree approach," *Chem. Phys. Lett.* **165**, 73-78 (1990).
2. Meyer, H. D.; Manthe, U.; Cederbaum, L. S.; Manthe, U.; Köppel, H., "The multi-configurational time-dependent Hartree approach New method for calculating wave packet dynamics: strongly coupled surfaces and the adiabatic basis," *Chem. Phys. Lett.* **165**, 73-78 (1990).
3. Wang, H.; Thoss, M., "Multilayer formulation of the multiconfiguration time-dependent Hartree theory," *J. Chem. Phys.* **119**, 1289-1299 (2003).
4. Manthe, U., "A multilayer multiconfigurational time-dependent Hartree approach for quantum dynamics on general potential energy surfaces," *J. Chem. Phys.* **128**, 164116 (2008).
5. Huarte-Larrañaga, F.; Manthe, U., "Full dimensional quantum calculations of the CH4+H → CH3+H2 reaction rate," *J. Chem. Phys.* **113**, 5115-5118 (2000).
6. Huarte-Larrañaga, F.; Manthe, U., "Vibrational excitation in the transition state: The CH4+H → CH3+H2 reaction rate constant in an extended temperature interval," *J. Chem. Phys.* **116**, 2863-2869 (2002).
7. Huarte-Larrañaga, F.; Manthe, U., "Quantum mechanical calculation of the OH+HCl → H2O+Cl reaction rate: Full-dimensional accurate, centrifugal sudden, and J-shifting results," *J. Chem. Phys.* **118**, 8261-8267 (2003).
8. Manthe, U.; Huarte-Larrañaga, F., "Partition functions for reaction rate calculations: statistical sampling and MCTDH propagation," *Chem. Phys. Lett.* **349**, 321-328 (2001).
9. Wu, T.; Werner, H.-J.; Manthe, U., "First-principles theory for the H+CH4 → H2+CH3 reaction," *Science* **306**, 2227-2229 (2004).
10. Schiffel, G.; Manthe, U.; Nyman, G., "Full-Dimensional Quantum Reaction Rate Calculations for H + CH4 → H2 + CH3 on a Recent Potential Energy Surface," *J. Phys. Chem. A* **114**, 9617-9622 (2010).
11. Welsch, R.; Manthe, U., "Reaction dynamics with the multi-layer multi-configurational time-dependent Hartree approach: H + CH4 → H2 + CH3 rate constants for different potentials," *J. Chem. Phys.* **137**, 244106 (2012).
12. Schiffel, G.; Manthe, U., "A transition state view on reactive scattering: Initial state-selected reaction probabilities for the H+CH4 → H2+CH3 reaction studied in full dimensionality," *J. Chem. Phys.* **133**, 174124 (2010).
13. Schiffel, G.; Manthe, U., "Quantum dynamics of the H+CH4 → H2+CH3 reaction in curvilinear coordinates: Full-dimensional and reduced dimensional calculations of reaction rates," *J. Chem. Phys.* **132**, 84103 (2010).
14. Schiffel, G.; Manthe, U., "Communication: A rigorous transition state based approach to state-specific reaction dynamics: Full-dimensional calculations for H+CH4 → H2+CH3," *J. Chem. Phys.* **132**, 191101 (2010).
15. Welsch, R.; Manthe, U., "Communication: Ro-vibrational control of chemical reactivity in H+CH4 → H2+CH3: full-dimensional quantum dynamics calculations and a sudden model," *J. Chem. Phys.* **141**, 51102 (2014).
16. Welsch, R.; Manthe, U., "The role of the transition state in polyatomic reactions: Initial state-selected reaction probabilities of the H+CH4 → H2+CH3 reaction," *J. Chem. Phys.* **141**, 174313 (2014).

17. Welsch, R.; Manthe, U., "Loss of Memory in H+CH4 → H2 + CH3 State-to-State Reactive Scattering," *J. Phys. Chem. Lett.* **6**, 338-342 (2015).
18. Ellerbrock, R.; Manthe, U., "H+CH4 → H2 + CH3 initial state-selected reaction probabilities on different potential energy surfaces," *Chem. Phys.* **482**, 106-112 (2017).
19. Ellerbrock, R.; Manthe, U., "Full-dimensional quantum dynamics calculations for H+CHD3 → H2 + CD3: The effect of multiple vibrational excitations," *J. Chem. Phys.* **148**, 224303 (2018).
20. Ellerbrock, R.; Manthe, U., "Natural reaction channels in H+CHD3 → H2 + CD3," *Faraday Disc.* **212**, 217-235 (2018).
21. Westermann, T.; Eisfeld, W.; Manthe, U., "Coupled potential energy surface for the F(^{2}P)+CH4 → HF+CH3 entrance channel and quantum dynamics of CH4 · F− photodetachment," *J. Chem. Phys.* **139**, 14309 (2013).
22. Vendrell, O.; Brill, M.; Gatti, F.; Meyer, H.-D., "Full dimensional (15-dimensional) quantum-dynamical simulation of the protonated water-dimer III: Mixed Jacobi-valence parametrization and benchmark results for the zero point energy, vibrationally excited states, and infrared spectrum," *J. Chem. Phys.* **130**, 234305 (2009).
23. Vendrell, O.; Gatti, F.; Lauvergnat, D.; Meyer, H.-D., "Full-dimensional (15-dimensional) quantum-dynamical simulation of the protonated water dimer. I. Hamiltonian setup and analysis of the ground vibrational state," *J. Chem. Phys.* **127**, 184302 (2007).
24. Vendrell, O.; Gatti, F.; Lauvergnat, D.; Meyer, H.-D., "Dynamics and infrared spectroscopy of the protonated water dimer," *Angew. Chemie Int. Ed.* **46**, 6918-6921 (2007).
25. Vendrell, O.; Gatti, F.; Meyer, H.-D., "Full dimensional (15-dimensional) quantum-dynamical simulation of the protonated water dimer. II. Infrared spectrum and vibrational dynamics," *J. Chem. Phys.* **127**, 184303 (2007).
26. Coutinho-Neto, M. D.; Viel, A.; Manthe, U., "The ground state tunneling splitting of malonaldehyde: Accurate full dimensional quantum dynamics calculations," *J. Chem. Phys.* **121**, 9207-9210 (2004).
27. Hammer, T.; Coutinho-Neto, M. D.; Viel, A.; Manthe, U., "Multiconfigurational time-dependent Hartree calculations for tunneling splittings of vibrational states: Theoretical considerations and application to malonaldehyde," *J. Chem. Phys.* **131**, 224109 (2009).
28. Hammer, T.; Manthe, U., "Intramolecular proton transfer in malonaldehyde: Accurate multilayer multi-configurational time-dependent Hartree calculations," *J. Chem. Phys.* **134**, 224305 (2011).
29. Schroeder, M.; Gatti, F.; Meyer, H.-D., "Theoretical studies of the tunneling splitting of malonaldehyde using the multiconfiguration time-dependent Hartree approach," *J. Chem. Phys.* **134**, 234307 (2011).
30. Hammer, T.; Manthe, U., "Iterative diagonalization in the state-averaged multi-configurational time-dependent Hartree approach: Excited state tunneling splittings in malonaldehyde," *J. Chem. Phys.* **136**, 54105 (2012).
31. Schroeder, M.; Meyer, H.-D., "Calculation of the vibrational excited states of malonaldehyde and their tunneling splittings with the multi-configuration time-dependent Hartree method," *J. Chem. Phys.* **141**, 34116 (2014).
32. Raab, A.; Worth, G. A.; Meyer, H. D.; Cederbaum, L. S., "Molecular dynamics of pyrazine after excitation to the S(2) electronic state using a realistic 24-mode model Hamiltonian," *J. Chem. Phys.* **110**, 936-946 (1999).

33. Cattarius, C.; Worth, G. A.; Meyer, H. D.; Cederbaum, L. S., "All mode dynamics at the conical intersection of an octa-atomic molecule: Multi-configuration time-dependent Hartree (MCTDH) investigation on the butatriene cation," *J. Chem. Phys.* **115**, 2088-2100 (2001).
34. Worth, G. A.; Meyer, H. D.; Koeppel, H.; Cederbaum, L. S.; Burghardt, I., "Using the MCTDH wavepacket propagation method to describe multimode non-adiabatic dynamics," *Int. Rev. Phys. Chem.* **27**, 569-606 (2008).
35. Meng, Q.; Meyer, H.-D., "A multilayer MCTDH study on the full dimensional vibronic dynamics of naphthalene and anthracene cations," *J. Chem. Phys.* **138**, 14313 (2013).
36. Manthe, U.; Weike, T., "On the multi-layer multi-configurational time-dependent Hartree approach for bosons and fermions," *J. Chem. Phys.* **146**, 64117 (2017).
37. Wang, H.; Thoss, M., "Numerically exact quantum dynamics for indistinguishable particles: The multilayer multiconfiguration time-dependent Hartree theory in second quantization representation," *J. Chem. Phys.* **131**, 24114 (2009).
38. Ellerbrock, R.; Martinez, T. J., "A multilayer multi-configurational approach to efficiently simulate large-scale circuit-based quantum computers on classical machines," *J. Chem. Phys.* **153**, 51101 (2020).
39. Manthe, U., "Wavepacket dynamics and the multi-configurational time-dependent Hartree approach," *J. Phys. Cond. Matt.* **29**, 253001 (2017).
40. Meyer, H. D., "Studying molecular quantum dynamics with the multiconfiguration time-dependent Hartree method," *WIRES: Comp. Mol. Sci.* **2**, 351 (2012).
41. White, S. R., "Density matrix formulation for quantum renormalization groups," *Phys. Rev. Lett.* **69**, 2863 (1992).
42. White, S. R., "Density-matrix algorithms for quantum renormalization groups," *Phys. Rev. B* **48**, 10345-10356 (1993).
43. Daley, A. J.; Kollath, C.; Schollwöck, U.; Vidal, G., "Time-dependent density-matrix renormalization-group using adaptive effective Hilbert spaces," *J. Stat. Mech.* **2004**, P04005 (2004).
44. Kühner, T. D.; White, S. R., "Dynamical correlation functions using the density matrix renormalization group," *Phys. Rev. B* **60**, 335 (1999).
45. Jeckelmann, E., "Dynamical density-matrix renormalization-group method," *Phys. Rev. B* **66**, 45114 (2002).
46. Ramasesha, S.; Pati, S. K.; Krishnamurthy, H. R.; Shuai, Z.; Brédas, J. L., "Low-lying electronic excitations and nonlinear optic properties of polymers via symmetrized density matrix renormalization group method," *Synth. Met.* **85**, 1019-1022 (1997).
47. Hallberg, K. A., "Density-matrix algorithm for the calculation of dynamical properties of low-dimensional systems," *Phys. Rev. B* **52**, R9827 (1995).
48. Verstraete, F.; Cirac, J. I., "Renormalization algorithms for quantum-many body systems in two and higher dimensions," *arXiv*, cond-mat/0407066 (2004).
49. Shi, Y. Y.; Duan, L. M.; Vidal, G., "Classical simulation of quantum many-body systems with a tree tensor network," *Phys. Rev. A* **74**, 22320 (2006).
50. Vidal, G., "Entanglement renormalization," *Phys. Rev. Lett.* **99**, 220405 (2007).
51. Orús, R., "Tensor networks for complex quantum systems," *Nature Rev. Phys.* **1**, 538-550 (2019).
52. Evenhuis, C.; Nyman, G.; Manthe, U., "Quantum dynamics of the CH$_3$ fragment: A curvilinear coordinate system and kinetic energy operators," *J. Chem. Phys.* **127**, 144302 (2007).

53. Braams, B. J.; Bowman, J. M., "Permutationally Invariant Potential Energy Surfaces in High Dimensionality," *Int. Rev. Phys. Chem.* **28**, 577 (2009).
54. Ischtwan, J.; Collins, M. A., "Molecular potential energy surfaces by interpolation," *J. Chem. Phys.* **100**, 8080 (1994).
55. Chen, J.; Xu, X.; Xu, X.; Zhang, D. H., "Communication: An accurate global potential energy surface for the OH + CO → H + CO2 reaction using neural networks," *J. Chem. Phys.* **138**, 154301 (2013).
56. Manthe, U., "A time-dependent discrete variable representation for (multiconfiguration) Hartree methods," *J. Chem. Phys.* **105**, 6989-6994 (1996).
57. van Harrevelt, R.; Manthe, U., "Multidimensional time-dependent discrete variable representations in multiconfiguration Hartree calculations," *J. Chem. Phys.* **123**, 64106 (2005).
58. Smolyak, S. A., "Quadrature and interpolation formulas for tensor products of certain classes of functions," *Soviet Math. Dokl.* **4**, 240-243 (1963).
59. Ellerbrock, R.; Manthe, U., "A non-hierarchical correlation discrete variable representation," *J. Chem. Phys.* **156**, 134107 (2022).
60. Otto, F., "Multi-Layer Potfit: An accurate potential representation for efficient high-dimensional quantum dynamics," *J. Chem. Phys.* **140**, 14106 (2014).
61. Pelaez, D.; Meyer, H. D., "The multigrid POTFIT (MGPF) method: Grid representations of potentials for quantum dynamics of large systems," *J. Chem. Phys.* **138**, 14108 (2013).
62. Wodraszka, R.; Carrington, T., "A new collocation-based multi-configuration time-dependent Hartree (MCTDH) approach for solving the Schrödinger equation with a general potential energy surface," *J. Chem. Phys.* **148**, 044115 (2018).
63. "MCTDH Package, v8.6." https://www.pci.uni-heidelberg.de/cms/mctdh.html#package (accessed November 1, 2023).
64. Lin, R.; Molignini, P.; Papariello, L.; Tsatsos, M. C.; Leveque, C.; Weiner, S. E.; Fasshauer, E.; Chitra, R.; Lode, A. U. J., "MCTDH-X: The multiconfigurational time-dependent Hartree method for indistinguishable particles software," *Quantum Sci. Tech.* **5**, 024004 (2020).
65. Jackle, A.; Meyer, H.-D., "Product representation of potential energy surfaces," *J. Chem. Phys.* **104**, 7974-7984 (1996).
66. Fishman, M.; White, S. R.; Stoudenmire, E. M., "The ITensor Software Library for Tensor Network Calculations," *SciPost Phys. Code.*, 4 (2022).
67. Roberts, C.; Milsted, A.; Ganahl, M.; Zalcman, A.; Fonatine, B.; Zou, Y.; Hidary, J.; Vidal, G.; Leichenauer, S., "TensorNetwork: A Library for Physics and Machine Learning," *arxiv*, 1905.01330 (2019).
68. Lyakh, D. I.; Nguyen, T.; Claudino, D.; Dumitrescu, E.; McCaskey, A. J., "ExaTN: Scalable GPU-Accelerated High-Performance Processing of General Tensor Networks at Exascale," *Front. Appl. Math. Stat.* **8**, 838601 (2022).
69. "OpenBLAS." https://github.com/OpenMathLib/OpenBLAS (accessed November 1, 2023).
70. "oneAPI Math Kernel Library." https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html (accessed November 1, 2023).
71. "cuBLAS." https://developer.nvidia.com/cublas (accessed November 1, 2023).
72. "ScaLAPACK." https://www.netlib.org/scalapack (accessed November 1, 2023).
73. Tomov, S.; Dongarra, J.; Baboulin, M., "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Comput.* **36**, 232-240 (2010).
74. "BLAS++." https://github.com/icl-utk-edu/blaspp (accessed November 1, 2023).

75. ”LAPACK++.” https://github.com/icl-utk-edu/lapackpp (accessed November 1, 2023).
76. ”TiledArray: A general-purpose scalable block-sparse tensor framework.” https://github.com/valeevgroup/tiledarray (accessed November 1, 2023).
77. ”PyTorch.” https://pytorch.org (accessed November 1, 2023).
78. ”TensorFlow.” https://www.tensorflow.org (accessed November 1, 2023).
79. ”JAX: Composable transformations of Python+NumPy programs.” https://github.com/google/jax (accessed November 1, 2023).
80. Kossaifi, J.; Panagakis, Y.; Anandkumar, A.; Pantic, M., “TensorLy: Tensor Learning in Python,” *J. Mach. Learn. Res.* **20**, 1-6 (2019).
81. Beck, M. H.; Meyer, H.-D., “An efficient and robust integration scheme for the equations of motion of the multiconfiguration time-dependent Hartree (MCTDH) method,” *Z. Phys. D* **42**, 113-129 (1997).
82. Ellerbrock, R.; Hoppe, H.; Manthe, U., “A numerically exact correlation discrete variable representation for multi-configurationla time-dependent Hartree calculations,” *J. Chem. Phys.* **158**, 244103 (2023).
83. Nielsen, M. A.; Chuang, I. L., Quantum Computation and Quantum Information. Cambridge University Press: Cambridge, 2010.
84. Larsson, H. R., “Computing vibrational eigenstates with tree tensor network states (TTNS),” *J. Chem. Phys.* **151**, 204102 (2019).
85. Weike, T.; Manthe, U., “Symmetries in the multi-configurational time-dependent Hartree wavefunction representation and propagation,” *J. Chem. Phys.* **154**, (2021).
86. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G. S. L.; Buell, D. A., et al., “Quantum supremacy using a programmable superconducting processor,” *Nature* **574**, 505-510 (2019).
87. Boixo, S.; Isakov, V. S.; Smelyanskiy, V. N.; Babbush, R.; Ding, N.; Jiang, Z.; Bremner, M. J.; Martinis, J. M.; Neven, H., “Characterizing quantum supremacy in near-term devices,” *Nature Phys.* **14**, 595-600 (2018).
88. Zhou, Y.; Stoudenmire, E. M.; Waintal, X., “What limits the simulation of quantum computers?,” *Phys. Rev. X* **10**, 41038 (2020).
89. Lipton, R. J.; Regan, K. W., Quantum Algorithms via Linear Algebra. MIT Press: Cambridge, 2014.
90. Manthe, U., “Layered discrete variable representations and their application within the multiconfigurational time-dependent Hartree approach,” *J. Chem. Phys.* **130**, 054109 (2009).
91. Manthe, U., “The state averaged multiconfigurational time-dependent Hartree approach: Vibrational state and reaction rate calculations,” *J. Chem. Phys.* **128**, 064108 (2008).
92. ”QuTree.” https://github.com/roman-ellerbrock/QuTree.