EDITORS: Kathryn Mohror, mohror1@llnl.gov
John M. Shalf, JShalf@lbl.gov

DEPARTMENT: LEADERSHIP COMPUTING

Accelerating HPC With Quantum Computing: It Is a Software Challenge Too

Martin Schulz, *Technical University of Munich, 80333, Munich, Germany*

Martin Ruefenacht, *Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities, 85748, Garching near Munich, Germany*

Dieter Kranzlmüller, *Ludwig–Maximilians–Universität München, 80538, Munich, Germany*

Laura Brandon Schulz, *Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities, 85748, Garching near Munich, Germany*

With quantum computing (QC) maturing, high-performance computing (HPC) centers are already preparing to host early-phase production versions of such systems. Unlike their experimental predecessors in physics laboratories, with a very small and dedicated user community, this next generation of systems needs to serve a wider user community and must work in concert with existing HPC systems and software stacks. This article describes our vision for an integrated ecosystem that combines existing HPC and evolving quantum software stacks into a single system to enable a common and continuous user experience. This integration comes with several major challenges as quantum systems pose significantly different requirements including increased need for compilation at run time, long optimization times, statistical evaluations of results, and the need to work with few centralized resources. To overcome these challenges, new scheduling approaches on the HPC side and new programming approaches on the QC side are required.

MOTIVATION

Quantum computing (QC), i.e., the idea of using quantum states and transformations to express computation, is taking shape. After decades of experimentation in physics laboratories, many large-scale research efforts in academia, laboratories, and industry world-wide have started to target usable and accessible quantum computing devices. These efforts explore a wide range of underlying technologies from superconducting qubits, spin-qubits, ion traps to neutral atoms, to name just a few.

While these developments are highly promising, it is becoming clear that quantum computing systems will not replace existing compute architectures; they more likely augment them by accelerating certain suitable tasks or kernels. The computation of other kernels and work needed for I/O and workflow management will (at least for the foreseeable future) remain bound to the existing compute approaches. Additionally, quantum computing relies on several compute-intensive tasks, which require support from HPC systems. Consequently, QC must seamlessly become part of HPC, enabling common user access and experience.

In order to enable the needed integration, it will be essential to not only develop QC hardware and physically connect it to HPC systems, which is currently the main focus for several groups with approaches ranging from loosely coupled cloud or modular access,[1,2] to near quantum compute options,[3] to actual deep integration targeting low-latency access.[4] We also need to focus on a continuous software stack that enables a user to harness the combined computational capabilities of both

1521-9615 © 2023 IEEE
Digital Object Identifier 10.1109/MCSE.2022.3221845
Date of current version 4 January 2023.

system components in a seamless fashion. In this article, we describe our efforts toward this combined software stack: we build on novel features for the HPC software side, particularly dynamic adaptivity and malleability across all software components and modern workflow features. On the QC side, we start with the standard programming packages available to QC researchers, such as Cirq[5] and Qiskit,[6] and link them to the HPC resources management as well as augment their back-ends to efficiently and transparently onload compute intensive tasks back to the HPC system, ideally using the HPC workload manager to identify otherwise unused resources.

Combining these two worlds, however, is not trivial, as they work on radically different assumptions. The HPC world is trimmed for low latency and precompilation; on the other hand, QC systems require a much larger runtime component. The latter is caused by compiling at runtime as input arguments are likely to change the intended quantum program (i.e., the generated circuits). Further, the computation needed to compile and map to the final topology of the QC system is substantial.

Also, QC systems only target single-user operations and offer limited support for multiuser operations. Therefore, integration requires a careful redesign of the runtime component to enable efficient overlap of QC executions from multiple users and to mitigate idle times caused by complex preparation tasks.

Our framework addresses these challenges by adding the needed resource isolation coupled with QC operation interleaving and integrates QC execution scheduling with the ability to onload work needed to drive the QC execution back to the HPC system efficiently. Our approach bridges the software stacks from the two worlds, enabling a single system abstraction for the end user while ensuring high system efficiency.

We are implementing our vision as part of a series of European HPC Software Stack projects and with the Munich Quantum Valley (MQV), a large research initiative driving the development of three different quantum computing technologies with a common software stack. The resulting software will ultimately drive the usage of QC technologies at the Leibniz Supercomputing Centre (LRZ) and compute centers world-wide.

THE ANATOMY OF HPC VERSUS QC SOFTWARE STACKS

Seamless integration needs to support both the HPC users requiring a traditional HPC software stack and the QC users accustomed to the existing stand-alone QC development environments, as well as emerging user groups targeting hybrid HPCQC operations with direct access to QC systems from within HPC systems. Consequently, we need to, at a minimum, support the existing HPC and QC software stacks while also adding support for integrated usage.

A key challenge comes from existing software stacks; the two types of systems are radically different in structure, approach, and user interfaces. This disparity is rooted in the maturity and current scale of the different system components and the diverging requirements stemming from the fundamentally different technologies. Consequently, simply integrating one stack into the other is neither possible nor desirable. Instead, we will build on state-of-the-art software stacks for HPC and QC, respectively, and work to provide an efficient link between the stacks offering the needed integration. This will ensure a solution that offers the best of both worlds while offering a cohesive, integrated view of the system for the users and developers.

HPC Software Stacks

On the HPC side, we build on top of widely available technology forming the software stacks as they are now available on most HPC systems. Examples for this are HPC enabled stacks, such as OpenHPC,[a] the E4S initiative,[b] or the DEEP-SEA stack developed for the European exascale systems,[c] as they are used in most large-scale HPC centers. These include state-of-the-art components for compilers, runtimes, parallel programming abstractions including MPI[7] and OpenMP,[8] support for accelerators following the evolution of system architectures, as well as a wide range of libraries to support efficient application development.

HPC stacks typically target the optimization of execution time by implementing as much work as possible at compile time (compilation, optimization, job preparation, etc.) and, with that, reducing runtime overheads. The latter is then limited to scheduling operations at the job level and actual execution overheads. For this reason, compiled languages such as C, C++, or Fortran are dominant, while interpreted languages play a minor role, e.g., only for coarse-grained workflow orchestration. Further, HPC software stacks typically offer system access via command-line driven batch execution, enabling users direct and fine-grained control of their execution.

QC Software Stacks

Quantum computing stacks, on the other hand, feature a radically different approach to programming

[a]https://openhpc.community/
[b]https://e4s-project.github.io/
[c]https://www.deep-projects.eu/

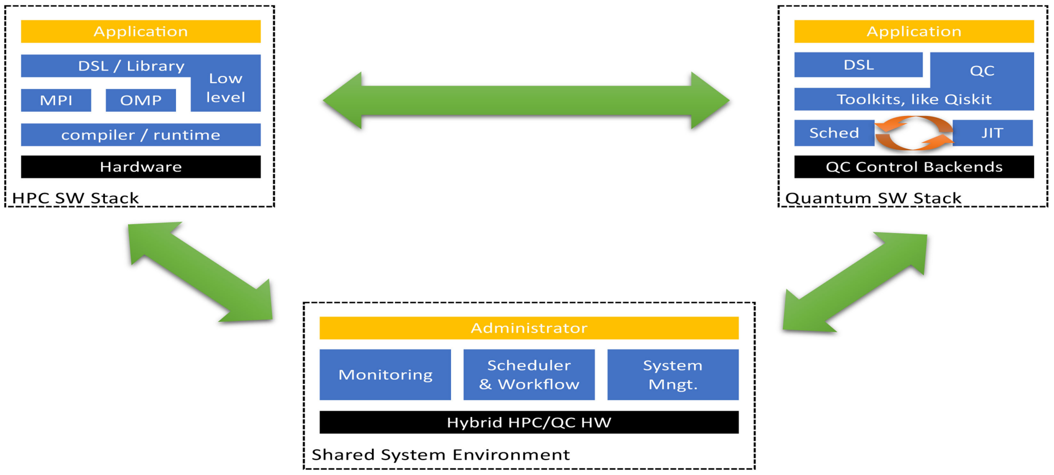

FIGURE 1. Integrating the HPC and QC software environments as well as the overall system environments to reach a seamless hybrid HPCQC system with a combined workflow.

and computing. QC programs, typically in the form of quantum circuits, are defined at runtime using interpreted languages, such as Python, for convenience. Typical programming front-ends are frameworks such as Qiskit[6] or Cirq,[5] which use Python as their base. Programs are then assembled and translated into lower level representations at runtime. The latter is driven by the property of QC programs that changing parameters influences the structure and composition of the program and hence requires new compilations.

This approach works for current scenarios, where individual users access a specific quantum system to execute a particular circuit. As circuits are small, usage is typically interactive and preparation time for a particular problem is ignored as it does not contribute to overall execution time. This, however, changes when users intend to run larger as well as multiple QC applications with only minor parameter changes triggering frequent recompilation and optimization. To maximize system utilization, the resource QC system, which is rare compared to the typical abundance of HPC nodes, needs to be time and space shared; this prohibits the currently used single-user allocation model.

Gaps and Challenges

To combine these two software stacks with their radically different properties, we must include the inherent runtime component of the QC stack and attempt to hide it as much as possible during the execution. This requires both novel scheduling techniques to overlap QC circuit compilation and execution, to overlap executions from multiple users, and novel approaches to reduce the QC circuit generation and optimization times by parallelizing and onloading this work back to the HPC system.

Additionally, it will be important to unify the system software environments, especially with respect to system management, monitoring, and scheduling in order to enable an efficient and stable operation of the joint HPCQC system. This requires new developments to fuse the monitoring environments available in the different systems, an integrated scheduling approach combining the coarse-grained batch scheduling with the fine-grained scheduling of individual QC experiments, and a joint management concept for the involved nodes or subsystems. This will ensure that system administrators can monitor and manage the overall system as one entity and using a single set of consistent policies.

LINKING THE HPC AND QC SOFTWARE STACKS

As discussed in the previous section, we need to build on top of the existing software stacks for both HPC and QC, to ensure continuity for users, but at the same time we also need to provide an integrating bridge. This concept is illustrated in Figure 1, showing the interactions of the two software stacks for HPC and QC, and the connection to the underlying shared system environment. In the following, we will discuss the interactions between the components in more details.

Support for Offloading

The most natural way of integration is to allow HPC systems to push workloads to the QC system, a mechanism we refer to as "offloading" in congruence with offloading mechanisms in other accelerators. This requires the ability to specify quantum computation from within HPC programs, which is challenging due to the different programming approaches. To solve this challenge, we are developing new techniques to 1) enable Python-driven front-ends and integrate them into existing offload approaches such as OpenMP and 2) use noninterpreted approaches directly in C/C++, similar to the XACC approach.[9]

In all cases, the quantum environments require a compilation on the fly, i.e., at runtime, as input parameters will (in most cases) affect the required circuit, as initial conditions need to be directly encoded. Such a compilation can be trivial in the naive case. However, in most cases, it requires intensive computations to achieve the needed optimizations and mapping to the underlying qubit topologies on the quantum devices. This introduces extra latencies, which will block the calling entity. While this is acceptable for individual experiments through direct access from frameworks like Qiskit, it causes substantial delays when used from within HPC jobs. In the latter case, a large number of nodes are held until the translation and optimization is complete, causing unacceptable idle times.

Support for Onloading

In order to combat the challenges identified above, we need to find ways to speed up compilations and optimization of quantum circuits. Using the connected HPC resources is a promising approach but requires 1) the ability to send optimization requests back to the HPC system and 2) novel quantum circuit optimizers, which are parallelized. The latter requires quantum development environments to be implemented on compiled platforms with the necessary parallel structures. We are working on new compilers and optimizers that enable such optimization. Our current targets are implementations in Rust, which combine safe coding practices and efficient threading models for parallelization.

Common Scheduling Mechanisms

A common issue for both onloading versus offloading is efficient scheduling: when to execute which QC computation, how to schedule compilation steps versus executing on the actual QC system, and how to onload QC operational considerations back to the HPC systems. Such scheduling needs to be resource-driven, i.e., the execution from multiple users must be interleaved and compilation of quantum circuits must be pushed back to the HPC system looking for idle resources, e.g., for nodes that actually issued the quantum computation and may be idle until the results return.

CONCLUSIONS

Quantum computing is a highly promising technology that has the potential to accelerate certain computations substantially. However, QC must be integrated into the context of existing HPC ecosystems, and for this we need to carefully consider requirements and changes to the respective software stacks, too.

This article presents our vision for a combined HPC and QC software stack. It builds on existing software stacks from HPC and QC, and extends them to ensure tight integration. We provide extensions to include scheduling components for QC-enabled portions on the HPC side. At the same time, on the QC side, we add programming models and the ability to push operations to the HPC side to speed up operations. For this, we leverage recent developments in the HPC software stack for malleability to integrate the more dynamic nature of QC computations.

The result is an efficient link between the HPC and QC software stacks, allowing them to leverage each other and achieve efficient hybrid execution. Ultimately, this will form the needed bridge between the two worlds and enable efficient HPCQC operations. We are pursuing this direction both as part of several European software stack projects and the MQV with the goal of providing the first truly integrated HPCQC software stack to be deployed in production.

ACKNOWLEDGMENTS

This work was supported in part by the German Federal Ministry for Education and Research under Grant 13N15689 (DAQC), Grant 13N16063 (Q-Exa), Grant 13N16188 (MUNIQC-SC), and Grant 13N16078 (MUNIQC-ATOMS), in part by the German Federal Ministry for Economic Affairs and Climate Action under Grant 01MQ22004 C (QuaST), as well as the Bavarian State Ministry of Science and the Arts as part of Munich Quantum Valley (MQV).

REFERENCES

1. V. Bartsch et al., "QC | HPC: Quantum for HPC," Oct. 2021, doi: 10.5281/zenodo.5555960.
2. D. Binosi et al., "European quantum computing & simulation infrastructure," 2022. [Online]. Available: https://qt.eu//app/uploads/2022/02/20220202_HPC-QCS-JWP-final.pdf
3. M. P. Johansson, E. Krishnasamy, N. Meyer, and C. Piechurski, "Quantum computing–a European perspective," Sep. 2021, doi: 10.5281/zenodo.5547408.
4. M. Ruefenacht et al., "Bringing quantum acceleration to supercomputers," 2022. [Online]. Available: https://meetiqm.com/uploads/documents/IQM_HPC-QC-Integration-Whitepaper.pdf

5. C. Developers, “Cirq,” Apr. 2022, See full list of authors on Github: https://github. com/quantumlib/Cirq/graphs/contributors, doi: 10.5281/zenodo.6599601.
6. M. S. Anis et al., “Qiskit: An open-source framework for quantum computing,” 2019, doi: 10.5281/zenodo.2562111. [Online]. Available: https://zenodo.org/record/2562111#.Y4fNnOzMLsI
7. Message Passing Interface Forum, MPI: A. Message-Passing Interface Standard Version 4.0, Jun. 2021. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf
8. OpenMP Architecture Review Board, “OpenMP application program interface version 5.2,” Nov. 2021. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf
9. A. McCaskey, D. Lyakh, E. Dumitrescu, S. Powers, and T. Humble, “Xacc: A system-level software infrastructure for heterogeneous quantum-classical computing,” *Quantum Sci. Technol.*, vol. 5, no. 2, 2020, Art. no. 024002, doi: 10.1088/2058-9565/ab6bf6.

MARTIN SCHULZ is a full professor of computer architecture and parallel systems in the Department of Computer Engineering, Technical University of Munich, 80333, Munich, Germany, and is on the Board of Directors at the Leibniz Supercomputing Centre, Bavarian Academy of Sciences and Humanities, 85748, Garching, Germany. Contact him at https://www.ce.cit.tum.de/caps/mitarbeiter/martin-schulz/ or schulzm@cit.tum.de.

MARTIN RUEFENACHT is a researcher for HPCQC integration in the Department of Quantum Computing and Technologies, Leibniz Supercomputing Centre, Bavarian Academy of Sciences and Humanities, 85748, Garching, Germany. Contact him at Martin.Ruefenacht@lrz.de.

DIETER KRANZLMÜLLER is the chairman of the Board of Directors, Leibniz Supercomputing Centre, Bavarian Academy of Sciences and Humanities, 85748, Garching, Germany, and a full professor for Communication Systems and System Programming, Institute of Informatics, Ludwig-Maximilians-Universität München, 80538, Munich, Germany. Contact him at https://www.mnm-team.org/~kranzlm/ or dieter.kranzlmueller@lrz.de.

LAURA SCHULZ is the head of strategic development and partnerships as well as the head of the Department of Quantum Computing and Technologies, Leibniz Supercomputing Centre, Bavarian Academy of Sciences and Humanities, 85748, Garching, Germany. Contact her at schulz@lrz.de.