Towards a Unified Architectural Representation in HPCQC: Extending sys-sage for Quantum Technologies

Durganshu Mishra
Technical University of Munich
Garching, Germany
durganshu.mishra@tum.de

Stepan Vanecek
Technical University of Munich
Garching, Germany
stepan.vanecek@tum.de

Jorge Echavarria
Leibniz Supercomputing Centre
Garching, Germany
Jorge.Echavarria@lrz.de

Xiaolong Deng
Leibniz Supercomputing Centre
Garching, Germany
Xiaolong.Deng@lrz.de

Burak Mete
Leibniz Supercomputing Centre
Garching, Germany
Burak.Mete@lrz.de

Laura Schulz
Leibniz Supercomputing Centre
Garching, Germany
laura.schulz@lrz.de

Martin Schulz
Technical University of Munich
Garching, Germany
schulzm@in.tum.de

***Abstract*—Quantum Computing (QC) presents a significant departure from and substantial parallels to classical computing. Some such parallels include the need to understand topology and system characteristics information and make that available to users as well as other system components like compilers, runtimes, and schedulers.**

We analyze the differences and extend sys-sage to capture quantum information. This leverages code reuse and automatically incorporates QC information into the software stack for hybrid HPCQC integration. This paper addresses the extensions to sys-sage, a classical HPC-relevant library, to also represent and store the characteristics of Quantum Processing Units (QPUs).

Our work also presents the role of sys-sage in pre-processing and analyzing the topology and properties of different components of QCs to provide valuable insights that compilers, schedulers or circuit mappers can use. These features aim to facilitate the HPCQC ecosystem by providing a unified interface.

***Index Terms*—Quantum Computing, High Performance Computing, HPCQC Integration.**

I. Motivation

Quantum Computing (QC) presents a significant departure from classical computing paradigms. Utilizing the principles of quantum mechanics to handle information in ways beyond the capabilities of classical computers, quantum computers promise the potential to solve specific types of complex problems that are challenging or inefficient for classical computers to handle. In particular, the distinctive characteristics of quantum bits (qubits), such as superposition and entanglement, are poised to open the door to algorithms with different complexity properties for key computationally intensive problems in fields such as cryptography [1], chemistry [2], and optimization [3].

However, recent developments in the field position QC as an addition to traditional High-Performance Computing (HPC), not its replacement [4]. That way, a Quantum Processing Unit (QPU) can be integrated into a heterogeneous HPC infrastructures as an additional accelerator, in similar manner to a GPU or an FPGA [5]. This paradigm offloads certain computations to QPUs while managing the program workflow using the classical HPC resources. Many promising algorithms in the near term rely on such a hybrid model that combines classical and quantum information processing. This leads to a push for integration of HPC and QC systems, which dramatically impacts the development of quantum computer software and affects the required software infrastructure [6].

A variety of technologies are being developed to realize practical quantum computers, each differing in the implementation and manipulation of qubits. Among the most prominent are superconducting qubits [7], neutral atom systems [8], and trapped-ion quantum platforms [9]. These technologies exhibit quite different and distinct approaches and representations for qubit management. For instance, in superconducting qubit-based systems, one of the state-of-the-art methods for achieving two-qubit gates is the use of tunable couplers [10], which couples two neighboring qubits whose positions are fixed and static. Therefore, the coupling map is a critical property that defines the topology of the qubit network. This map specifies the physical layout of qubits and their interconnections, effectively defining which pairs of qubits a quantum gate acts upon based on the hardware's physical configuration [11].

Conversely, the concept of a coupling map does not directly apply to neutral atom-based quantum computers. In these systems, qubits are represented by atoms fixed at specific sites, which can be dynamically rearranged during computation using optical tweezers, a process known as atom shuttling. Shuttling enables flexible, non-local entanglement by dynamically rearranging neutral atom qubits, often beginning with closely paired qubits and entangling them using a global controlled-Z (CZ) gate [12]. This modality also enables non-local interactions through a process called *Rydberg Blockade* [13], which can also influence the connectivity of the qubits.

As quantum computing technologies continue to evolve, additional approaches are likely to emerge, each with unique physical properties and topologies. As a consequence, compilers that can span these technologies, while exploring hardware-aware techniques [14]–[16] to map to specific technologies, have a critical role in using QC systems. This is particularly true in hybrid HPCQC environments where details of QC systems should be hidden from domain users. Compilers for such systems must be able to extract, explore and apply the capabilities and constraints of the available target architectures, and — as a consequence — tools within the compilation chain and end-users require access to these platform-specific properties, whether for target-specific or target-agnostic compilation processes. This highlights a critical need for universally reusable software frameworks capable of unifying the representation of multiple quantum technologies. Such frameworks must abstract the underlying physical details, providing higher-level tools in the compilation chain with a consistent interface to seamlessly integrate various quantum technologies. In addition to the necessity of having tools that achieve unified quantum compilation across different modalities, another important aspect of the compilation stack in QC is to have a co-design that includes the interconnections between multiple levels of quantum computation, such as the quantum algorithm, control electronics and the quantum hardware [17], [18]. In this framework, there have been use-case-aware [19]–[21] methods that aim at improving the efficacy of quantum computation on certain hardware types, given insights into the algorithmic design and the quantum circuits.

In this paper, we present an extension to the HPC-oriented sys-sage library[1] [22], which is a comprehensive library designed to capture the hardware topology of classical computing systems and their attributes. sys-sage poses as a unifying solution, standing between various APIs providing information about HPC systems and their attributes, thus providing additional context to the partial information from the different APIs. It manages both static and dynamic data contexts, available at different stages of an application's lifecycle.

In this paper, we extend this library and its classic property descriptions to include the static and dynamic characteristics of quantum systems as well as the relevant context. More specifically, we add new types of components specific to the QC world, and also introduce Relations, a new abstraction to sys-sage's Data Paths, which adds the ability to represent all QC concepts, such as quantum gates. Adding the QC context to the already-existing HPC concept will allow storing and accessing the static and dynamic properties of the systems – both on the classical as well as the quantum side – and will help us reach a true hybrid HPCQC environment.

Specifically, this paper makes the following contributions:

- We explore the concept of topology information on classical systems using the sys-sage library and discuss how to integrate quantum topology information into a common hybrid base.
- We describe how the extended sys-sage library can be used to express constraints and system properties of QC systems to the remaining tool and compilation chain.
- We demonstrate the capabilities of the extended sys-sage library on a real-world use case utilizing a superconducting 20 Qubit system.
- We show the low overhead of the newly extended library for importing, querying and using data from sys-sage and the impact of its potential.

The remainder of the paper is structured as follows. In Section II we introduce the core design of sys-sage, along with the extensions made for representing the quantum systems. In Section III we discuss the potential of sys-sage as a FoMaC library and highlight how sys-sage can be used by other tools such as compilers, schedulers, or mappers. In Section IV, we discuss a real-world use case related to identifying and avoiding the hotspots in scheduling using the metrics provided by sys-sage. In Section V, we discuss a further use-case of sys-sage supporting the schedulers. In Section VI, we evaluate the performance of sys-sage and the overhead of using some of the API calls. Finally, in Section VII, we discuss some related works in the domain and in Section VIII we conclude the work.

II. Capturing Systems Properties with sys-sage

HPC environments are equipped with numerous tools, APIs, interfaces, and instruments that offer partial insights into system architectures, components (e.g., nodes or CPUs), and their operational behavior. Tools such, as *hwloc* and *nvidia-smi*, primarily provide static data, describing the structural configuration of the system. In contrast, tools like PAPI [23] and Likwid [24] deliver dynamic metrics that characterize system performance and application behavior, but often require external sources to map these metrics to static context information.

Additionally, dynamic system characteristics, which may significantly influence performance, must also be considered, even for data typically regarded as static. These characteristics are often gathered through targeted benchmarking to evaluate specific hardware components and resources. However, existing tools tend to focus on specialized domains and collect limited subsets of information, lacking the ability to understand the system comprehensively.

To achieve a complete analysis of the system data, it is essential to integrate static information—available both before and during application execution—with dynamic information,

[1] https://github.com/caps-tum/sys-sage

which evolves throughout an application's lifecycle. Such an integrative approach facilitates a deeper understanding of system performance and behavior, bridging the gap between static configurations and dynamic execution metrics.

These observations and requirements also apply to quantum computers, despite their radically different nature. In fact, they become even more critical, as topology data for Qubit operation, fidelity rates and physical characteristics, external environment influences and system properties like gate sets can vary greatly from system to system or even from run to run. As with classical systems, we need to capture all information — static and dynamic — map it to system properties, track their dynamic behavior and make it available for compilation and runtime systems.

This similarity between classical and quantum systems motivates the use of HPC approaches for system topology and property tracking as well as for quantum systems.

A. Building on the sys-sage Library

The sys-sage library, recently introduced as part of a European system software effort, was specifically designed to address the requirements of classical HPC systems. It is a user-level library focused on the collection, storage, and retrieval of arbitrary hardware-relevant information within an HPC environment. The library effectively manages both static and dynamic data contexts, available at various stages of an application's lifecycle. It provides functionality to acquire, store, update, and query a comprehensive range of hardware and software configuration data for individual nodes or entire systems. This includes details on dynamic hardware topology, current states, system capabilities, and other relevant information essential for understanding and optimizing HPC performance.

Quantum systems have the same requirements for handling their information and, hence, can benefit from sys-sage and its extensions beyond other solutions, like hwloc. We, therefore, build on this work and aim to encapsulate and elucidate the complex topology inherent in quantum computing systems. This advancement attempts to develop a universal query interface, a feature that allows users to interact with different quantum computing platforms using a single set of commands.

B. Core-Design of sys-sage

In sys-sage, the HPC system is represented using a comprehensive structure that includes components, their hierarchical physical structure known as the *Component Tree*, and *Data Paths*. This structure allows for a detailed and multifaceted representation of the system's architecture and its dynamic characteristics.

Components are fundamental units within the HPC system, each representing distinct hardware or software elements. The component tree captures the hierarchical organization of these components. The tree structure illustrates the static physical composition of the system, detailing how various components, such as processors, memory units, and storage devices, are arranged and interconnected. Inspired by hwloc's method of representing CPU data, sys-sage extends this approach to encompass a more comprehensive array of components, ensuring that all on-node resources, including both hardware and system-related elements, are captured rather than being limited to CPU resources.

In addition to the component tree, sys-sage utilizes Data Paths to provide a more nuanced, less hierarchical and dynamic view of the system. Data Paths encapsulate additional attributes and relationships between a pair of components that extend beyond the hierarchical structure of the component tree. These paths form a *Data Path Graph* – a graph structure which captures information orthogonal to the component tree. While the component tree focuses on the static, physical organization of the system, the data path graph provides insights into dynamic and supplementary relationships, such as performance metrics and communication pathways.

Data paths convey a wide range of information, including, but not limited to, data transfer, performance, power consumption, or even application-specific data. They are designed to model any property regarding the connection or relationship between two components.

The component tree and the data path graph in sys-sage are not necessarily static representations. They are designed to be modifiable at runtime, allowing sys-sage to dynamically adapt to changes within the HPC system.

C. Extensions for Representing Quantum Systems

The preceding section has detailed the core architecture of the sys-sage library, designed to provide an accurate and user-centric representation of heterogeneous HPC systems. By offering intuitive abstractions and high-level representations of quantum systems, our approach prioritizes usability while enabling effective management through both static and dynamic topologies. While quantum systems are fundamentally different in their architecture and their design, their operation and the information needed to track them have very similar properties, both in their static structure and dynamic behavior. We, therefore, extend the existing sys-sage to capture quantum-specific information, enabling us to use established technology as well as form the foundation for hybrid usage covering properties of HPC and QC systems in a single library.

In particular, our extensions focus on the development of a universal query interface, which can be installed alongside matching interfaces for classical system information which enables users to interact with diverse quantum computing platforms through a standardized set of commands. This interface is designed to ensure broad compatibility across various quantum computing technologies, including but not limited to superconducting qubits, neutral atoms, and trapped ion-based systems. Such a capability is critical for unifying the interaction paradigm and simplifying user access in the rapidly evolving quantum computing landscape.

Like its use in HPC systems, sys-sage is adept at collecting and representing static and dynamic information about quantum systems.

1) **Static Information**: This includes immutable details, such as the number of qubits in a particular QC system or backend, and elements like the coupling map for superconducting qubit-based systems (analog to the system hierarchy in classical systems).
2) **Dynamic Information**: This covers dynamic data such as decoherence times or fidelities, which can change over time or with different operational conditions (analogue to dynamic performance properties in classical systems).

sys-sage is able to integrate different sources of partial information in heterogeneous HPC systems, focusing on their dynamic aspects. We leverage this flexibility to extend the same concepts for quantum systems and add new sources covering static and dynamic information from quantum systems. The data can both be read from static files (in the case of cloud-based QCs with periodically published information, like with IBM systems) or directly read from QC systems via online interfaces (in the case of on-site QC systems). Once read in, structured and saved in the sys-sage data structure, the data can be read by many tools such as quantum compilers, circuit mappers or schedulers, and be used for system debugging, monitoring, analysis and optimization.

To enable this functionality we expand on the existing sys-sage structures as well as introduce generalizations that allow us to integrate new properties of quantum systems while maintaining old functionality alongside.

1) ***Quantum Backend:*** In HPCQC architectures, quantum computers function as accelerators integrated into HPC systems, forming an essential part of the overall computational ecosystem. While QCs are treated as components within the system, they are distinguished by their unique characteristics and operational principles, which differ fundamentally from those of classical computing systems.

From a topological perspective, for example, each quantum computer exhibits a distinct set of physical and logical properties that must be accurately represented to enable storage, efficient retrieval, and seamless integration with HPC systems. To address these requirements and align with the concept of HPCQC integration, it is logical to classify QCs as a specialized type of a sys-sage component.

To this end, following the core design of sys-sage, we introduce a new C++ class, *QuantumBackend*, derived from the existing *Component* class, to represent quantum systems within the HPCQC environment. This class encapsulates the unique attributes and functionalities of quantum computers, ensuring their distinctive nature is effectively modeled. Moreover, this approach preserves coherence with the established HPC infrastructure, facilitating seamless integration while leveraging the unique advantages of quantum computing.

2) ***Qubit:*** Qubits serve as the fundamental units of quantum information. To accurately represent their unique characteristics, we implement a new C++ class, *Qubit*, derived from the existing *Component* class. This specialized class encapsulates the distinctive attributes and functionalities of qubits, enabling accurate modeling within the system.

Representing qubits as a distinct type of component is advantageous, as they are hierarchically associated with quantum computers and form the foundational elements of their architecture. However, qubits possess unique properties that necessitate specialized treatment and representation, distinguishing them from other components. This approach ensures that their role within the broader quantum computing framework is accurately modeled while accommodating their specific characteristics.

3) ***AtomSite:*** Quantum systems based on superconducting qubits exhibit significantly different physical characteristics compared to those based on Neutral Atoms (NA) or Trapped Ions (TI). For example, in superconducting qubit systems, coupling maps play a critical role as they define the fixed physical connectivity between qubits. These maps are central to the system's architecture and constrain the implementation of quantum algorithms. In contrast, the concept of coupling maps is less relevant for NA and TI systems, which are inherently more flexible. In these systems, qubits are represented by neutral atoms or ions that can be dynamically positioned, trapped, and shuttled according to the requirements of the quantum circuit. This mobility introduces a level of dynamism absent in superconducting qubit systems, allowing for greater adaptability in quantum operations.

To demonstrate the flexibility and extensibility of sys-sage, we have developed a new class, *AtomSite*, specifically designed to represent quantum backends based on NA and TI technologies. The *AtomSite* class extends the capabilities of the QuantumBackend class by inheriting its foundational features while incorporating additional properties and mappings unique to NA and TI systems. This approach ensures that the dynamic nature and specific requirements of NA and TI-based quantum systems are accurately modeled and efficiently integrated into a broader HPCQC framework.

We extend sys-sage with heterogeneity of the target architectures in mind. The central *QuantumBackend* class stores both technology-agnostic and technology-specific common attributes. The Qubit class represents static and dynamic properties of single qubits across technologies, while the *AtomSite* class handles neutral atoms and trapped ion systems. *QuantumGate* and *CouplingMap* classes (discussed in later subsections) represent connections or gates for qubits or atom sites. Our design hence captures the topologies and attributes of major quantum architectures in a single API.

Figure 1 illustrates an example of a component tree, where different components are represented in various colors. The tree follows a hierarchical structure, with each component depicted as a parent or a child, showcasing their relationships within the system.

4) ***Relations:*** To accurately model the interactions between system components, we have introduced a more versatile abstraction, termed *Relation*. This new representation expands and generalizes the existing Data Paths, which represent information transfer between components. While the *Data Path* abstraction has proven effective in classical HPC systems for modeling interactions, such as data transfers between proces-

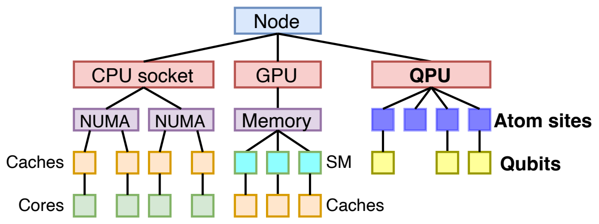

Fig. 1. We extended sys-sage Component tree with quantum-related component types.

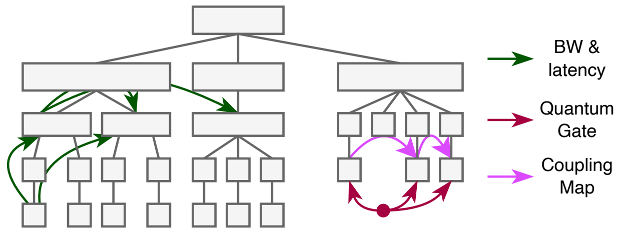

Fig. 2. We propose a generalization of data paths called Relations. It represents sys-sage's data paths as well as Relations of multiple components, such as a multi-qubit quantum gate.

sors, memory units, and other components, its applicability is limited when addressing the unique interaction paradigms of quantum systems.

In classical HPC systems, data paths can be characterized by attributes, such as bandwidth, latency, and energy consumption, and they inherently involve a well-defined source and target. However, quantum computing systems diverge significantly from this paradigm. Interactions in quantum systems are often governed by quantum gates, which operate on one or multiple qubits and are further constrained by, for example, a coupling map—a representation of which qubits can interact with one another. In these systems, the traditional notion of directional data flows, with a clear-cut source and target, does not always apply.

Now, relations between the components are represented by the *Relation* class, a novel C++ abstraction designed to model interactions between components, regardless of whether they belong to a classical HPC system or a QC. This class provides a generalized framework for capturing the concept of interactions, serving as a base class from which specialized interaction types can be derived to address the distinct requirements of various computing paradigms. Thus, *Data Path* is now an inherited class of the *Relation*.

Relations are not limited to addressing the challenges of QC, but are designed as a flexible, versatile abstraction. It provides a unified interface for setting and retrieving properties, such as the type and ID of the interaction and the components involved. By offering broad applicability, the *Relation* class facilitates the integration, simulation, and analysis of hybrid systems that combine classical and quantum elements. This unified representation of interactions enables seamless modeling and analysis across diverse computing domains, paving the way for more comprehensive and interoperable computing architectures.

Figure 2 shows an example configuration with Relations of different types depicted with arrows of different colors. An arrow represents an interaction between two Components, be it HPC information, such as bandwidth and latency, or cache partitioning, or HPCQC-related information, such as Coupling Maps or Quantum Gates.

In summary, it is important to highlight that even though we use a tree structure for the bare topology—thanks to its simplicity and ability to hierarchically decompose information—we also map data to other structures. The *Data Path Graph* concept was already implemented in sys-sage as an orthogonal structure to the tree. For qubit interactions, we introduce *Relations* (specifically *QuantumGate* and *CouplingMap*), which can be seen as graph edges connecting multiple vertices. This orthogonal structure extends and generalizes the *Data Path Graph*, addressing scenarios where a tree-like structure alone is insufficient.

*5) **Quantum Gates**:* Quantum gates are fundamental operations that manipulate qubits in QC. Each type of gate has unique properties and operational characteristics that must be accurately represented. For this, we introduce a new class called *QuantumGate*, derived from *Relation*, to represent quantum gates' operations on qubits. To ensure broad technological applicability, the *QuantumGate* class has been carefully designed to address variations in qubit-pair metrics encountered in superconducting quantum computers, a variability absent in, for example, atom-based systems, where identical atoms result in consistent two-qubit fidelities across all qubits. For instance, the fidelity of a CNOT gate acting on Qubits 1 and 2 may differ from the fidelity of the same CNOT gate operating on Qubits 2 and 3. This variability is particularly pronounced in superconducting qubit systems, where gate properties are often highly dependent on the physical characteristics of the qubits involved. This is because atom-based qubits are quite identical, whereas superconducting-based qubits would differ from each other due to errors during fabrication.

To accurately model and account for these differences, the *QuantumGate* class has been carefully designed to represent both homogeneous and heterogeneous gate scenarios with accuracy and efficiency.

In cases where the properties of a CNOT gate differ between qubit pairs (e.g., Qubits 1 and 2 versus Qubits 2 and 3), these gates are stored independently within the system, each assigned a unique identifier and name. This approach ensures that the heterogeneity of gate properties is accurately captured, facilitating modeling and analysis of quantum gate behavior in diverse quantum computing architectures.

Conversely, if gate properties remain consistent regardless of the qubits they act on, the list of qubits, within the *QuantumGate* class, can remain empty, indicating no specific relation exists between the qubits concerning that quantum gate. This scenario may apply more to systems based on neutral atoms or trapped ions, where uniform gate properties are common. In either situation, the *QuantumGate* class can store and provide all possible gates based on their size (i.e., the number of qubits involved) and type (e.g., CNOT, RZ).

*6) **Coupling Mapping**:* Similarly, *Coupling Mapping* is used to denote the connectivity constraints of a quantum backend. It is denoted as a directed graph, with qubits as nodes and 2-qubit mappings as edges. This information is then used by the compiler during the circuit mapping stage and when transpiling to the supported gate set. This is an important characteristic, especially for superconducting-qubit-based systems. To model this relation between qubits, we have introduced a new class called *CouplingMap*.

III. Capturing Quantum Properties in sys-sage

As indicated in the section above, sys-sage can capture a wide range of static and dynamic system information, store and organize it and make it available to users or other system components. In classical HPC use cases, this is, e.g., used for process placements based on system topology or to identify and track contention on shared resources. For this, the information is read via parsers, processed and structured inside sys-sage and then made available via a shared API to tools or system components.

The extensions for quantum systems work in a similar way. Data is acquired from quantum systems via platform/vendor/system specific information, processed by a new set of parsers, and from there it is ingested into sys-sage and its (extended) data structures. The resulting information is often referred to as *FoMaCs* or “Figures of Merit and Constraints”, as it captures both the hierarchical structure and its constraints as well as quality metrics, which act as optimization targets or figures of merit for the system.

In the following, we discuss the components needed in sys-sage to make this processing possible.

A. Quantum Specific Input Processing

At its core, sys-sage serves as a repository for hardware-relevant information, with the ability to import data from multiple sources. These sources include structured APIs, JSON files, and specialized formats passed from online interfaces to QC systems. This flexibility is achieved through sys-sage’s reliance on *external_interfaces* and customizable *parsers*, enabling developers to tailor data ingestion pipelines according to the specific structure of input data.

On the classical side, sys-sage comes with parsers for various benchmarks and tools, including *hwloc*, *Mt4G*, *CC-CBench*, and *caps-numa-benchmark*. In this work, we extend sys-sage by introducing a new parser, called *IQMParser* for importing data from IQM’s API, allowing the topology and dynamic characteristics of IQM-based quantum systems to be represented effectively.

Beyond data storage, sys-sage plays a critical role as a FoMaC library, which is meant to ingest and analyze the raw data coming from the quantum backends into sys-sage. By pre-processing and analyzing the imported data, the raw hardware data can be transformed into actionable insights for downstream applications, such as compilers and schedulers. For instance, sys-sage can calculate the weights of qubits based on their physical and operational characteristics. These weights, which reflect qubit performance metrics, such as fidelity and coupling strength, provide valuable information for optimizing resource allocation and scheduling in quantum computing workloads.

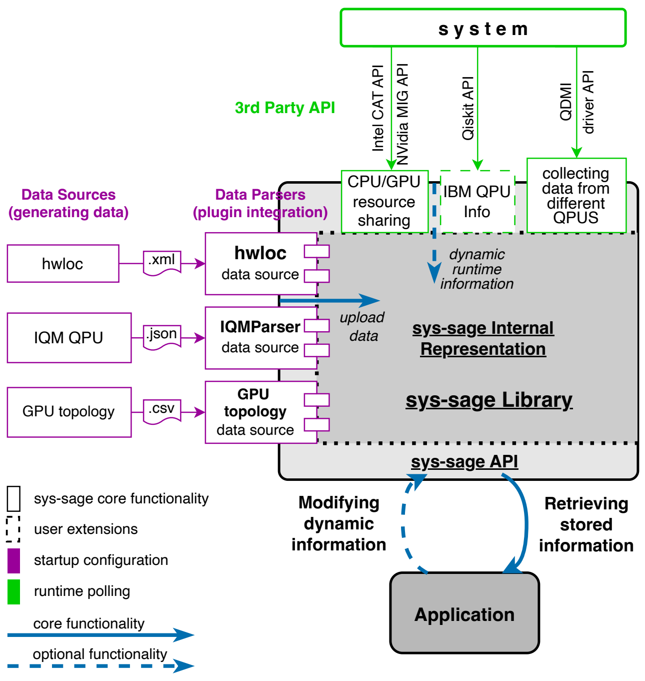

Fig. 3. Integration of system information in sys-sage

B. Data- and Workflow inside sys-sage

To facilitate integration, sys-sage exposes a straightforward API for data parsing and processing. Functions like *parseIQM* can be used to import data for IQM from specified sources (e.g., JSON files containing calibration data of IQM Quantum Backends) or the Python-based IQM API. This can then be associated with an instance of *Component* or *QuantumBackend*. The following highlights illustrate the processing workflow:

1) **Data Parsing and Topology Generation**: The *parseIQM* function initializes the parsing process by reading the specified data source and generating the topology for quantum systems. The internal representation is then populated by the properties queried by the parser. This step ensures that the physical and logical structure of the system is accurately represented within sys-sage’s internal hierarchy.
2) **Dynamic Data Integration**: The parser supports importing dynamic data, such as calibration parameters or runtime state changes. This information is timestamped and stored in the internal representation, enabling time-series analyses of qubit and system characteristics.
3) **Weight Calculation and Analysis**: After parsing, users access the sys-sage API, for instance, to compute

performance-related metrics for system components. These metrics are stored back in sys-sage where they are accessible to other tools for further optimization or visualization. They do not reside in a static data structure because they might only be useful for that specific quantum backend. Instead, they are stored as attributes.

4) **Hierarchical Representation and Querying**: The internal representation, built as a hierarchy of components, allows efficient querying and manipulation. For instance, qubits and their relationships (e.g., coupling maps) are modeled explicitly, with dynamic attributes like weights and histories accessible for post-processing.

Fig. 3 summarizes the integration of system information in sys-sage. There are two options for integrating the data. The purple arrows depict the integration of data, primarily already available before the startup (such as JSON files) and the green arrows for data available at runtime (such as the Quantum Device Management Interface). Additionally, blue arrows show how the user interacts with the library.

The introduction of the *IQMParser* demonstrates sys-sage's extensibility and versatility. This parser leverages sys-sage's modular design to integrate new data sources seamlessly, ensuring compatibility with evolving quantum computing technologies. Furthermore, by maintaining a coherent representation of both static and dynamic contexts, sys-sage facilitates the convergence of classical HPC and quantum computing paradigms.

IV. A Real-World Use Case

In this section, we illustrate a real-world use case on a physical quantum computer of sys-sage being used to calculate the weights of the qubit according to physical properties.

This use-case presents and evaluates sys-sage as a FoMaC library within the Munich Quantum Software Stack (MQSS) [25], the novel full software stack output of the Munich Quantum Valley (MQV) initiative. In this context, sys-sage can provide the topological information regarding the quantum systems to compilers or schedulers, and it handles importing quantum system information from different formats from different vendors. Specifically for this use-case, we use sys-sage's IQMParser to parse the input and build the quantum computer representation in sys-sage, including dynamic calibration information over different time periods. We utilize sys-sage to preprocess the raw data to calculate valuable quality metrics, which can, in turn, be used by quantum compilers, schedulers, or circuit mappers [26] to make insightful decisions when mapping the logical circuit to physical qubits on a device.

Our experiments were performed on a quantum computer built by IQM based on superconducting qubits [27], which is housed at the Leibniz Supercomputing Centre (LRZ), inside the compute cube of LRZ (see Figure 4).

Different types of QPUs have—by design and based on their architecture—different constraints, such as a limited set of native gates and limited qubit-qubit connections. Therefore not all pairs of qubits can interact directly via two-qubit gates. A common solution to this challenge is to introduce SWAP gates into the quantum circuit, which sequentially swap qubits' positions until the two-qubit gate can be executed as physically required. Optimizing the arrangement of qubits through such routing is crucial for the efficient execution of quantum algorithms [28].

In a noisy intermediate-scale quantum (NISQ) device, the coherence times, gate fidelities and readout errors of individual qubits can vary significantly, especially when QPUs are automatically re-calibrated in HPC centers [29]. Additionally, some qubits may perform better than others for certain types of operations or circuits. Consequently, it is advantageous to assess and identify the qubits with superior performance characteristics for a particular operation or to determine optimal qubit layouts before performing mapping circuits to a physical QPU.

In order to enable such optimizations, we must assign a quality-weighted metric to each qubit, incorporating factors such as coherence times, gate fidelities, and readout errors, along with their configurable importance. This converts raw measurements, as we get them from QC systems, to actionable FoMaCs properties that can be consumed by system components, such as compilers. To establish the baseline quality of qubits we include relaxation time $T1$, dephasing time $T2$, 1-qubit gate fidelity $1F$, 2-qubit gate fidelity $2F$ and readout fidelity RF and assign them configurable coefficients as follows:

$$\begin{aligned} W_q &= \alpha \min\left(\frac{T1_q}{\max(T1_q)}, \frac{T2_q}{\max(T2_q)}\right) \\ &\quad + \beta 1F_q + \gamma RF_q + \delta\langle 2F_q\rangle, \end{aligned} \tag{1}$$

where q is the qubit index, α, β, γ and δ are weights, respectively, and $\langle ... \rangle$ denotes the average over the number of the qubit q's neighbors. Intuitively, higher fidelities lead

Fig. 4. 20-qubit IQM Quantum Computer inside the LRZ compute cube (photo credit: LRZ).

to higher weights, and longer $T1$ and $T2$ times mean less energy decay and better coherence, and thus higher weights. We choose the minimum between $T1$ and $T2$ and normalize them for consistency.

1) Determining Weights: The next step is the assignment of weights to α,β, γ and δ. We choose a flexible weighting approach, which encompasses various schemes tailored to different circuit types, as well as the dynamic updating of calibration data. For illustrative purposes, we present two examples: one involving a Variational Quantum Eigensolver (VQE) [30] circuit and the other focused on Trotter evolution [31].

a) VQE: VQE circuits are characterized by shallow depths, repeated measurements (as different parts of the target Hamiltonian must be measured), and the use of parameterized gates [32]. In this context, the accuracy of measurements and gate fidelity (especially two-qubit gates) are more critical than coherence time. In VQE, we need accurate expectation values based on multiple measurements, and the results directly affect the classical optimization. We also need entangling gates to achieve correlations between different qubits. Errors in the gates directly affect the energy estimation. Given these considerations, we assign a weight of 40% to readout fidelity due to the importance of repeated measurements. For two-qubit gate fidelity, a weight of 30% is allocated, while single-qubit gate fidelity is assigned 20%. Coherence time is assigned a lower weight of 10%, reflecting the shallow depth of the circuit.

This weighting scheme can be dynamically adjusted based on many factors, such as changes in the number of measurements (shots), the number of variational parameters, and the depth of the circuit. This constitutes a measurement-heavy weighting approach, wherein the relative importance of each quality metric is responsive to the specific requirements of the quantum algorithm and its evolving characteristics.

b) Trotter Evolution: In contrast, the weighting scheme for Trotter evolution differs significantly. In the Trotterization of the Hamiltonian, the evolution time is usually divided into many small time steps. Each time step involves applying a series of gates from different terms of the target Hamiltonian. Smaller time steps improve the accuracy of the evolution, but they also result in deeper circuits and longer overall execution times. Decoherence errors tend to accumulate exponentially with circuit depth, while gate fidelity plays a crucial role in maintaining the quantum state throughout the evolution process.

Given these characteristics, the most important weighting factor is coherence time. The secondary weighting is assigned to two-qubit gate fidelity, followed by single-qubit gate fidelity. We assign the following weights: $[0.5, 0.3, 0.1, 0.1]$ to $[T2/T1, 2F, 1F, RF]$, respectively. As with VQE, dynamic adjustments can be made to the weighting scheme based on specific Trotter parameters. This results in a gate-heavy weighting approach, where the relative importance of each quality metric is tailored to the requirements of the Trotter evolution, with a particular emphasis on the fidelity of the gates and the coherence time.

It is worth mentioning that the weight assignment can be easily automated, rather than relying on manual assignment of values. For example, the ratio between the number of single-qubit gates and two-qubit gates, or the circuit depth, can offer valuable insight into assigning appropriate weights to the single-qubit, two-qubit, and readout fidelities. The weights in our equation do not account for connectivity between qubits and do not capture other complex errors, such as cross-talks, so the actual performance may vary. However, the accuracy of these weights can be approximated or further refined based on circuit-specific parameters.

2) Experimental Validation: Having established the parameters for calculating qubit weights, we apply our method to real-world data from the IQM quantum computer at LRZ with 20 qubits.

Using the IQMParser introduced in Section III, we utilize JSON files containing calibration data of IQM Quantum Backends to collect key properties, including times, readout fidelities, 1-qubit fidelities, and 2-qubit fidelities, over 54 days. This data is stored in the sys-sage's internal representation, enabling subsequent normalization and weight calculations. Importantly, the qubit weights were computed and maintained in the FoMaC component of the library, external to the internal representation, avoiding redundant storage of transient values.

Figure 5 illustrates the calculated and normalized qubit weights in a heatmap, with timestamps on the vertical axis and qubit indices on the horizontal axis. The normalization allows us to pick up even very small variations and differences between qubits in the calculated weights, enabling a nuanced analysis. The coupling map of the qubits, depicted in Figure 6, provides further insight into the physical connectivity between qubits.

Our analysis of the dynamic weights demonstrates that recalibration can significantly affect the suitability of certain qubits for specific quantum circuits. For instance, on 05.12.2024, qubits 6, 10, and 11 exhibit weights that sug-

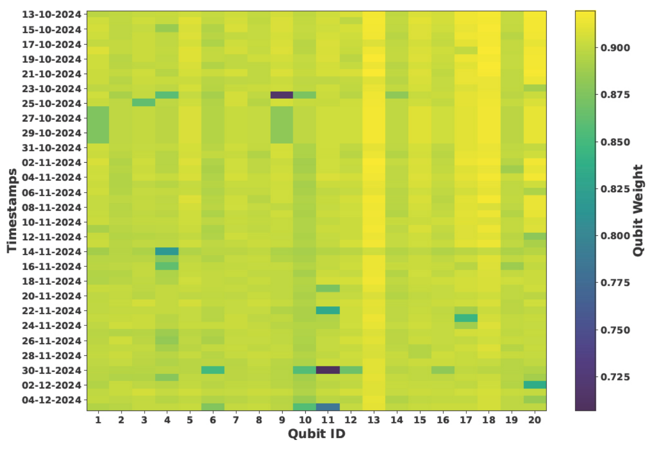

Fig. 5. Normalized Qubit Weight Distribution (higher is better)

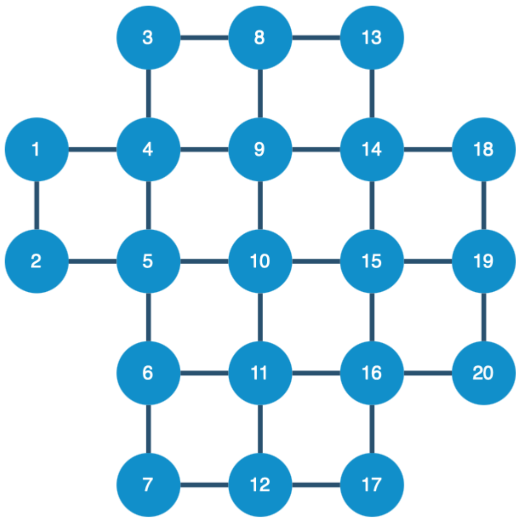

Fig. 6. Connectivity of the Qubits

gest they are less suitable for executing Variational Quantum Eigensolver (VQE) circuits. This observation highlights the utility of dynamic qubit weights in optimizing circuit placement and scheduling, ensuring the reliability and efficiency of computations.

V. Towards a Complete HPCQC Stack

Within the MQSS framework, a hierarchical three-level scheduling architecture has been proposed to efficiently manage HPC-QC hybrid systems. At the top level, SLURM is responsible for scheduling entire jobs across the HPC infrastructure. At the lowest level, quantum control systems utilize QC-specific schedulers to manage and orchestrate individual quantum operations, commonly referred to as “pulses”. The focus of the current work lies on the intermediate scheduler, which plays a critical role in coordinating among various quantum devices and their respective topologies to ensure optimal resource utilization and workflow integration.

This intermediate scheduler is tasked with selecting the most suitable quantum device for executing a given quantum job. To perform this effectively, it relies heavily on input from sys-sage, a system information service that provides detailed metrics and interfaces about the available hardware resources. The data provided by sys-sage informs the scheduler’s decision-making process, particularly in assessing the compatibility and efficiency of mapping quantum circuits onto different device architectures.

A key function of this scheduler is to optimize target device selection based on the computational effort required to adapt an arbitrary quantum circuit to each device’s physical layout, thereby reducing Just-In-Time (JIT) compilation overhead. For example, if sys-sage data indicates that mapping a circuit to the topology of device A would result in higher latency or resource usage compared to device B, and no other operational constraints override this consideration, the scheduler would prioritize device B. This informed selection process contributes to improved system throughput and responsiveness in heterogeneous QC environments.

VI. Performance Evaluation

Apart from the broad usability, performance is also a critical aspect in QC systems, especially considering that compilation takes place at run time. Hence, fast access and low memory consumption are needed. This section analyzes the performance of our use-case implementation.

The main steps of the presented workflow are: (1) creating the static topology of the machine, which means a representation of its qubits and the coupling maps of the system; (2) the dynamic information, such as decoherence times and fidelities, are updated in the existing static topology; (3) with the new information, the FoMaC computes updated weights for each qubit – these depend on the dynamic information from the qubit itself and its coupling maps; and finally (4) as an example of further integration, the FoMaC searches for n ($n = 10$) qubits with the highest weights to pass them to a scheduler.

To show the scalability of our implementation, we measure the times needed for these operations on artificially generated inputs of up to 25,600 qubits. We perform the measurements on an Apple M2 Pro (P-core), 6 P-cores, 32 MB LLC, running macOS Sonoma 14.6.1. Our inputs are in the form of full 2-dimensional mesh, where each Qubit has four coupling maps to its top, bottom, left, and right neighbors.[2]

Figure 7 presents the operation times **per Qubit** for different input sizes. The main observation important for scaling is that the time needed to process the steps is constant per-qubit, meaning it is linear in the number of qubits. The largest item, ranging from 24 to 30 µs, is the total parsing time, including parsing the input JSON file, and both the static and dynamic updates. Given the static and dynamic parsing take roughly 10 µs in total, the majority of the time is spent in parsing the JSON file. Providing other interfaces than JSON-encoded data might, therefore, speed up the process. Both the static and the dynamic part of the topology creation take ca. 5 µs

[2]Qubits on the edge of the topology have fewer coupling maps.

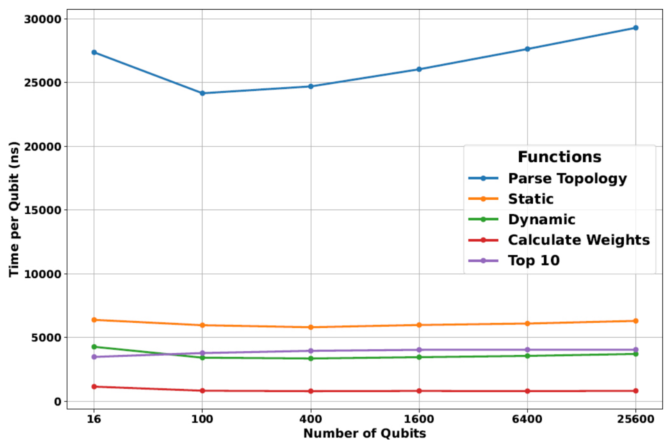

Fig. 7. Performance Metrics for Quantum Operations

per qubit, as well as retrieving the top 10 qubits with highest weights. Finally, the FoMaC functionality to calculate weights, as discussed in Section IV, needs less than $5\,\mu s$ per qubit. In absolute numbers, parsing the topology takes $2\,\mathrm{ms}$ and $750\,\mathrm{ms}$ and calculating all qubit weights takes $82\,\mu s$ and $21\,\mathrm{ms}$ for 100 and 25600 qubit-systems, respectively.

a) Memory requirements: In terms of memory usage, both qubits and coupling maps are sys-sage objects, which need to be allocated in the memory. Therefore, the memory complexity is linearly dependent on the number of qubits and coupling maps in the topology. Our synthetic topology of n qubits has $2n - (2\sqrt{n})$ coupling maps. Each qubit allocates ca. $190\,\mathrm{B}$[3] and each coupling map allocates ca. $60\,\mathrm{B}$ of data. In terms of overall data allocation, the 100-qubit, 180-coupling map system requires about $30\,\mathrm{kB}$, and the 25600-qubit, 50880-coupling map system fits in $8\,\mathrm{MB}$.

VII. Related Work

The extension of sys-sage draws inspiration from several related works in classical systems. One of its foundational influences is *hwloc* [33], which provides a hierarchical representation of computing hardware, detailing relationships between CPUs, memory, and devices to optimize resource allocation and task scheduling. Building upon *hwloc*, sys-sage addresses its limitations for modern workloads by integrating dynamic information, connecting various topology-related data, and generalizing its representation.

In quantum computing, Qiskit offers an API for creating, executing, and managing quantum circuits [34]. Qiskit is an open-source quantum computing framework enabling users to design, compile, and run quantum programs on simulators or quantum processors via an online platform. However, like other proprietary solutions, Qiskit is mostly compatible with IBM quantum solutions. Through its extensive API, Qiskit allows querying the physical properties and topologies of IBM systems.

When it comes to open-source platforms, CUDA-Q is a quantum development platform that allows the integration and programming of QPUs, GPUs, and CPUs within a unified system [35]. CUDA-Q differs from other such platforms by being qubit and QPU-agnostic in nature, thus enabling the researchers to execute quantum algorithms on different QPUs or GPU-accelerated simulators.

Besides these programming frameworks, researchers are also working on state-of-the art integration frameworks that facilitate a smooth integration of QPUs with classical computers. Beck et al. [36] present a comprehensive, hardware-agnostic framework designed to integrate QC resources into existing HPC ecosystems. Building on the expertise of Oak Ridge National Laboratory and aligned with the mission objectives of the U.S. Department of Energy, their approach strategically incorporates QC to enhance classical HPC workflows through detailed analyses, benchmarking, and code optimization. The framework fosters a synergistic environment for seamless interaction between quantum and classical systems, addressing key challenges such as resource management, network latency, and heterogeneous programming environments. A cornerstone of this integration effort is the Quantum Computing User Program (QCUP), which provides 271 users with access to a variety of quantum systems. Despite these advancements, integrating multiple QC platforms into classical HPC workflows remains complex. To address this, the authors introduce a prototype—Quantum Framework (QFw)—aimed at mitigating issues related to resource allocation, application deployment, and the coupling of QC resources with traditional HPC infrastructure.

In parallel research, Cranganore et al. [37] have proposed a comprehensive framework for hybrid quantum-classical scientific workflows. Their investigation examines the potential computational advantages offered by quantum integration for addressing computationally intensive tasks, including molecular dynamics simulations, quantum chemistry calculations, optimization problems, and machine learning applications. The authors present a detailed software architecture designed to facilitate efficient management of these integrated computational processes, through a molecular dynamics use-case.

The Quantum Device Management Interface (QDMI), a core component of the MQSS, is presented in [38]. Traditional quantum software tools are typically hard-coded for specific hardware platforms and lack the flexibility to dynamically adapt to varying physical characteristics and constraints. QDMI addresses this limitation by enabling hardware platforms to expose their physical properties in a standardized format, which can be queried by software tools. This facilitates a more adaptive and efficient compilation process, allowing software to automatically tailor its behavior to different hardware environments and optimize performance based on platform-specific constraints.

Similarly to the other integration frameworks, sys-sage aims to ease the integration of upcoming quantum technologies with classical HPC systems. Compared to vendor-specific APIs, sys-sage offers universality, reusability, and ease of use by unifying multiple interfaces—whether it is Qiskit or separate vendor-specific APIs for topology and dynamic data aggregation—thus simplifying integration. sys-sage acts as a layer atop their data acquisition capabilities, offering refinement, analysis, and unifying abstractions rather than direct competition with the vendor-specific programming frameworks. For example, we extend sys-sage to make it compatible with QDMI v0.1, to access quantum devices and provide topological information to other tools within MQSS, and with Qiskit and IQM, vendor-specific interfaces, which were necessary for conducting our experiments. Under the hood, sys-sage can use QDMI, Qiskit, and IQM interfaces for acquiring static and dynamic information about underlying quantum hardware and post-process it in the same fashion to provide valuable insights to other tools.

[3]Storing historical information requires additional $48\,\mathrm{B}$ per snapshot per Qubit.

VIII. Conclusions

In this work, we have extended the capabilities of sys-sage to enhance its functionality for representing and interacting with quantum systems. The primary objective was to create a unified interface that integrates diverse quantum technologies, enabling effective management through both static and dynamic topologies. By providing a high-level abstraction for quantum systems, our approach facilitates improved interoperability and usability across various quantum computing tasks.

The extensions introduced to sys-sage emphasize universality and compatibility with existing tools and libraries. We proposed a comprehensive API framework for creating and interacting with quantum system representations, with a focus on topology generation. These developments include the introduction of new C++ classes, such as *QuantumBackend* and *Qubit*, derived from the foundational *Component* class. Additionally, we introduced a novel abstraction, *Relation*, to model interactions between HPC or QC components. Derived classes like *DataPath* (for HPC systems) and *QuantumGate* (for QC systems) further enrich this abstraction. These additions aim to provide an efficient and precise representation of HPC-QC systems, bridging the gap between classical and quantum computing domains.

Beyond representing and caching quantum backend data, sys-sage also preprocesses this information to calculate meaningful metrics for use in tools such as compilers, schedulers, and mappers. Its role as a FoMaC library was highlighted through its ability to provide precomputed insights for various quantum computing tasks. For example, we demonstrated its utility by calculating qubit weights for IQM's quantum systems using real-world data. We presented the methodology used for calculating these metrics and the resulting distribution of qubit weights. Furthermore, we investigated the performance implications of topology generation and retrieval, observing that the introduced extensions incurred only minimal computational overhead.

A key focus of this study was exploring the potential for extending sys-sage to support HPC systems and facilitate integration with other tools. Such integration holds the promise of significantly enhancing the functionality and interoperability of quantum computing ecosystems, fostering a more cohesive environment for quantum research and development.

Future efforts could prioritize expanding sys-sage's integration with additional quantum computing platforms and tools, incorporating more diverse quantum technologies, and addressing further use cases as a FoMaC library. Enhanced post-processing and advanced analyses of retrieved data, including correlation studies, are also avenues for exploration. Finally, testing and refining the library for unified HPCQC systems could unlock its full potential. For instance, users could retrieve combined topologies of classical and quantum systems to optimize task scheduling and resource allocation. This unification promises more efficient utilization of quantum computing resources, fostering a future where HPC and QC systems operate in synergy, thereby advancing computing capabilities and accelerating scientific discovery.

Acknowledgements

The DEEP-SEA project has received funding from the European Union's Horizon 2020/EuroHPC research and innovation programme under grant agreement No 955606. National contributions, in Germany from BMBF, match the EuroHPC funding. The work was further funded by the Munich Quantum Valley (MQV), which is supported by the Bavarian State Government with funds from the Hightech Agenda Bayern.

References

[1] J. Yin, Y.-H. Li, S.-K. Liao, M. Yang, Y. Cao, L. Zhang, J.-G. Ren, W.-Q. Cai, W.-Y. Liu, S.-L. Li *et al.*, "Entanglement-based secure quantum cryptography over 1,120 kilometres," *Nature*, vol. 582, no. 7813, pp. 501–505, 2020.

[2] H. Shang, L. Shen, Y. Fan, Z. Xu, C. Guo, J. Liu, W. Zhou, H. Ma, R. Lin, Y. Yang *et al.*, "Large-scale simulation of quantum computational chemistry on a new sunway supercomputer," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–14.

[3] B. Poggel, N. Quetschlich, L. Burgholzer, R. Wille, and J. M. Lorenz, "Recommending solution paths for solving optimization problems with quantum computing," in *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2023, pp. 60–67.

[4] A. Elsharkawy, X.-T. M. To, P. Seitz, Y. Chen, Y. Stade, M. Geiger, Q. Huang, X. Guo, M. A. Ansari, C. B. Mendl *et al.*, "Integration of quantum accelerators with high performance computing - a review of quantum programming tools," *arXiv preprint arXiv:2309.06167*, 2023.

[5] T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, pp. 15–23, 2021.

[6] M. Schulz, M. Ruefenacht, D. Kranzlmüller, and L. B. Schulz, "Accelerating hpc with quantum computing: It is a software challenge too," *Computing in Science & Engineering*, vol. 24, no. 4, pp. 60–64, 2022.

[7] O. Ezratty, "Perspective on superconducting qubit quantum computing," *The European Physical Journal A*, vol. 59, no. 5, p. 94, 2023.

[8] K. Wintersperger, F. Dommert, T. Ehmer, A. Hoursanov, J. Klepsch, W. Mauerer, G. Reuber, T. Strohm, M. Yin, and S. Luber, "Neutral atom quantum computing hardware: performance and end-user perspective," *EPJ Quantum Technology*, vol. 10, no. 1, p. 32, 2023.

[9] H. Haffner, C. F. Roos, and R. Blatt, "Quantum computing with trapped ions," *Physics Reports-Review Section of Physics Letters*, vol. 469, no. 4, pp. 155–203, 2008.

[10] H. Goto, "Double-transmon coupler: Fast two-qubit gate with no residual coupling for highly detuned superconducting qubits," *Physical review applied*, vol. 18, no. 3, p. 034038, 2022.

[11] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020.

[12] L. Schmid, S. Park, and R. Wille, "Hybrid circuit mapping: Leveraging the full spectrum of computational capabilities of neutral atom quantum computers," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3655959

[13] M. Saffman, T. G. Walker, and K. Mølmer, "Quantum information with rydberg atoms," *Reviews of modern physics*, vol. 82, no. 3, pp. 2313–2363, 2010.

[14] H. Wang, P. Liu, D. B. Tan, Y. Liu, J. Gu, D. Z. Pan, J. Cong, U. A. Acar, and S. Han, "Atomique: A quantum compiler for reconfigurable neutral atom arrays," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 293–309.

[15] L. Schmid, D. F. Locher, M. Rispler, S. Blatt, J. Zeiher, M. Müller, and R. Wille, "Computational capabilities and compiler development for neutral atom quantum processors—connecting tool developers and hardware experts," *Quantum Science and Technology*, vol. 9, no. 3, p. 033001, 2024.

[16] D. Maslov, "Basic circuit compilation techniques for an ion-trap quantum machine," *New Journal of Physics*, vol. 19, no. 2, p. 023035, 2017.

[17] M. Bandic, S. Feld, and C. G. Almudever, “Full-stack quantum computing systems in the nisq era: algorithm-driven and hardware-aware compilation techniques,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1–6.

[18] X. Ren, J. Wan, Z. Liang, and A. Barbalace, “Tackling coherent noise in quantum computing via cross-layer compiler optimization,” *arXiv preprint arXiv:2410.09664*, 2024.

[19] G. Li, Y. Shi, and A. Javadi-Abhari, “Software-hardware co-optimization for computational chemistry on superconducting quantum processors,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 832–845.

[20] L. Lao and D. E. Browne, “2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 351–365.

[21] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, “Quantum-assisted quantum compiling,” *Quantum*, vol. 3, p. 140, 2019.

[22] S. Vanecek and M. Schulz, “sys-sage: A unified representation of dynamic topologies & attributes on hpc systems,” in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, pp. 363–375.

[23] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, “A portable programming interface for performance evaluation on modern processors,” *The international journal of high performance computing applications*, vol. 14, no. 3, pp. 189–204, 2000.

[24] J. Treibig, G. Hager, and G. Wellein, “Likwid: A lightweight performance-oriented tool suite for x86 multicore environments,” in *2010 39th international conference on parallel processing workshops*. IEEE, 2010, pp. 207–216.

[25] M. Schulz, L. Schulz, M. Ruefenacht, and R. Wille, “Towards the munich quantum software stack: Enabling efficient access and tool support for quantum computers,” in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 2. IEEE, 2023, pp. 399–400.

[26] R. Wille and L. Burgholzer, “Mqt qmap: Efficient quantum circuit mapping,” in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 198–204.

[27] X. Deng, S. Pogorzalek, F. Vigneau, P. Yang, M. Schulz, and L. Schulz, “Calibration and performance evaluation of a superconducting quantum processor in an hpc center,” in *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, 2024, pp. 1–9.

[28] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the Qubit Routing Problem,” in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), W. van Dam and L. Mančinska, Eds., vol. 135. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 5:1–5:32. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TQC.2019.5

[29] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.

[30] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[31] I. M. Georgescu, S. Ashhab, and F. Nori, “Quantum simulation,” *Reviews of Modern Physics*, vol. 86, no. 1, pp. 153–185, 2014.

[32] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, “Noisy intermediate-scale quantum algorithms,” *Reviews of Modern Physics*, vol. 94, no. 1, p. 015004, 2022.

[33] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, “hwloc: A generic framework for managing hardware affinities in hpc applications,” in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 180–186.

[34] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp *et al.*, “Qiskit backend specifications for openqasm and openpulse experiments,” *arXiv preprint arXiv:1809.03452*, 2018.

[35] Y.-Y. Hong, D. J. D. Lopez, and Y.-Y. Wang, “Solar irradiance forecasting using a hybrid quantum neural network: A comparison on gpu-based workflow development platforms,” *IEEE Access*, 2024.

[36] T. Beck, A. Baroni, R. Bennink, G. Buchs, E. A. C. Pérez, M. Eisenbach, R. F. da Silva, M. G. Meena, K. Gottiparthi, P. Groszkowski, T. S. Humble, R. Landfield, K. Maheshwari, S. Oral, M. A. Sandoval, A. Shehata, I.-S. Suh, and C. Zimmer, “Integrating quantum computing resources into scientific hpc ecosystems,” *Future Generation Computer Systems*, vol. 161, pp. 11–25, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24003583

[37] S. S. Cranganore, V. De Maio, I. Brandic, and E. Deelman, “Paving the way to hybrid quantum–classical scientific workflows,” *Future Generation Computer Systems*, vol. 158, pp. 346–366, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24001596

[38] R. Wille, L. Schmid, Y. Stade, J. Echavarria, M. Schulz, L. Schulz, and L. Burgholzer, “Qdmi-quantum device management interface: Hardware-software interface for the munich quantum software stack,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 2. IEEE, 2024, pp. 573–574.