A Conceptual Architecture for a Quantum-HPC Middleware

Nishant Saurabh[*], Shantenu Jha[†‡], Andre Luckow[§¶]
[*]Utrecht University, NL
[†]Rutgers University, NJ, US
[‡]Brookhaven National Lab, NY, US
[§]Ludwig Maximilian University Munich, Germany
[¶]BMW Group, Munich, Germany
[*]n.saurabh@uu.nl, [†]shantenu.jha@rutgers.edu, [‡]shantenu@bnl.gov, [§]andre.luckow@ifi.lmu.de, [¶]andre.luckow@bmwgroup.com

2023 IEEE International Conference on Quantum Software (QSW) | 979-8-3503-0479-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/QSW59989.2023.00023

***Abstract*—Quantum computing is important for science and industry as it offers the potential to solve certain complex problems and perform calculations significantly faster than classical computers. Quantum computing systems evolved from monolithic systems towards modular architectures comprising multiple quantum processing units (QPUs) coupled to classical computing nodes (HPC). With the increasing scale, middleware systems that facilitate the efficient coupling of quantum-classical computing are becoming critical. Through an in-depth analysis of quantum applications, integration patterns and systems, we identified a gap in understanding Quantum-HPC middleware systems. We present a conceptual middleware to facilitate reasoning about quantum-classical integration and serve as the basis for a future middleware system. A key contribution of this paper lies in leveraging well-established high-performance computing abstractions for managing workloads, tasks, and resources to seamlessly integrate quantum computing into HPC systems.**

***Index Terms*—Quantum Computing, HPC, Middleware**

I. Introduction

Quantum computing promises to accelerate complex computation for many applications. Quantum algorithms utilize quantum information sciences and promise speedups by requiring fewer steps than their classical counterparts. Many applications amenable to quantum computing traditionally utilize high-performance systems, simulations, machine learning, and optimization techniques [1].

Algorithms for both *Noisy Intermediate Scale Quantum Computers (NISQ)* and *Fault-tolerant Quantum Computers (FTQC)* require the coupling of quantum and classical systems. For example, variational algorithms [2] depend on classical optimization and quantum error correction codes require significant classical computation of the syndrome measurements.

The increasing maturity and modularity of quantum hardware is giving rise to the question of how to integrate quantum systems into HPC systems [3]–[6]. In the future, quantum computers will tightly integrate quantum processing units (QPUs) with classical HPC resources. Some NISQ devices are available in HPC centers [7] and the cloud [8]. While the focus is often on hardware- and network-level integration, software integration has received less attention.

In this paper, we explore the requirements of quantum applications and the need and types of integrations with classical HPC computing. We discuss and analyze 17 quantum application scenarios across optimization, machine learning, and simulation domains. The qualitative findings underscore the importance of categorizing these scenarios into three distinct integration patterns: *HPC-for-Quantum*, *Quantum-in-HPC*, and *Quantum-about-HPC*. For each scenario, we investigate different characteristics, such as the coupling of quantum and classical tasks, and the resulting requirements for Quantum-HPC middleware. Our examination of the current state-of-the-art reveals significant limitations in the currently fragmented quantum software and middleware landscape, particularly concerning the ability to manage the application complexity and heterogeneous resources at scale.

To address these challenges, we design a conceptual middleware system that facilitates seamless interaction between classical (HPC) and quantum resources. Our conceptual middleware enables quantum-classical integration by providing unified resource access and management that enables applications to flexibly allocate resources and manage quantum and classical tasks. The middleware utilizes proven high-performance computing abstractions to manage workloads, tasks, and resources to enable the seamless integration of quantum computing into HPC systems.

Using a chemistry application workflow as an example, we demonstrate the middleware system's ability to support the three integration patterns. This framework lays the foundation for developing a robust middleware system capable of managing and optimizing quantum applications in a unified Quantum-HPC computing environment.

This paper is structured as follows: Section II presents an overview of quantum applications and algorithms. We identify three integration patterns between classical (HPC) and quantum tasks in section III. We investigate how current quantum software and Quantum-HPC middleware systems address the challenges and needs of the identified integration patterns in section IV. We propose a conceptual middleware for Quantum-HPC workflows in section V. Section VI concludes with a discussion of the results and future work.

II. Applications and Algorithms

In this section, we discuss the application scenarios and algorithms.

979-8-3503-0479-4/23/$31.00 ©2023 IEEE
DOI 10.1109/QSW59989.2023.00023

A. Optimization

Quantum algorithms, such as quantum annealing [9] and quantum approximate optimization [10], are gaining traction in solving complex optimization problems.

1) Use Cases: Optimization use cases include solving combinatorial optimization problems [10] in scheduling [11], logistics [12], and transportation application domains [1]. Quantum optimization also finds usage in quantum chemistry and drug discovery processes [13], [14]. Further, quantum computing can help optimize renewable energy system operations [15], robotics (e. g., route optimization [16]), and machine learning (e. g., optimizing trainable parameters for image classification and NLP [17]).

2) Algorithms: such as quantum annealing (QA) [9] map the optimization problem [18], [19] onto an Ising model to explore the solution space. In contrast, Quantum Adiabatic Evolution (QAA) [20] encodes the optimization problem [21] into the ground state of a known Hamiltonian. Further, it adiabatically evolves the system through a path of Hamiltonians to explore the solution space. Algorithms, such as the quantum approximate optimization algorithm (QAOA) [10], including enhancements like warm-starting [22] and recursive QAOA (RQAOA) [23], use a combination of quantum and classical resources to find approximate solutions to combinatorial optimization problems. FTQC algorithms, such as Grover algorithm [24] can also be used for optimization [25].

B. Machine Learning

Quantum computing can improve different parts of machine learning applications, e. g., linear algebra routines and generative machine learning methods.

1) Use Cases: Quantum machine learning (QML) use cases are often categorized based on the type of input data, i. e., classical, quantum, and the type of algorithm, i. e., supervised, unsupervised, and generative. A typical QML use case with quantum data involves using quantum data for learning phases in many-body physics simulations [26].

QML applications using classical data [27] include unsupervised clustering (e. g., the Sloan digital sky survey data, X-ray and weather data [28]). Generative use cases of QML involve creating scientific simulation datasets (e. g., Monte Carlo events for particle physics process simulations [29], [30]) and chemical synthesis for molecular simulations [31]. QML also finds usage in accelerating many-body Hamiltonian simulations [32]), enhancing quantum many-body simulations [33]–[35], control in quantum routines [36], and optimizing quantum compilation [37].

2) Algorithms: FTQC algorithms, such as HHL [38] and QPE [39], can be used for linear algebra subroutines (e. g., eigenvalues estimation, matrix inversion), distance computation between two quantum states [40] and finding closest neighbors [27]. Unsupervised FTQC approaches, such as Q-means [41], can identify data clusters and support the nearest centroid classification [42].

There also exist variational algorithms for QML, e. g., variational quantum linear solvers (VQLS) [43], differential quantum circuits (DQC) [32], and quantum neural networks (QNN) [44]. While VQLS and DQC are utilized for linear algebra problems, QNNs are used similarly to classical neural networks for classification problems, e. g., recognizing quantum states [45]. Quantum kernel methods [46] learn a kernel function that maps data into a higher-dimensional Hilbert space using a variational circuit.

Generative QML techniques, such as Quantum Circuit Born Machines (QCBM) [47], [48] and Quantum GANs (QGAN) [49], demonstrated comparable training performance to classical models, requiring fewer parameters [50]. They can be utilized to generate data samples (e. g., input data for Monte Carlo simulations or generating new molecular states in quantum chemistry). QML is also applicable for surrogate modeling (e. g., FermiNet [51]).

C. Simulation

Quantum computers promise an exponential advantage in simulating quantum mechanical systems [52]. For classic numerical simulations and modeling complex systems in science and engineering, promising algorithms, e. g., HHL [38], exist.

1) Use Cases: Application domains for simulation include material science, viz., the design of new materials, optimization of materials properties, and predicting material behavior. Quantum simulation methods can be used to identify and design new compounds with desired medicinal properties, predicting the potential side effects of drugs [14].

Quantum computers can also be used for classical numerical simulations [53], e. g., to study climate models [54], to perform complex aerospace simulations [55], such as 3D computational fluid dynamics (CFD) [56], [57], and to provide airflow predictions and estimate aircraft wing turbulence.

2) Algorithms: Hamiltonian simulations [58] is the most well-known FTQC algorithm for simulating quantum-mechanical systems [59], [60]. Hamiltonian simulations utilize further quantum subroutines, e.g., quantum phase estimation [39] for computing eigenvalues. There also exist NISQ approaches, e. g., the variational quantum eigensolver (VQE) [61], and Quantum Monte Carlo (QMC) [62], [63]. While VQE utilizes a parameterized state with a classical optimizer to estimate the ground state of a Hamiltonian, QMC estimates the property of a quantum system using classical Monte Carlo methods and assesses the overlap between two quantum states on QPUs.

In the case of PDE-based numerical simulations, HHL [38] can be used to solve linear systems of equations, and QPE [39] for estimating the eigenvalues of a matrix. There also exist NISQ algorithms, such as DQC [32] and VQLS [43].

D. Discussion

Quantum algorithms can benefit nearly all HPC applications. Many applications will likely emerge as incrementally quantum-enabled hardware and algorithms mature. While the initial focus is on optimizing essential quantum algorithms, increasingly, the integration of these quantum-enhanced components in end-to-end application workflows needs to be considered.

TABLE I
Quantum Applications and Algorithms: COMMON NISQ AND FTQC ALGORITHMIC APPROACHES PROPOSED FOR DIFFERENT APPLICATIONS.

App class / Algo.	Optimization	Machine learning	Quantum simulation	Classical simulation
Fault-tolerant algorithm	Grover [24]	HHL [38], Distance estimation	Hamiltonian simulation	QPE [39] HHL [38]
NISQ algorithm	QAOA [10]	QNN,QGAN QCBM	VQE [61]	VQLS [43] DQC [32]

Table I summarizes the discussed quantum algorithms in optimization, machine learning, and simulation. FTQC algorithms require many logical qubits, while the current capabilities of existing QPUs limit NISQ algorithms. These typically possess only a few noisy qubits with limited coherence times.

All quantum algorithms will be hybrid [64], i. e., a significant computation part is done on classical resources. In particular, for NISQ only critical kernels that provide decisive quantum advantages will be run on a QPU. We expect these quantum kernels to be highly algorithm- and hardware-dependent (e. g., qubit modality, simulator, interconnect).

Increasingly, algorithms from all three domains are used together, e. g., simulation output data is used for machine learning or as input for prescriptive optimization. Further, ML-generated data frequently serves as input to simulations.

III. Integration Patterns

While quantum computers can encode any function that a classical computer can, running complete workflows on quantum computers will soon not be feasible due to the high depth and qubit count required. Thus, quantum applications will need to contend with hybrid resources. Only a minimal kernel, providing a quantum advantage, will often be executed on a standalone QPU. These kernels will be augmented with significant classical components (both for NISQ and FTQC). Hence, a better understanding of the interaction between classical and quantum components requires investigating their integration patterns and analyzing the types of coupling and the application structure.

We identify three types of integration between classical and quantum tasks: *HPC-for-Quantum*, *Quantum-in-HPC* and *Quantum-about-HPC*. We investigate two main characteristics: the coupling [65] and the application structure.

Coupling: Coupling describes the time sensitivity of the interaction between components. The coupling of tasks can occur tightly within the coherence time of the QPU, i. e., the time that a QPU can maintain its state, in near time, for example, to post-process measurements (i. e., medium), and in end-to-end application workflows (i. e., loose) [65].

We refer to *tight-coupling* if tasks need to interact within strict time-sensitive bounds, e. g., within the QPU coherence window. Examples of tight-coupling are quantum error mitigation, error correction, and algorithms that utilize mid-circuit measurements.

Medium-coupled scenarios comprise tasks that require frequent, time-sensitive interaction, but coupling between QPU at coherence time is not needed. The coupling of quantum and classical tasks happens outside the coherence time of the quantum computer. Examples are variational algorithms, such as VQE and QAOA, which process the measurements after each execution of the circuit.

In *loosely-coupled* scenarios, less frequent interaction is needed, e. g., the results are processed together after the parallel job. Loose coupling refers to a coupling on the workflow layer that integrates seamlessly across quantum, classical, and hybrid components of tasks and their dependencies.

The *application structure* describes how the application exploits various types of parallelism, e. g., ensemble, task parallelism, data parallelism, and accelerators. Quantum programs can expose different types of parallelism, both in the classical and quantum parts. For example, quantum algorithms typically involve repeated sampling from circuits, i. e., the circuit must be repeatedly executed and, thus, are amenable to parallelism. Circuit knitting allows the partitioning and parallel execution of circuit parts on multiple QPUs.

A. HPC-for-Quantum

The HPC-for-Quantum integration pattern describes the usage of classic compute and HPC techniques on the low-level system layer to accommodate I/O, dynamic circuits, error mitigation, and other techniques that enable the most effective utilization of the QPU. The layer primarily concerns low-level circuit developers that ensure the execution of quantum circuits on the QPU (Ref. [5]). HPC and quantum tasks are tightly connected and interact in real-time. Approaches are mostly application-agnostic, e. g., error correction, compilation, and parallelism. HPC techniques, e. g., parallelization and acceleration using GPUs and FPGAs and high-performance networking, can provide significant advantages to the quantum kernel and application.

Table II summarizes scenarios for the HPC-for-Quantum integration type. HPC support is crucial to support quantum control, error mitigation, and error corrections, as well as dynamic circuits that require tight coupling of classic and quantum tasks.

HPC technologies are increasingly essential for quantum control systems and enable the optimal manipulation of the qubits through physical operations [69]. For example, determining the optimal timing and method of sending pulses to control the qubits is computationally expensive. Scalable approaches have been proposed, e. g., Quandary [70] uses MPI to distribute necessary computations.

Further, many quantum error mitigation and correction aspects are computationally intensive and require tight integration [69]. For example, the error mitigation of quantum circuits can be performed by running multiple noisy experiments so that errors cancel out [5].

Dynamic circuits involve both the evolution of the quantum state and mid-circuit measurements. The measurements must simultaneously be processed classically (i. e., within coherence

TABLE II
HPC-for-Quantum Application Scenarios categorized by coupling and application structure, focusing on the different classical and quantum tasks. Significant classical computation is required to support a quantum computer.

Scenario	Description	Coupling	Category	Application structure	Classical Task	Quantum Task
Quantum control	Controlling adiabatic/ diabatic schedule	tight	NISQ	Accelerators	Bayesian Optimization/ Reinforcement Learning	All QCs
Error-mitigation & correction [66]	Embedded into algorithm with repeated measurements & application corrections	tight	NISQ/ FTQC	Accelerators	Surface codes with significant classical processing	All QCs
Dynamic circuits	Circuits that are conditioned on the input of real-time classical components	tight	NISQ	Accelerators	classical processing of auxiliary qubit with feedback into circuit	All QCs
Circuit knitting [67]	Decomposing large quantum circuits into smaller circuits for distribution across QPUs	tight	NISQ	Task Parallelism	Circuit decomposition & result collection	All QCs
Classic simulation [68]	Using HPC resources, methods to simulate quantum computers	-	Classical	Ensemble, Task Parallelism, Accelerators	Statevector, tensor networks, density matrix simulation	n/a

TABLE III
Quantum-in-HPC Application Scenarios: FTQC and NISQ algorithms require coupling with classical tasks for pre-, post- and optimization tasks.

Scenario	Description	Coupling	Category	Application structure	Classical Task	Quantum Task
Hamiltonian Simulation [58]	Time evolution of Schrödinger's equation.	-	FTQC	-	-	Hamiltonian simulation
Quantum Phase Estimation (QPE) [39]	Subroutine to extract eigenvalue and eigenstates from a Hamiltonian.	medium	FTQC /FTQC	Task Parallelism	Different variants of QPE with different levels of classical processing interweaved [71]	QPE circuit
Imaginary Time Evolution [73]	Variational algorithm utilizing trial quantum state classic linear solver	medium	NISQ	Task Accelerator	linear equations solving	Trial state
Variational Algorithms [2]	Quantum kernels with classical optimizers (VQE, QAOA, QNN)	medium	NISQ	Task Parallelism	Optimization loop, warm-starting, pre-/post-processing, RQAOA elimination [23]	Parameterized circuit (ground state estimation, QAOA circuit)
Generative AI [48]	QCMB, QGANs (quantum generator with classic or quantum discriminator)	medium	NISQ	Task Parallelism, Accelerators	Optimizer, Discriminator module	Copula circuit, strongly entangled circuit

time) and are used to steer further quantum processing, e. g., by branching or setting variables. Corcoles et al. [71] demonstrated algorithm improvements, e. g., in QPE.

Classic simulations of quantum systems are also an essential building block and critical for evaluating quantum algorithms and hardware. HPC and AI techniques provide the necessary scale to simulators, e. g., by using parallel and GPU-accelerated simulators (e. g., cuQuantum [68] for multi GPU and node statevector simulation) and task-parallel tensor network simulations [72].

B. Quantum-in-HPC

Quantum-in-HPC is an integration pattern where a quantum component is medium-coupled with a classical HPC component. In contrast to HPC-for-Quantum scenarios, e. g., dynamic circuits, the interweaving of classical and quantum computation does not occur in real-time during the coherence of the QPU, but after each measurement cycle.

Table III summarizes different scenarios. Typically, these scenarios involve an ensemble of quantum tasks for repeated measurements. The orchestration of the application typically resides on the classical HPC system, with the QPU only providing acceleration for the different types of parameterized quantum circuits (PQC).

Examples, where this is already the case, are variational quantum algorithms (VQAs) [2], e. g., VQE for estimating the ground state energy of a molecular system, QAOA for solving combinatorial optimization problems, VQLS [43] for solving linear equations, or quantum GANs for generative AI [48]. The amount of classical computing can vary significantly from a classical optimizer loop to comprehensively pre-computing states, e. g., using warm starting procedures (see Quantum-about-HPC integration type).

Different types of parallelisms must be managed by the middleware. For example, ensembles, i. e., multiple independent tasks that are executed on a QPU, and more general task parallelism, where complex task dependencies must be handled. For example, VQAs exhibit more complex task parallelism, where each generation of quantum tasks depends on the results of the previous generation.

C. Quantum-about-HPC

The *Quantum-about-HPC* integration pattern describes scenarios where a quantum-enhanced kernel is integrated into an end-to-end quantum-classical workflow. In other words,

TABLE IV

Quantum-about-HPC Application Scenarios enable the integration of quantum application components into end-to-end workflows.

Scenario	Description	Coupling	Category	Application structure	Classical Task	Quantum Task
Classic preprocessing	Encode classical data into a quantum state, e. g., in QML and Quantum Chemistry	loose	NISQ	Workflow	Data embedding for ML [27], molecular Hamiltonian and initial parameters using Hartree-Fock method [59]	Quantum-in-HPC application component (Table III)
Classic post-processing	Partial extraction of quantum computation results via QPU measurements. Classic post-processing for reconstructing state for further processing.	loose	NISQ /FTQC	Workflow	Parallel processing of expectation values from observable computed on a QPU	Same as above
Hyperparameter optimization [74]	Select optimal parameters for quantum kernel (e. g., cost function, learning rate, initialization)	loose	NISQ	Ensemble	Parameter selection & post-processing	Same as above
AI workflows	QML to learn complex states as input for simulation and property prediction (chemistry [75] and optimization [76])	loose	NISQ	Workflow	Classic simulation loop, pre/post-processing optimization	Same as above
Warm starting	Warm-starting of quantum algorithm with classical solution [22]	loose	NISQ	Workflow	Heuristics (MILP, CPLEX) to pre-compute initial parameters	Same as above
Hybrid Quantum Monte Carlo [63]	Complex part (e. g., sign problem) on QPU (overlap between sample and trial wave function), executing other parts classically (time evolution)	medium	NISQ	Task Parallelism	Sample generation, time evolution	Overlap estimation between trial and sample wave function
Quantum-Quantum coupling	Coupling Hamiltonian simulation and analysis of static properties results	tight	FTQC	Workflow	Optimizer, Discriminator module	Hamiltonian simulation

the quantum component is added without modifying the HPC application – unlike in the Quantum-in-HPC integration pattern. In this case, the quantum component is used as a blackbox but requires further input or output to be effective. The main application control loop generally resides in the classical system. On this level, the quantum and classical components are often looser coupled than with other integration types.

Table IV summarizes different application scenarios for the Quantum-about-HPC integration type. For example, pre- and post-processing tasks commonly need to be performed, e. g., for data encoding, loading and converting data, and pre-conditioning quantum algorithms. An example of pre-conditioning is warm-starting QAOA, for which a classic solution is used to determine the initial parameters. In other cases, the quantum results are inputs for further classic or quantum processing. For instance, the output of quantum generative methods (e. g., QGAN, QCBMs) serves as input for further optimization and numerical simulations (e. g., in quantum chemistry and high-energy physics [30]).

However, such workflows can exhibit more complex integration patterns, e. g., the in-situ processing of quantum data with quantum machine learning (ML), the training of ML surrogate models to mitigate the data readout bottleneck, and the coupling of generative Quantum ML with other techniques, e. g., simulation and optimization (both quantum and classical).

D. Discussion

Figure 1 summarizes application patterns for the different Quantum-HPC integration types. The HPC-for-Quantum scenarios describe the tight integration of HPC and quantum resources, often in real-time, i. e., within the coherence time of the quantum system. It is characterized by frequent data exchange between the classical and quantum systems to process mid-circuit measurements, e. g., for error correction.

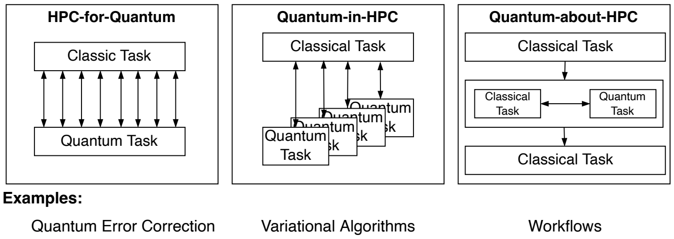

Fig. 1. **Quantum-HPC Integration Patterns:** HPC-for-Quantum requires interactions within the coherence time of the QPU, Quantum-in-HPC utilizes a classical task to orchestrate short-running quantum tasks, Quantum-about-HPC connects composable tasks to workflows.

Quantum-in-HPC describes interactions where the QPU is integrated as an accelerator for specific task types, e. g., for the evaluation of a quantum state. The classical task is long-lived, maintaining the overall state and utilizing the QPU for specific quantum tasks. These are short-lived compared to the classical task. As described in Table III, variational algorithms have been proposed for nearly all problem domains, e. g., for machine learning, optimization, and linear algebra. Finally, Quantum-about-HPC workflows integrate heterogeneous tasks, i. e., quantum, classical or composable tasks, into end-to-end application scenarios. Workflows comprise distinct stages, e. g., data collection, pre-/post-processing, and simulation.

Managing quantum and classical resources can be difficult due to the varying and unpredictable resource demands, requiring a sophisticated approach to resource management. For example, the QPU resource demands for variational circuits can vary significantly as using different optimizers, e. g., can result in a different number of circuit executions. Gradient-based optimizers require more executions of a quantum circuit to estimate the gradient using the parameter shift rule than non-gradient-based optimizers. Thus, a middleware system that can adaptively manage the resources is required. With scale, data and computational requirements will become even more demanding, exacerbating the need for careful resource management.

IV. State of the Art and Related Work

In this section, we describe the current ecosystem of quantum software frameworks, Quantum-HPC integration and middleware systems. Particularly, we investigate how these systems address the challenges related to integration patterns, as described in Section III.

A. *Quantum Software Libraries*

Various quantum software frameworks emerged, e. g., Pennylane [77], Qiskit [78], Cirq [79], Intel Quantum SDK [80], [81], Quil [82] and Quantum Brilliance SDK [83]. Here we summarize the key aspects and limitations of these quantum software frameworks. For a detailed survey, refer to Serrano et al. [84].

The existing frameworks support creating and executing quantum circuits on multiple quantum backends (e. g., simulators and real quantum devices) and enable interfacing with various hardware platforms (e. g., superconducting and ion trap platforms). Further, several high-level libraries that provide ready-to-use quantum-based algorithms for optimization, machine learning, and simulation have been developed. For QML applications, utilizing a gradient-based optimizer with quantum circuits is critical. For example, Pennylane supports differentiable quantum circuits by integrating with machine learning frameworks like PyTorch, Jax, and Tensorflow.

Most frameworks also provide some parallelization and accelerator support, e. g., for just-in-time compilation and accelerated GPU simulators. However, they are often limited to specific cloud platforms and must better interface with HPC resource managers. As workload and task management are deeply integrated into these frameworks, the degree of integration with HPC systems and, thus, the scale is limited.

B. *Quantum-HPC Integration*

Current trends show a transition from remote cloud access for QPUs to a more tightly integrated HPC model, wherein the QPU is co-located alongside classical HPC computing resources [3]–[6].

Minimizing the latency and bandwidth is critical for HPC-for-Quantum and HPC-in-Quantum use cases. Ella et al. [69] investigate low-level HPC-for-Quantum integration on pulse-level required for quantum control, error correction, and mitigation. They emphasize the need for HPC classical resources, including accelerators, to be co-located with the QPU.

Another vital concern, particularly for HPC-in-Quantum use cases, is resource management. While existing resource management systems, e. g., SLURM, can support QPUs (e. g., using SLURM's Generic Resource abstraction), but there are significant limitations, e. g., regarding support for different QPU types and the prioritization of QPU jobs. Further, application-level workload and task management systems, such as Pilot-Jobs [85] and Hyperqueue [86], must be integrated with quantum software frameworks. Ruefenach et al. [6] summarizes many of these challenges, e. g., ensuring the optimal utilization of the QPUs, while minimizing the time-to-solution and energy-to-solution, and propose a quantum resource manager.

C. *Quantum-HPC Middleware*

Integrating quantum and classical tasks and quantum-HPC middleware systems are subject to intense research. For example, XACC [87] introduces a quantum-classical programming model that allows tighter integration between both computing paradigms. CUDA Quantum [88] is a platform for integrating classical and quantum computing devices using a common programming model similar to XACC. It offers optimized control and communication between different quantum processors and classical tasks. It integrates with the cuQuantum GPU library for accelerated simulations with scaling across distributed multi-node and multi-GPU systems. Further, support for selected QPUs (e. g, Rigetti) will be available.

Increasingly, hybrid quantum-classical runtimes are integrated into existing quantum software frameworks. For example, Qiskit Runtime [89] and Braket Jobs [90] provide mechanisms to manage classical computing with quantum tasks more efficiently. These are limited to proprietary cloud environments.

Quantum workflows have become critical in addressing the need to integrate quantum components into end-to-end applications (Quantum-about-HPC integration type). Weder et al. [91], [92] investigate workflow technology for orchestrating quantum applications. *Tierkreis [93]* focuses on task parallelism exhibited by hybrid quantum-classical applications and utilizes a dataflow-based programming model. Other commercial tools emerged, e. g., Orquestra [94] and Covalent [95].

V. Quantum-HPC Middleware: Toward a Conceptual Framework

In this section, we define the functional layers for a Quantum-HPC middleware, identify challenges and design objectives for each layer, and describe a conceptual middleware.

A. *Functional Layers*

We adopt the four-layered model for managing scientific workflows on HPC resources proposed by Turilli et al. [96]. Figure 2 illustrates the functional layers: the resource (L1), task (L2), workload (L3), and workflow (L4) layers. Further, it shows different quantum software libraries alongside appropriate layers.

The highest layer is the workflow layer (L4) which encapsulates the application semantics and logical dependencies between the different workflow levels. It is the most abstract

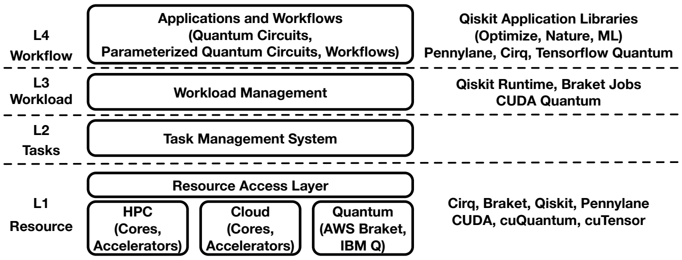

Fig. 2. **Functional Levels for Quantum-HPC Middleware (adapted from [96]):** The middleware system can be partitioned into four layers. The workflow layer encapsulates the application semantics and logical dependencies between the different workflow stages. The workload layer translates the workflow into a set of tasks that can be executed, potentially concurrently. The task layer executes these tasks using the resource layer.

layer and is often exposed using a domain-specific language (DSL) to describe the workflow. The workflow manager translates the workload description into a workload, i. e., a set of tasks that can be executed, potentially concurrently.

The workload and task layers are the middle Layers (L2, L3). The workload layer (L3) selects the appropriate resources for the given workload. The task layer (L2) executes these tasks on the selected resource. For this purpose, it includes functionality to acquire respective HPC resources (e. g., a Pilot-Job [85]). For quantum computing, the co-allocation of quantum and classical resources is critical.

The resource layer (L1) is responsible for scheduling and assigning computational tasks to the various resources within the HPC system, such as nodes, processors, and QPUs. In particular, in the context of quantum computing, the heterogeneity at this layer is challenging. Advances in the intermediate representation, e. g., the Quantum Intermediate Representation (QIR) [97] and Open Quantum Assembly Language (QASM) [98], and unified access APIs are critical to ensure unified access to heterogeneous hardware.

On the resource level (L1), the focus is on executing quantum tasks (i. e., quantum circuits) and related classical tasks on quantum/classical resources. The execution of quantum circuits involves compilation, error mitigation/correction, measurements, and other low-level optimization steps. Further, repeated measurements to obtain a representative sample of the quantum state and post-processing (e. g., to compute expectation values) are managed on this layer.

B. Challenges and Design Objective

This section identifies challenges and design objectives (O) at each functional layer that can lead to high-level Quantum-HPC middleware architecture.

*L4 – Workflow Layer (***O-1***):* Applications require integrating a diverse set of classical (e. g., classical AI and HPC tasks) and quantum tasks (e. g., simulated quantum tasks) in the end-to-end workflows. Achieving such objectives requires modular and composable architectural designs to enable reuse at different levels, e. g., the function, library, and system level. Additionally, the ability to integrate diverse quantum software libraries and components is important [96]. Quantum workflows possess additional complexity, e. g., they require incorporating different types of QPUs (simulated, ion-traps, superconducting). Often, applications involve, e. g., simulated and different physical QPUs, and, thus, require special adaptations while the application logic remains the same. The high-level workflow description is then converted into a workload comprising heterogeneous tasks that need to be mapped to a complex infrastructure of nodes, CPUs, GPUs and QPUs. In particular, in the current (early) stage, the software ecosystem is highly fragmented and standards for describing and executing quantum workflows are missing.

*L3, L2 – Workload and Task Layer (***O-2***):* Workloads consist of highly heterogeneous containing components and tasks implemented in different languages and frameworks as described previously. For instance, quantum machine learning requires the integration of quantum frameworks, like Pennylane, with machine learning frameworks, such as PyTorch and Jax. Application resource requirements can vary significantly with specific configurations (e. g., the QPU type or optimizer in a VQA). Often the same workload must be executed at different scales and on other resource types (e. g., classical simulators and QPU)s, leading to significantly different execution characteristics on the workload and task layer. For example, the execution time and results of the same quantum tasks can vary considerably with the QPU type (simulated vs. physical QPU of different modalities) [99]. A challenge is to identify emerging workload patterns that need to be supported by the workload management system.

*L1 – Resource Layer (***O-3***):* The resource layer encapsulates the heterogeneous quantum and classical resources. Challenges arise concerning integrating quantum resources and supporting the tight coupling of quantum and classical tasks, e. g., for quantum error correction and dynamic circuits. Tight coupling requires the co-allocation of resources to ensure frequent and low-latency interactions between quantum and classical tasks. A tighter integration at the hardware and network level is desirable, particularly for HPC-for-Quantum scenarios. While in the traditional accelerator model, GPUs are dedicated to a single application, the scarcity of physical QPUs requires more careful consideration of resource allocation.

C. Conceptual Middleware

Figure 3 illustrates the conceptual Quantum-HPC middleware with four layers, namely, workflow, workload, task, and resource layers, based on the functional levels identified in Figure 2.

1) Workflow Layer: The workflow layer offers a high-level abstraction for quantum workflows, containing both quantum and classical components. It receives high-level descriptions of dependencies, input/output data, and computational tasks. The workflow manager coordinates workflow descriptions and prepares their execution.

We identify three task types: classical, quantum, and composite tasks. Classical tasks are self-contained classical computations, such as data loading, pre-processing, and post-processing. Quantum tasks are self-contained quantum cir-

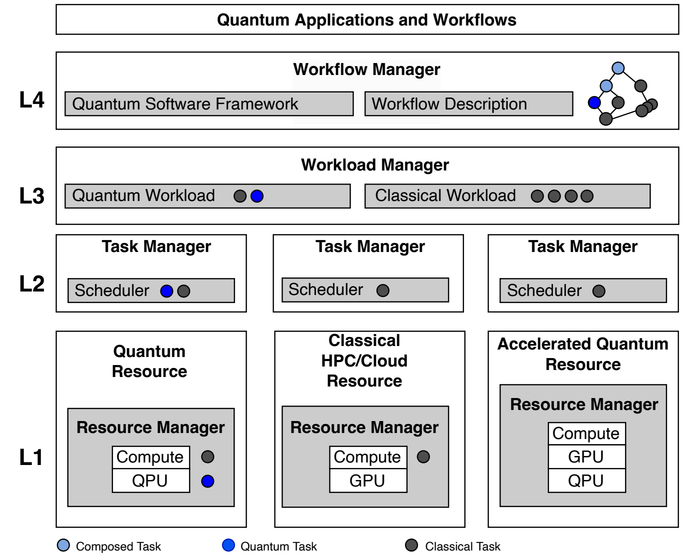

Fig. 3. **Conceptual Quantum-HPC Middleware:** The Quantum-HPC middleware is composed of four layers: workflow, workload, task, and resource layers. By decoupling application, workload, task, and resource management concerns, the middleware enables the necessary scale for next-generation Quantum-HPC systems.

cuits, executable on a QPU. Quantum workloads contain both classical and quantum tasks.

A quantum task refers to self-contained quantum computation, also called a circuit, that is executable on a QPU. A quantum task can be defined using a low-level language (e. g., OpenQASM) or a quantum software framework (e. g., Qiskit). Quantum tasks are typically coupled to different classical computational tasks, e. g., for post-processing, quantum error correction, and hybrid algorithms.

Composite tasks consist of multiple sub-tasks, often integrating external software frameworks, which is crucial for quantum workflows. For example, a composite task can arise by integrating an external software framework, an essential requirement for quantum workflows. For example, Qiskit provides several application frameworks for optimization, simulation, and machine learning. Using composite tasks, a Qiskit QAOA implementation can be integrated into a workflow with further pre- and post-processing steps.

The workflow manager transforms the workflow into an executable state by resolving dependencies. A workload is a set of interdependent tasks that can be executed across different computing resources. The workflow manager can also apply certain optimization, such as the utilization of parallelism for certain parts of the workflow. An example of multi-QPU parallelism is circuit knitting, which decomposes complex quantum circuits into smaller circuits [5]. Another example of task parallelism is ensemble parallelism, comprising loosely coupled tasks with minimal dependencies, e. g., found in parameter sweeps to evaluate a circuit with different parameters and when conducting parallel measurements.

2) Workload Layer: This workload layer orchestrates the execution of the tasks emitted by the workflow layer. The workload manager is the core entity of this layer and is responsible for selecting resources, partitioning the workload, and assigning tasks to resources [96]. We distinguish between classical and quantum workloads. A classical workload only comprises classical tasks, while quantum workloads are hybrid, containing both classical and quantum tasks. Quantum workloads can be highly heterogeneous and hardware-dependent. For example, simulated, ion-trap, and superconducting QPUs have different runtime and fidelity trade-offs.

Quantum-HPC systems face unique workload management and scheduling challenges, including (i) the lack of a unified standard for accessing QPUs and expressing hybrid workloads, (ii) the complex dependencies of applications to specific Quantum Processing Units (QPUs) that often necessitate manual, application-level adaptations, and (iii) the limited availability of physical QPUs that complicates the balancing between application-level and system-level objectives. Our conceptual middleware addresses these challenges, by encapsulating the workload management and scheduling concerns while allowing for effective information flows between application and Quantum-HPC systems.

The workload manager acquires the respective resources via the task layer. Considering the coupling between the quantum and classical tasks, it assigns and co-locates tasks to the resources. For example, classical tasks tightly coupled to quantum tasks, e. g., for the HPC-for-Quantum scenarios, must be co-allocated and assigned to resources in close proximity.

The scheduling of tasks requires both application- and system-level information [100]. Thus, application-level schedulers and Pilot-Jobs [85], [101] may be crucial in bringing together application-level and system-level information. The assigning of tasks to resources is also referred to as binding. The binding of tasks can occur both early and late. Early binding directly assigns tasks to resources based on currently available information. Late binding allows for more dynamism and addresses, e. g, uncertainties like resource fluctuations and other variations in the infrastructure.

3) Task Layer: The task layer is an integral component of the middleware system and comprises a collection of task managers. The task manager orchestrates the execution of tasks on a specific resource as assigned by the workload manager. A task manager is typically responsible for a single resource and manages the resource allocation, acquisition, scheduling, assignment, and monitoring to ensure the tasks run successfully. Typically, tasks are executed on HPC resources as part of a job or within a container on cloud resources. As described, a common mechanism to manage tasks across heterogeneous resources are Pilot-Jobs.

The task manager also supports dynamic allocation during runtime, allowing for the acquisition and release of resources as needed. Additionally, the task manager is responsible for handling errors and failures that may arise during the execution of tasks. It ensures that the necessary resource requirements for successful task execution are met. In the context of quantum computing, resource co-location is crucial. For tightly coupled tasks, the QPU must be co-located with sufficient classical computing capacities, e. g., GPUs and CPU.

For example, quantum error correction requires the co-location of QPU, GPU, and CPU due to the tight coupling between classical and quantum computing tasks. Variational algorithms also benefit from nearby classical computing resources, but the coupling does not occur within the coherence time of the QPU.

4) Resource Layer: This layer represents the diverse HPC and cloud resources, such as classical computing (e. g.,CPUs), QPUs, and accelerators (e. g., GPUs). While QPUs have been located remotely from the classical computation, increasingly tighter integrations of classical resources are emerging [5], [6]. For error correction, accelerated classical computing is required to perform the classical processing of the syndrome measurements. The increased complexity of the resource layer demands abstractions so that the resource layer is consistently presented.

Parallelization of quantum workloads across nodes, cores, and accelerators (including QPUs) is critical to achieving the necessary performance and scale. For this purpose, the resource layer must integrate with the underlying HPC technologies, including QPU-specific compilers, GPU libraries (such as cuQuantum and cuTensor), and networks.

D. Example: Quantum Chemistry Workflow

In the following, we consider a quantum chemistry application where the task is to compute the ground state energy of a molecule, which is frequently used to predict the chemical properties of a molecule [59].

Workflow Layer: The Variational Quantum Eigensolver (VQE) is a NISQ algorithm for computing the ground state energy of a molecule. The ground state estimation is often part of an end-to-end workflow, which includes several pre- and post-computing steps (*Quantum-about-HPC* integration type). Examples of preparation steps are, e. g., reading the molecule data from a file and computing an approximate solution to the ground state using the Hartree-Fock (HF) method.

The VQE algorithm itself is an example of the Quantum-in-HPC integration pattern. The quantum part is a parameterized quantum circuit, which executes on a QPU and estimates the ground state energy. The classical component optimizes the parameters using a classical optimization algorithm. Further, enhancements emerged, e. g., embedding techniques that restrict the simulated quantum particles by utilizing classical simulation techniques. While this reduces the required qubits, it requires additional classical resources [102]. Generally, this algorithm and the Quantum-in-HPC pattern require frequent communication between classical and quantum components as they iteratively update the parameters of the quantum circuit.

The application creates a high-level workflow description that interweaves quantum tasks (specifically a quantum circuit representing the molecular Hamiltonian) and classical tasks (optimization). This representation serves as the basis for efficiently managing dependencies (e. g., external libraries like Qiskit), inputs, execution, and outputs. The workflow description on this level is abstract, i. e., resource-independent. However, it is configurable to allow users some control (e. g., over resource types).

Current quantum software frameworks (e. g., Qiskit and Pennylane) define workflows on a lower level involving precise implementation steps (e. g., in Python) and concrete resource mapping. While these frameworks provide some extension mechanisms, these are typically highly platform-specific and limited. For example, Qiskit's Provider API and Pennylane's Device API allow for integrating custom backends. Our conceptual middleware decouples the workflow description from the implementation details and thus enables optimization on the middleware level.

The middleware system is responsible for mapping the workflow description to a set of tasks and resources. For this purpose, the workflow manager resolves all dependencies and allocates the necessary resources. The output is a workload, i. e., the resources and tasks ready-to-run, which is forwarded to the workload manager.

Workload and Task Layer: The workload and task managers are essential to enable the scalable execution of quantum workflows and their associated workloads (objective **O-2**). The workload manager schedules and orchestrates the execution of quantum and classical tasks emitted by the workflow manager. For VQE, e. g., sufficient quantum and classical resources need to be allocated to optimize interactions between both components. The task manager is then responsible for the execution of individual tasks, either on the QPU or classical resources.

Depending on the characteristics of the workload, particularly the coupling between quantum and classical tasks, the tasks can be placed accordingly, ensuring performance and scale. For example, the pre- and post-computing steps of the workflow do not necessarily be co-located with the QPU. For current QPU capabilities, remote QPU access is sufficient for VQE. With the increasing scale, a co-allocation of resources is required to enable scalable variational algorithms. Further, a co-allocation is critical for HPC-for-Quantum scenarios, e. g., error mitigation routines.

Resource Layer: The emitted quantum and classical tasks are executed on the resource layer. To allow low-level resource-specific optimization (objective **O-3**), the execution typically involves just-in-time compilation steps to optimize the circuit for defined hardware. To evaluate the quantum state, repeated measurements are critical.

VI. Conclusion

Managing and executing quantum workflows poses significant challenges, necessitating middleware solutions. As quantum applications and algorithms progress, we identified three integration patterns for quantum and HPC applications. The need for HPC techniques, such as task parallelism and GPU acceleration, arises in several parts at several levels, e. g., in the classical simulation of quantum circuits, variational algorithms, and quantum error correction. Thus, scalable and efficient integration of quantum and HPC systems on the middleware level is critical. The conceptual middleware decouples workload and task management from the application software while allowing low-level HPC optimization (e. g., resource-specific compilation steps, resource, and task co-allocation). It utilizes established high-performance computing abstractions for this purpose and enables the seamless integration of quantum computing into HPC systems.

As a future work, we aim to develop a reference implementation of the conceptual middleware. Further, we plan to explore the emulation of workloads and resources for hybrid quantum applications. The emulator development includes tools and techniques to mimic the characteristics of applications and infrastructure, thereby enabling the optimization of resource and task allocations before deploying them on actual quantum systems. Finally, the development of workflow and application-level benchmarks [103] will help to establish an understanding of the impact of quantum computing in end-to-end applications.

Acknowledgment

We thank Pradeep Mantha for proofreading, valuable comments and discussions. The authors generated parts of this text with OpenAI's language-generation models. Upon generation, the authors reviewed, edited, and revised the language.

References

[1] A. Bayerstadler, G. Becquin *et al.*, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, p. 25, 2021.

[2] M. Cerezo, A. Arrasmith *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021. [Online]. Available: https://doi.org/10.1038/s42254-021-00348-9

[3] T. S. Humble, A. McCaskey *et al.*, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, pp. 15–23, 2021.

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

[5] S. Bravyi, O. Dial *et al.*, "The future of quantum computing with superconducting qubits," *Journal of Applied Physics*, vol. 132, no. 16, p. 160902, oct 2022. [Online]. Available: https://doi.org/10.1063%2F5.0082975

[6] M. Ruefenacht, B. G. Taketani *et al.*, "Bringing quantum acceleration to supercomputers," IQM/LRZ Technical Report, https://www.quantum.lrz.de/fileadmin/QIC/Downloads/IQM_HPC-QC-Integration-Whitepaper.pdf, 2022.

[7] O. R. L. C. Facility, "Quantum Computing User Program," 2021. [Online]. Available: https://www.olcf.ornl.gov/olcf-resources/compute-systems/quantum-computing-user-program/

[8] Amazon, "Amazon Braket," 2022. [Online]. Available: https://aws.amazon.com/braket/

[9] A. Das and B. K. Chakrabarti, *Quantum annealing and related optimization methods.* Springer Science & Business Media, 2005, vol. 679.

[10] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[11] J. Choi, S. Oh, and J. Kim, "Quantum approximation for wireless scheduling," *Applied Sciences*, vol. 10, no. 20, p. 7116, 2020.

[12] A. Awasthi, F. Bär *et al.*, "Quantum computing techniques for multi-knapsack problems," 2023.

[13] Y. Cao, J. Romero, and A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 6–1, 2018.

[14] Y. Xiang, D. W. Zhang, and J. Z. Zhang, "Fully quantum mechanical energy optimization for protein–ligand structure," *Journal of computational chemistry*, vol. 25, no. 12, pp. 1431–1437, 2004.

[15] B. Li, P. Hu *et al.*, "Performance analysis and optimization of a cchp-gshp coupling system based on quantum genetic algorithm," *Sustainable Cities and Society*, vol. 46, p. 101408, 2019.

[16] L. Gao, R. Liu *et al.*, "An advanced quantum optimization algorithm for robot path planning," *Journal of Circuits, Systems and Computers*, vol. 29, no. 08, p. 2050122, 2020.

[17] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, oct 2014. [Online]. Available: https://doi.org/10.1080%2F00107514.2014.964942

[18] Z. Bian, F. Chudak *et al.*, "Mapping constrained optimization problems to quantum annealing with application to fault diagnosis," *Frontiers in ICT*, p. 14, 2016.

[19] D. Pastorello and E. Blanzieri, "Quantum annealing learning search for solving qubo problems," *Quantum Information Processing*, vol. 18, no. 10, pp. 1–17, 2019.

[20] B. F. Schiffer, J. Tura, and J. I. Cirac, "Adiabatic spectroscopy and a variational quantum adiabatic algorithm," *PRX Quantum*, vol. 3, no. 2, p. 020347, 2022.

[21] A. Perdomo, C. Truncik *et al.*, "Construction of model hamiltonians for adiabatic quantum computation and its application to finding low-energy conformations of lattice protein models," *Physical Review A*, vol. 78, no. 1, p. 012320, 2008.

[22] D. J. Egger, J. Mareček, and S. Woerner, "Warm-starting quantum optimization," *Quantum*, vol. 5, p. 479, jun 2021. [Online]. Available: https://doi.org/10.22331%2Fq-2021-06-17-479

[23] S. Bravyi, A. Kliesch *et al.*, "Obstacles to variational quantum optimization from symmetry protection," *Phys. Rev. Lett.*, vol. 125, p. 260505, Dec 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.125.260505

[24] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[25] A. Gilliam, M. Pistoia, and C. Gonciulea, "Optimizing quantum search using a generalized version of grover's algorithm," *arXiv preprint arXiv:2005.06468*, 2020.

[26] A. Dawid, J. Arnold *et al.*, "Modern applications of machine learning in quantum sciences," 2022. [Online]. Available: https://arxiv.org/abs/2204.04198

[27] M. Schuld and F. Petruccione, *Machine Learning with Quantum Computers*, ser. Quantum Science and Technology. Springer International Publishing, 2021. [Online]. Available: https://books.google.de/books?id=-N5IEAAAQBAJ

[28] M. Weinstein, F. Meirer *et al.*, "Analyzing big data with dynamic quantum clustering," *arXiv preprint arXiv:1310.2700*, 2013.

[29] C. Bravo-Prieto, J. Baglio *et al.*, "Style-based quantum generative adversarial networks for Monte Carlo events," *Quantum*, vol. 6, p. 777, Aug. 2022. [Online]. Available: https://doi.org/10.22331/q-2022-08-17-777

[30] F. Rehm, S. Vallecorsa *et al.*, "Quantum Machine Learning for HEP Detector Simulations," pp. 363–368, 2021. [Online]. Available: https://cds.cern.ch/record/2824092

[31] P. Jain and S. Ganguly, "Hybrid quantum generative adversarial networks for molecular simulation and drug discovery," *arXiv preprint arXiv:2212.07826*, 2022.

[32] O. Kyriienko, A. E. Paine, and V. E. Elfving, "Solving nonlinear differential equations with differentiable quantum circuits," *Phys. Rev. A*, vol. 103, p. 052416, May 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.103.052416

[33] X. Liang, M. Li *et al.*, "2^{1296} exponentially complex quantum many-body simulation via scalable deep learning method," *arXiv preprint arXiv:2204.07816*, 2022.

[34] Z. Cai and J. Liu, "Approximating quantum many-body wave functions using artificial neural networks," *Physical Review B*, vol. 97, no. 3, p. 035116, 2018.

[35] X. Zhao, M. Li *et al.*, "Ai for quantum mechanics: High performance quantum many-body simulations via deep learning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.

[36] D. Xu, A. B. Özgüler *et al.*, "Neural network accelerator for quantum control," 2022. [Online]. Available: https://arxiv.org/abs/2208.02645

[37] L. Moro, M. Paris *et al.*, "Quantum compiling by deep reinforcement learning," *COMMUNICATIONS PHYSICS*, vol. 4, no. 1, pp. 1–8, 2021.

[38] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.*, vol. 103, p. 150502, Oct 2009. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.103.150502

[39] A. Kitaev, "Quantum measurements and the abelian stabilizer problem," 1995. [Online]. Available: https://arxiv.org/abs/quant-ph/9511026

[40] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[41] I. Kerenidis, J. Landman *et al.*, "q-means: A quantum algorithm for unsupervised machine learning," 2018. [Online]. Available: https://arxiv.org/abs/1812.03584

[42] S. Johri, S. Debnath *et al.*, "Nearest centroid classification on a trapped ion quantum computer," 2020. [Online]. Available: https://arxiv.org/abs/2012.04145

[43] C. Bravo-Prieto, R. LaRose *et al.*, "Variational quantum linear solver," 2019. [Online]. Available: https://arxiv.org/abs/1909.05820

[44] A. Abbas, D. Sutter *et al.*, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.

[45] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," *Nature Physics*, vol. 15, no. 12, pp. 1273–1278, 2019. [Online]. Available: https://doi.org/10.1038/s41567-019-0648-8

[46] M. Schuld, "Supervised quantum machine learning models are kernel methods," 2021. [Online]. Available: https://arxiv.org/abs/2101.11020

[47] S. Cheng, J. Chen, and L. Wang, "Information perspective to probabilistic modeling: Boltzmann machines versus born machines," *Entropy*, vol. 20, no. 8, 2018. [Online]. Available: https://www.mdpi.com/1099-4300/20/8/583

[48] M. Benedetti, D. Garcia-Pintos *et al.*, "A generative modeling approach for benchmarking and training shallow quantum circuits," *npj Quantum Information*, vol. 5, no. 1, p. 45, 2019. [Online]. Available: https://doi.org/10.1038/s41534-019-0157-8

[49] P.-L. Dallaire-Demers and N. Killoran, "Quantum generative adversarial networks," *Phys. Rev. A*, vol. 98, p. 012324, Jul 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.98.012324

[50] C. A. Riofrío, O. Mitevski *et al.*, "A performance characterization of quantum generative models," *arXiv e-prints*, pp. arXiv–2301, 2023.

[51] G. E. Karniadakis, I. G. Kevrekidis *et al.*, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021. [Online]. Available: https://doi.org/10.1038/s42254-021-00314-5

[52] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, 1982.

[53] W. C. Skamarock, J. B. Klemp, and J. Dudhia, "Prototypes for the wrf (weather research and forecasting) model," in *Preprints, Ninth Conf. Mesoscale Processes, J11–J15, Amer. Meteorol. Soc., Fort Lauderdale, FL*, 2001.

[54] M. Singh, C. Dhara *et al.*, "Quantum artificial intelligence for the science of climate change," in *Artificial Intelligence, Machine Learning and Blockchain in Quantum Satellite, Drone and Network*. CRC Press, 2021, pp. 199–207.

[55] F. Oz, R. K. Vuppala *et al.*, "Solving burgers' equation with quantum computing," *Quantum Information Processing*, vol. 21, pp. 1–13, 2022.

[56] R. Steijl and G. N. Barakos, "Parallel evaluation of quantum algorithms for computational fluid dynamics," *Computers & Fluids*, vol. 173, pp. 22–28, 2018.

[57] S. Jóczik, Z. Zimborás *et al.*, "A cost-efficient approach towards computational fluid dynamics simulations on quantum devices," *Applied Sciences*, vol. 12, no. 6, p. 2873, 2022.

[58] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.273.5278.1073

[59] Y. Cao, J. Romero *et al.*, "Quantum chemistry in the age of quantum computing," *Chemical Reviews*, vol. 119, no. 19, pp. 10 856–10 915, 2019, pMID: 31469277. [Online]. Available: https://doi.org/10.1021/acs.chemrev.8b00803

[60] B. Bauer, S. Bravyi *et al.*, "Quantum algorithms for quantum chemistry and quantum materials science," *Chemical Reviews*, vol. 120, no. 22, pp. 12 685–12 717, 2020, pMID: 33090772. [Online]. Available: https://doi.org/10.1021/acs.chemrev.9b00829

[61] A. Peruzzo, J. McClean *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, p. 4213, 2014. [Online]. Available: https://doi.org/10.1038/ncomms5213

[62] A. Montanaro, "Quantum speedup of monte carlo methods," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 471, no. 2181, p. 20150301, sep 2015. [Online]. Available: https://doi.org/10.1098%2Frspa.2015.0301

[63] W. J. Huggins, B. A. O'Gorman *et al.*, "Unbiasing fermionic quantum monte carlo with a quantum computer," *Nature*, vol. 603, no. 7901, pp. 416–420, 2022. [Online]. Available: https://doi.org/10.1038/s41586-021-04351-z

[64] A. Callison and N. Chancellor, "Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond," *Physical Review A*, vol. 106, no. 1, p. 010101, 2022.

[65] A. McCaskey, E. Dumitrescu *et al.*, "A language and hardware independent approach to quantum–classical computing," *SoftwareX*, vol. 7, pp. 245–254, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711018300700

[66] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, "Towards practical classical processing for the surface code," *Phys. Rev. Lett.*, vol. 108, p. 180501, May 2012. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.108.180501

[67] T. Peng, A. W. Harrow *et al.*, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Lett.*, vol. 125, p. 150504, Oct 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.125.150504

[68] L. Fang, ahehn nv *et al.*, "Nvidia/cuquantum: cuquantum python v22.11.0.1," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7523366

[69] L. Ella, L. Leandro *et al.*, "Quantum-classical processing and benchmarking at the pulse-level," 2023.

[70] S. Günther, N. A. Petersson, and J. L. Dubois, "Quandary: An open-source c++ package for high-performance optimal control of open quantum systems," 2021. [Online]. Available: https://arxiv.org/abs/2110.10310

[71] A. D. Córcoles, M. Takita *et al.*, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Phys. Rev. Lett.*, vol. 127, p. 100501, Aug 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.127.100501

[72] T. Vincent, L. J. O'Riordan *et al.*, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, p. 709, may 2022. [Online]. Available: https://doi.org/10.22331%2Fq-2022-05-09-709

[73] S. McArdle, T. Jones *et al.*, "Variational ansatz-based quantum simulation of imaginary time evolution," *npj Quantum Information*, vol. 5, no. 1, p. 75, 2019. [Online]. Available: https://doi.org/10.1038/s41534-019-0187-2

[74] S. S. Cranganore, V. De Maio *et al.*, "Molecular dynamics workflow decomposition for hybrid classic/quantum systems," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 346–356.

[75] N. W. A. Gebauer, M. Gastegger *et al.*, "Inverse design of 3d molecular structures with conditional generative neural networks," *Nature Communications*, vol. 13, no. 1, p. 973, 2022. [Online]. Available: https://doi.org/10.1038/s41467-022-28526-y

[76] J. Alcazar, M. G. Vakili *et al.*, "Geo: Enhancing combinatorial optimization with classical and quantum generative models," 2021. [Online]. Available: https://arxiv.org/abs/2101.06250

[77] V. Bergholm, J. Izaac *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv:1811.04968*, 2018.

[78] A. Cross, "The ibm q experience and qiskit open-source quantum computing software," in *APS March meeting abstracts*, vol. 2018, 2018, pp. L58–003.

[79] A. Hancock, A. Garcia *et al.*, "Cirq: A python framework for creating, editing, and invoking quantum circuits."

[80] P. Khalate, X.-C. Wu *et al.*, "An llvm-based c++ compiler toolchain for variational hybrid quantum-classical algorithms and quantum accelerators," 2022.

[81] G. G. Guerreschi, J. Hogaboam *et al.*, "Intel quantum simulator: a cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, vol. 5, no. 3, p. 034007, may 2020. [Online]. Available: https://doi.org/10.1088%2F2058-9565%2Fab8505

[82] R. S. Smith, "Quil: A portable quantum instruction language," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3677541

[83] T. Nguyen, D. Arya *et al.*, "Software for massively parallel quantum computing," *arXiv preprint arXiv:2211.13355*, 2022.

[84] M. A. Serrano, J. A. Cruz-Lemus *et al.*, "Quantum software components and platforms: Overview and quality assessment," *ACM*

Comput. Surv., vol. 55, no. 8, dec 2022. [Online]. Available: https://doi.org/10.1145/3548679

[85] A. Luckow, M. Santcroos *et al.*, "P*: A model of pilot-abstractions," in *2012 IEEE 8th International Conference on E-Science*, 2012, pp. 1–10.

[86] J. Beránek, A. Böhm *et al.*, "It4innovations/hyperqueue: v0.15.0," Apr. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7838764

[87] A. J. McCaskey, D. I. Lyakh *et al.*, "XACC: a system-level software infrastructure for heterogeneous quantum–classical computing," *Quantum Science and Technology*, vol. 5, no. 2, p. 024002, feb 2020. [Online]. Available: https://doi.org/10.1088%2F2058-9565%2Fab6bf6

[88] NVIDIA, "NVIDIA CUDA Quantum: The platform for hybrid quantum-classical computing," https://developer.nvidia.com/cuda-quantum, 2023.

[89] B. Johnson, "Qiskit runtime, a quantum-classical execution platform for cloud-accessible quantum computers," *Bulletin of the American Physical Society*, 2022.

[90] D. Poccia, "Introducing amazon braket hybrid jobs – set up, monitor, and efficiently run hybrid quantum-classical workloads," https://aws.amazon.com/blogs/aws/introducing-amazon-braket-hybrid-jobs-set-up-monitor-and-efficiently-run-hybrid-quantum-classical-workloads/, 2021.

[91] B. Weder, U. Breitenbücher *et al.*, "Integrating quantum computing into workflow modeling and execution," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 279–291.

[92] B. Weder, J. Barzen *et al.*, *Quantum Software Development Lifecycle*. Cham: Springer International Publishing, 2022, pp. 61–83. [Online]. Available: https://doi.org/10.1007/978-3-031-05324-5_4

[93] S. Sivarajah, L. Heidemann *et al.*, "Tierkreis: A dataflow framework for hybrid quantum-classical computing," 2022. [Online]. Available: https://arxiv.org/abs/2211.02350

[94] Zapata Computing, "Orquestra: A platform for hybrid quantum-classical computing," Zapata Computing, https://www.zapatacomputing.com/orquestra-platform/, 2023.

[95] Covalent, "Covalent: Open source workflow orchestration for heterogenous computing," Covalent, https://www.covalent.xyz/, 2023.

[96] M. Turilli, V. Balasubramanian *et al.*, "Middleware building blocks for workflow systems," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 62–75, jul 2019. [Online]. Available: https://doi.org/10.1109%2Fmcse.2019.2920048

[97] Q. Alliance. (2023) Qir alliance: The core of quantum development. Accessed: May 04, 2023. [Online]. Available: https://www.qir-alliance.org/

[98] (2023) Openqasm. Accessed: May 04, 2023. [Online]. Available: https://openqasm.com/

[99] T. Lubinski, C. Coffrin *et al.*, "Optimization applications as quantum performance benchmarks," 2023.

[100] F. Berman, R. Wolski *et al.*, "Application-level scheduling on distributed heterogeneous networks," in *Supercomputing '96:Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, 1996, pp. 39–39.

[101] M. Turilli, M. Santcroos, and S. Jha, "A comprehensive perspective on pilot-job systems," *ACM Comput. Surv.*, vol. 51, no. 2, apr 2018. [Online]. Available: https://doi.org/10.1145/3177851

[102] M. Rossmannek, P. K. Barkoutsos *et al.*, "Quantum HF/DFT-embedding algorithms for electronic structure calculations: Scaling up to complex molecular systems," *The Journal of Chemical Physics*, vol. 154, no. 11, 03 2021, 114105. [Online]. Available: https://doi.org/10.1063/5.0029536

[103] J. R. Finžgar, P. Ross *et al.*, "Quark: A framework for quantum computing application benchmarking," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2022, pp. 226–237.