Simulated Execution of Hybrid Quantum Computing Systems

Travis S. Humble[1,2], Ronald J. Sadlier[1,2], and Keith A. Britt[1,2]

[1]Oak Ridge National Laboratory, One Bethel Valley Road, Oak Ridge, Tennessee 37831
[2]Bredesen Center, University of Tennessee, Knoxville, Tennessee, 37996

ABSTRACT

We use modeling and simulation to study the behavior and performance of hybrid quantum computing systems. Our approach is based on a layered design with abstract machine models, which identify the key components and interfaces for quantum processing units, quantum programming models, and hybrid execution models. We use discrete-event simulating to track the dynamical state of hierarchical abstract machine models while executing test programs, and we collect statistics on time and energy consumption to forecast the resources required by quantum processors for future scientific computation.

Keywords: Quantum Computing, Modeling and Simulation, Abstract Machine Models

1. INTRODUCTION

Given the apparent slowdown in Moore's Law and the implication that future performance gains must come from alternative technologies,[1,2] the potential for quantum computing to impact existing applications has never been more significant. As a specific and practical use case, we explore the relevance of quantum processing units (QPUs) to serve as computational accelerators.[3] Our approach is based on the design of computer architectures using the accelerator paradigm, an approach that is likely to see increased usage over the course of the next decade. Future machine designs are anticipated to focus on extreme-scale heterogeneity, in which multiple accelerators may be allocated within a node or socket.[4] The impact of QPUs on computing may therefore be greater if they can be reconciled with these prevailing machine designs. In addition, existing application stacks are complex hierarchies of programming concerns and functional abstractions. The holistic rewriting of high-level application software is not only ineffective but also unlikely. Rather, it is advantageous to maximize the benefits of QPUs by integrating the novelty of these processors into existing work flows through abstractions of the specialized functionality. Akin to how graphical processing units (GPUs) have been integrated to accelerate low-level linear algebra methods, we expect QPUs may provide a similar boost to specific applied mathematical tasks.[5]

An outstanding challenge in realizing the accelerator paradigm with integrated, programmable QPUs is the development of the key system components and interfaces that drive such programming. In this contribution, we explore the design space for quantum programming within hybrid execution models. Our approach develops programming and execution-level models that integrate QPUs alongside conventional computing systems by using a hierarchy of languages and interfaces, including programming frameworks, memory systems, instruction set architectures, and native gate constraints. We then use numerical simulators to test the behavior of these models by simulating the movement of classical instructions and data, the latency and accuracy of QPU response, and the total system power consumption among other metrics of performance. The purpose of this work is to address the requirements that must be met by future QPUs to satisfy performance expectations within the accelerator paradigm.

Contact information: Travis S. Humble (humblets@ornl.gov)

Quantum Information Science, Sensing, and Computation X, edited by Eric Donkor, Michael Hayduk, Proc. of SPIE Vol. 10660, 1066002 · © 2018 SPIE
CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2303824

These results complement ongoing efforts to better understand the how quantum algorithms map into machine architectures as well as efforts to quantify the influence of non-ideal device behaviors, i.e., errors and noise, on application performance. While current demonstrations of application programming are limited by the size and a fidelity of experimental QPUs, modeling and simulation permits extrapolation of performance to much larger scales and more sophisticated use cases. In particular, we focus on the potential for quantum computing to surpass conventional methods of problem solving, whereby the expected resources are much greater than anything currently available. Our primary interest is in applications that impact scientific computing including the domains of materials science, chemistry, high-energy physics, and biology, but the impact on areas in business intelligence and operations research are equally posed for consideration. Of course, the expectations set by modeling and simulation must be validated against experimental observations. The current work defers the latter to a future in which such experiments are feasible.

The paper is organized as follows: after the introduction of Sec. 1, we recount the principles of operations for QPU devices and their interaction with conventional computing systems in Sec. 2. In Sec. 3, we describe the simulation of this model using the Structural Simulation Toolkit (SST) discrete-event simulator and the results from several demonstration programs. We offer final remarks in Sec. 4.

2. HYBRID QUANTUM COMPUTATIONAL MODEL

Computational methods play an important role in describing and understanding natural phenomena. These methods calculate estimates for the structure and dynamics of physical processes including materials, chemical, high-energy particles, and biological systems. The utility of these numerical methods are tied strongly to the available computational hardware, i.e., computers, and leading high-performance computing (HPC) systems are currently large-scale infrastructure investments that offer substantial capability for calculation.[6–8] State of the art HPC systems are currently capable of nearly 10^{17} floating-point operations per seconds (FLOPS) or 100 peta-FLOPS. Within the next 5 years, exa-FLOP (10^{18}) HPC systems are expected to be built for scientific use. However, the path to improving HPC performance beyond the exa-scale era is currently ill-defined due to the limitations anticipated on existing technology development paths as described by Moore's Law. This has raised a pressing need to identify alternative computational approaches, including both algorithms and hardware, that can advance computing beyond the exa-scale era.

A leading candidate for advancing computing beyond the exa-scale era is quantum computing. As first described by Feynman, quantum computing is a model of computation that relies on the principles of quantum mechanics. The quantum model has several distinguishing features that afford new opportunities for computational methods. In particular, quantum mechanics uses the wave function as the most fundamental and complete description of all knowable information. Notably, the wave function is defined as the complex-value probability amplitude of a distribution function. Moreover, the behavior of the wave function is described by Schrodinger's partial differential equation,

$$i\hbar\frac{\partial\Psi(t)}{\partial t} = H(t)\Psi(t) \tag{1}$$

where $\Psi(t)$ is the wave function at time t, $H(t)$ is the Hamiltonian that describes the energy interactions of the system including externally applied fields, and $\hbar$ is Planck's constant divided by 2π. Quantum computing is therefore fundamentally an exercise in quantum dynamical control of the Schrodinger equation by tailoring the Hamiltonian. In particular, Eq. (1) is a boundary value problem in which the desired quantum state $\Psi(t)$ is prepared assuming necessary initial conditions. By contrast, the principles of operation for conventional computing is typically derived from the Turing machine, a discrete, rule-based system for executing sequences of instruction within the principles of classical physics. The Church-Turing thesis provides a theoretical statement on the computability of these abstract machines, and a variety of designs may be used depending on the context undertaken. In particular, it is widely suspected that computability under the Church-Turing thesis is fundamentally changed by adopting an abstract machine model that is based on the principles of quantum mechanics.[9] This fundamental change in computability offers exciting new possibilities for future methods of computation, but also raises concerns regarding the disruptive nature of adopting a new approach.

Our point in contrasting the principles of operation between quantum and conventional computing is that these two models must integrate for the realization of quantum computing technology. At minimum, we expect

that usage of the technology requires the ability to initialize the quantum state into a well-defined value and the ability to readout results from the computation. The actions of initialization and readout demand an integration between conventional, aka classical, logic and quantum logic in order that the methods are human interpretable. Moreover, we expect that programming time-dependent control over the Hamiltonian $H(t)$ will require sustained interaction between a conventional logic controller driving the quantum physical system. Indeed, this is how all current quantum technologies operate. Therefore, in order to develop a hybrid model for a quantum-conventional computing system, we begin by first adopting a hierarchical description of the interfaces currently used for controlling the quantum dynamics. In particular, we model quantum processing units (QPUs) in terms of components that communicated using specified languages and which carry out the necessary control of the quantum physical systems. The purpose of this model is to enable a simulation of how these hybrid computing system execute computational programs.

2.1 Quantum Processing Units

In this section, we summarize the requirements and behaviors expected for large-scale general-purpose QPUs. The basic requirements expected for a quantum computer were first articulated by DiVincenzo,[10] and we further define the general principles of operation for these abstract devices. These requirements follow from the expected basic usage for quantum computing hardware, which may in general be very broad. These criteria represent the minimal behaviors needed to perform general-purpose quantum computing in the presence of likely architectural constraints. Only a subset of these criteria may be required by specializing a QPU to singular function, e.g., a special-purpose device, or by engineering the quantum physical interactions to implement prescribed behaviors. We do not consider that case here.

First among the criteria is the ability to address elements within a scalable register of quantum physical systems. These register elements perform the role of storing information and scalability implies a manufacturing capability to fabricate and layout as many register elements as needed for a specific computational method. Second, these register elements must be capable of being initialized with high fidelity, as the starting quantum state of the computation needs to be well known to ensure accurate results. As described by Eq. (1), this amounts to preparing the quantum physical system(s) in a specific starting state. As a matter of convenience, the leading choice for register element design corresponds with a two-level system, in which orthonormal levels labeled 0 and 1 corresponding to binary values of information. The principles of quantum mechanics, however, permits two-level quantum mechanical system to also encode superpositions of these two levels, namely, $|\Psi\rangle = c_0 |0\rangle + c_1 |1\rangle$, where we have used $|0\rangle$ and $|1\rangle$ to denote the normalized basis for this two-dimensional system. The complex-valued coefficients c_0 and c_1 represent the probability amplitude to reside in the 0 and 1 state, respectively, and the general term for such a state is a qubit. According to the quantum theory, the modulus squared characterizes the probability to observe the physical system in the corresponding basis state, e.g., the probability to observe the 0 state is $p_0 = |c_0|^2$, and similarly for 1. The second criterion implies that the levels 0 and 1 must be well-defined within the register element and that the ability to initialize the register element to a specific value of c_0 and c_1 is available. The third basic criterion for a quantum computer is the ability to measure register elements in this well-specified basis. As measurement corresponds to a sample drawn from the statistical distribution encoded by the quantum state according to the probabilities p_i over a given basis set, each measurement must accurately reflect the underlying distribution. Therefore, measurement is sampling and represents readout from the register of the quantum computer, which produces a classical binary value that may be subsequently processed.

The fourth criterion states the control over the register must include the ability to apply sequences of gates drawn from a universal set. Universality of the gate set characterizes the potential to perform an arbitrary unitary operation on the quantum state using a sufficiently long series of gates from that set. In particular, it is known that a finite set of gates is sufficient to approximate universality and, moreover, that a finite set of addressable one- and two-qubit gates are sufficient for universality.[11] The latter result, known as the Solovay-Kitaev theorem, provides a constructive method for composing arbitrary gates from a finite, universal gate set. Selection of a universal gate set raises the question of the optimal instruction set architecture for an intended application within a specific device technology.[12] The fifth criterion is that the gate operation times must be much shorter than the characteristic interaction times on which the register couples to other unintended quantum physical systems. These interactions induce decoherence of the stored quantum superposition states, which leads to the loss of information.[13,14] In order to maintain the stored quantum state with sufficient accuracy, the

duration of the gate sequence must be much shorter than the characteristic decoherence time. Fault-tolerant protocols for gate operations are designed to counter the losses from decoherence and other errors by redundantly encoding information with quantum error correction codes.[15]

Two additional functional criteria are necessary for operating a quantum computer with geometrical constraints on the layout of the quantum register elements. In particular, layout constraints may impose restrictions on which register elements can be addressed by multi-qubit gates, e.g., nearest neighbors within a two-dimensional rectangular lattice design. Physical layout restrictions may be overcome by moving stored quantum states between register elements. This is accomplished using the SWAP gate, a unitary operation that exchanges the quantum state between two register elements. In addition, a MOVE operation can support long distance transport of a stored value, in which the register element itself is displaced. The latter proves useful for distributed quantum registers that may requires interconnects, aka communication buses, to SWAP register values. The necessity of these functions depends on the purpose of the quantum computer and especially the limitations of the technology. Presently, all technologies for quantum computing face some constraints on register layout.

2.2 Component Model for a QPU

There are several different realizations of the quantum computational model which vary according to how the unitary operators, or gates, are implemented. We summarize two of the most prominent examples. The circuit model represents the sequence of unitary operators as an ordered sequence of discrete gates applied individually to register elements or perhaps applied to some subset of elements. The single and multi-qubit gates may be applied in series or in parallel depending on constraints placed on their execution. By contrast, the adiabatic model of quantum computation uses a single continuous time unitary applied global to all qubits simultaneously. Moreover, the applied operator must be restricted to adiabatically evolve the register, in which adiabaticity is defined according to the rate of change of the Hamiltonian relative to the internal energy scale of the quantum register. Nevertheless, the principles of operation for all of these quantum computational model begins with initializing the quantum register into a well-defined fiducial state followed by series of unitary operators applied to the register in series or parallel, and concluding with the register measured to generate the readout from the computation. This readout may be used to generate feedback that determines the next sequence of gate operations.

In Fig. 1, we present a component model capable of satisfying the functional requirements expected of a quantum processing unit. As we shall show, this model for QPU structure not only aligns with established separation of concerns in computer programming but it also mirrors the current implementation of experimental quantum processors. In general, a QPU defines a computational system that can allocate, initialize, transform, and readout a quantum register. The register is composed from addressable elements that store the quantum computational value, i.e., the quantum state representing a computational outcome. An individual quantum register element is modeled as a two-level quantum physical system capable of storing a single qubit of information. An array of n such elements has the capacity to store n qubits of information and, more generally, a quantum state in an 2^n-dimensional Hilbert space.

As quantum algorithms may be composed using gates applied to the register, it becomes necessary to implement multi-qubit gates in the general case. As noted above, multi-qubit gates are, at minimum, well approximated by sequences of one and two-qubit gates. Yet constraints imposed by the layout of the quantum register may limit the connectivity of multiple register elements. As some physical gates require that multiple elements be simultaneously addressable and spatially proximate due to the underlying Hamiltonian interactions. We therefore define register connectivity as the addressability of multiple elements during physical gate operation and we represent it by a graphical model of the possible interactions at any given instance of operation. Examples of the register connectivity include a complete graph, a linear chain (path graph) and a two-dimensional lattice (lattice graph). The connectivity of the register may not support gate operation requirements and, for these instances, additional swap instructions may be issued to reassign data stored in the register elements. However, modifying the register to satisfy a pending gate operation must be synchronized with subsequent instructions in order to ensure program correctness.

Applying gates to the quantum register elements implements control of the Hamiltonian $H(t)$ in Eq. (1). The leading approach to affect this control is the tailored application of analog fields, including optical, electrical,

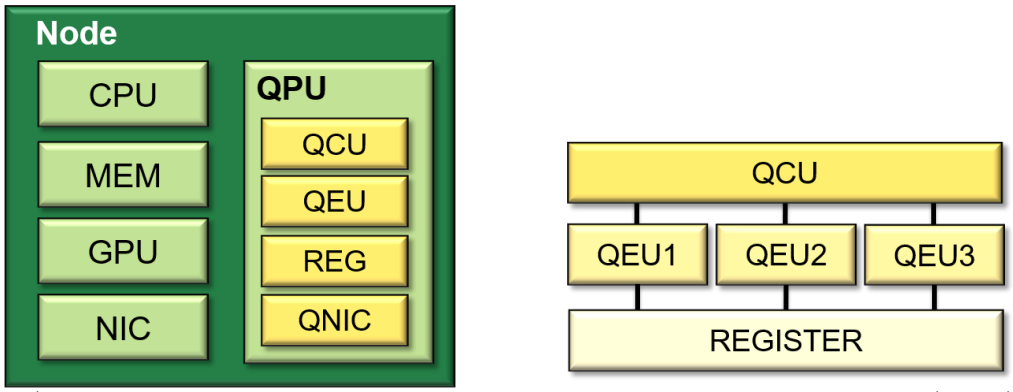

Figure 1: (left) Components of a computing node may include a CPU, memory system (MEM), GPU, and network interface (NIC) alongside a quantum processing unit (QPU). (right) A QPU may be decomposed into a quantum control unit (QCU) that manages quantum execution units (QEUs) that apply gates onto a register.

and magnetic, to the quantum physical system. Depending on the nature of the physics and the coupling interaction, the timing, amplitude, and phase of the field are designed to drive the encoded quantum state to a desired outcome. The efficacy of this design depends on the noise in the field generation and delivery as well as the extraneous coupling between other quantum physical systems. We model the components within the QPU that carry out the synthesis and delivery of the analog field to the quantum register as the quantum execution unit (QEU). Moreover, multiple QEUs may exist with each specialized by its gate execution or connectivity constraint. In practice, QEUs are implemented using arbitrary waveform generators (AWGs), which take as input specification for a waveform design and produce as output a desired analog electrical field. Less robust implementations may rely on function generators to perform the same task with limited variability. In either case, the output electrical field drives a signal generator of the necessary type, for example, modulated laser, microwave or radio frequency generator, etc., as required by the gate design.

The QEUs also serve to implement measurement, which is important for readout from the quantum register. Measurement of the register requires the detection of a field that characterizes the quantum physical state within a specified basis. This may include detection of the field to have a specific energy, time of arrival, current, voltage, etc. that characterizes the state in the defined basis. Often times, the QEU must apply a field in order to trigger a response from the register that may be detected. This requires synchronization between the application of the trigger field and the detection of the response field. This synchronization may come from either temporal scheduling of the fields, i.e., windowing, or it may results from spatial synchronization, e.g., dedicated channels. In either case, detection of the response field becomes a decision as to whether the field was present or not. This is a translation from the analog to digital domain and represents gate execution.

We have ascribed to QEUs the direct application of gates to register elements but the decision to apply a specific gate is made elsewhere. The component responsible for dictating the flow of gate execution is the quantum control unit (QCU). Given a sequence of instruction which must be carried out by the QPU, the role of the QCU is to decide how to carry out those instructions. This separation of concerns in the execution of a program is made to ensure that the tasking of multiple QEUs can be optimized with respect to the available resources and pending gate applications. In particular, the QCU must configure the flow of data to the relevant QEUs and this may necessitate decisions that decode the received instructions to achieve the desired logic. The resulting sequences of gates, or operands, that perform the equivalent logic as the original instruction may similarly generate a response that requires subsequent interpretation by the QCU. The implementation of the QCU may rely on fixed logic, for example, using application-specific integrated circuits (ASICs), or it may be implemented using conditional logic. In either, the translation of an instruction into a sequence of operands requires the QCU to have a fine grain understanding of register connectivity, QEU functionality, scheduling constraints, and timing of gate executions.

The QCU implements the instruction set architecture (ISA) that is native to the QPU. The ISA exposes the functionality of the QPU and provides the interface by which processing requests are submitted. The ISA for

a QPU may be tailored to the underlying technology by accounting for the constraints of register layout and gate execution that arise from the type of physical system, the computational model, and the signal generators. Moreover, the choice of ISA enables the QPU developer to tailor the interface to specific application concerns and computational models. This includes future definitions for standardized ISAs that enable portability between processing requests. This description emphasizes that the ISA provides a specification of the operations that can be interpreted by the QCU and translated into QEU operands. However, the ISA may also include conventional logic that is interpreted by the QCU. This includes conditional statements for which different instruction branches are executed depending on the local state of the QPU. For example, measurement outcomes may be used to evaluate conditions as to whether additional instructions must be performed to ensure a register is prepared in the desired state.

A notable concern for future QPU implementations is the development of fault-tolerant protocols that enable resilient execution of instruction sequences. The theory of fault-tolerant quantum computing currently relies on redundant encoding of the stored quantum state through the use of quantum error correction codes. Monitoring the collective encoded state through syndrome measurements using ancillary elements provides feedback that can trigger a corrective action by the QPU. An important design choice for future fault-tolerant QPUs is where to allocate this functionality. In our description of components within a QPU, it is not possible for the QEU to trigger measurement-driven feedback due to a lack of conditional logic. In addition, the QEU may be specialized to a given gate operation or subset of register elements that inhibit the ability to carryout QEC. Therefore the QCU is the default component from which fault-tolerant protocols may be implemented. An alternative to this design choice is to allocate management of the fault-tolerant protocol to a component outside of our current QPU model. While this alternative is perhaps likely for near-term QPU experiments, the design choice faces the burden that the necessary knowledge about the register state, the translation of instructions into operand sequences, and the constraints placed on the QEU execution must be exposed to the external manager.

3. PROGRAM EXECUTION MODEL IN A HYBRID COMPUTING SYSTEM

We have described the component model for a QPU in terms of a quantum control unit (QCU), quantum execution units (QEUs), and a quantum register. This has defined a set of component interfaces that necessitate a hierarchy of languages to enable programming the QPU. In this section, we discuss a model for execution of a complete program within the context of a hybrid computational node such as the one shown in the left panel of Fig. 1. For this execution model, a program loaded into the CPU main memory (MEM) is executed according to the operating principles of host node, i.e., the operating system. We model interactions between the host and the QPU using an accelerator programming model, whereby specific tasks are offloaded to the QPU for execution. The accelerator programming paradigm allocates specialized processing functionality to the attached accelerator device, which is a QPU in our model. The justification for this distinction is that the QPU can perform certain tasks more efficiently than the host processing system, i.e., the CPU. For the case of quantum accelerator programming, the identification of suitable tasks for QPU acceleration is made by a performance analysis that accounts for the expected computational complexity of the programmed method as well as the overhead required to offload the task to the specialized device. Notably, execution of a specialized tasks on the QPU requires the transfer of information between the host program and the attached accelerator. Depending on the technology model, communication between these devices may occur using memory buses such as PCI express or long-range communication networks such as Internet. For any interaction model, the data transfer and communication latency play a role in assessing the overall impact on performance. This is in addition to the intrinsic performance of the accelerator to perform the desired task.

Current QPUs represent a loosely integrated hybrid model of computation. Specifically, commercially available QPUs currently rely on a client-server interaction implemented using a representational state transfer application programmers interaface (REST API) over Internet with the HTTPS protocol. This permits the host program to push job requests to an external server that drives the execution of a connected control system. This control system represents the QCU while the subsequent digital to analog converters (DACs) and/or AWGs form the interface to the QEUs. Within the control server, the transfer of instructions representing the tasks allocated to the accelerator transfer data (code) through the main memory system of the server system. The fundamental interaction between conventional program instructions and the QPU is via the memory system.

The language for initiating and managing the transfer of tasks to a QPU can be varied. However, we have recently developed a directive-based approach to accelerator programming for allocating tasks to specific accelerator devices using a programming framework known as XACC.[16] XACC relies on a mixed-language representation to enable a single program to encode both conventional and quantum instruction sequences. In general, a program written to control the behavior of a QPU must be compiled into instructions that can be parsed by a machine.[17] In the accelerator programming paradigm adopted here, the host CPU must parse instructions for its native interpretation and it must issue instructions for the QPU to that device. A version of the written program, i.e., source code, is compiled into the machine readable format. The compiler targets the CPU and QPU ISAs for each type of coded instruction. Using XACC, directive keywords identify the presence of a quantum computing source code and indicate the necessary compiler.

3.1 Simulation Method

We have presented a model for the components of a hybrid computing system, a hierarchy of languages to communicate between these components, and a model for how these components execute the communicated instructions. In order to simulate this model, we rely on a discrete-event simulator, in which events represents transactions between components in the form of transmitting and receiving messages. The message packets deliver information that dictates the behavior of the receiving component. For example the QCU receives a messages packet relaying a sequence of instructions within the specified ISA while the QEU receives a packet specifying the gate operation codes issued by the QCI. Each component processes the received message packets according to a behavioral model that defines how subsequence messages, i.e., events, are generated.

We model the entire system using the Structural Simulation Toolkit (SST), a discrete-event simulation framework that has been used previously for modeling GPU accelerator architectures. SST provides a core discrete-event simulator, including an event manager, as well as a framework of classes, methods, and data types to model development. In addition, SST offers a variety of existing components that can be reused in our quantum-accelerator model. In particular, models for the memory hierarchy that includes main memory, cache, and memory controllers are useful for interfacing conventional CPU models with new QPU models. We extend the framework by introducing new quantum components that model the QPU, QCU, QEU, and register. These extensions are then customized to a chosen implementation of the language hierarchy and the necessary message packets are defined according to the component interface language.

The purpose of the simulation is to gain insight into the behavior and performance of the hybrid computing system. Performance may be quantified in terms of metrics such as time and energy, while behavior may be quantified by correctness in program execution as well as performance robustness with respect to changes in program design. We use the SST model to quantify the time required for program execution as well as the energy consumed. These metrics can be estimated by sampling the simulation and accumulating statistics that represent the aggregate behavior.

An SST model is defined by first specifying and then programming the necessary components using C/C++. These component classes are then compiled into a shared library that provides a Python wrapper to each method. The SST model is then composed by declaring instances of the components within a Python script and linking those components together into a defined network. Executing the Python script within the SST run-time environment initiates the discrete-event simulator to start simulating each component. Events are generated when each components interfaces with the simulations messaging systems, and the event manager routes these messages to the destinations based on the established connectivity. A fundamental parameter within the simulation is the clock that describes the frequency with which the event manager monitors changes in components. In addition, there are clocks for each component that define the frequency with which the internal state is refreshed. These clocks can be tuned to vary the granularity of the simulation. In addition, the initial state of the components can be defined by passing parameters when the components are declared.

4. CONCLUSION

We have presented an overview of methods and interfaces to simulate the execution of a computing system composed from both quantum and conventional processors. We have emphasized that the behavior of this system can be captured by using a hierarchy of languages that model the translation of user-defined programming

statements into the fundamental changes of a quantum physical system. We have further defined how these translations can be executed at run-time by specifying an execution model for the hybrid system. It is notable that we have relied on an accelerator programming paradigm, whereby directives are used to offload specific tasks to the targeted QPU. In the accelerator paradigm, a mixed-language program allocates specific sequences of instructions to the targeted device. The XACC programming framework offers an implementation of these ideas for quantum processing.

We have further described the methods for simulating the abstract machine models defined by these components and the associated language hierarchies. BY relying the SST discrete event simulator, we have defined how messages between components, i.e., events, may be used to trigger model behaviors. From these model behaviors, we can simulate the concurrent operation of multiple components to calculate the state of the complete hybrid computing system. Statistics that characterize the behavior of the system, e.g., in terms of metrics quantifying time and energy consumption, can be estimated by sampling the simulation.

There are numerous questions that may be posed with the simulation of hybrid computing systems. In particular, many novel algorithms have been developed for the quantum computational model. The foremost example relevant to scientific discovery is quantum simulation, that is to say, the simulation of Schrodinger's wave equation. This algorithm epitomizes the potential of quantum computing (it is BQP-complete), and it has immediate application to computational chemistry for electronic structure, materials theory for multi-site correlations, and high-energy physics for scattering amplitudes. In addition to these direct applications of quantum simulations, other algorithms for calculating partition functions, solving linear systems or differential equations, and minimizing unstructured functions. Yet the understanding of how this relatively straightforward quantum algorithm can be realized within a hybrid computing system is an outstanding concern for the development of the technology.

REFERENCES

[1] T. M. Conte, E. Track, and E. DeBenedictis, "Rebooting computing: New strategies for technology scaling," *Computer*, vol. 48, no. 12, pp. 10–13, 2015.

[2] J. M. Shalf and R. Leland, "Computing beyond Moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, 2015.

[3] K. A. Britt and T. S. Humble, "High-performance computing with quantum processing units," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 39, 2017.

[4] Department of Energy Advanced Scientific Computing Research, "Extreme Heterogeneity Workshop." https://www.orau.gov/ExHeterogeneity2018. Accessed: 17 March 2018.

[5] K. A. Britt, F. A. Mohiyaddin, and T. S. Humble, "Quantum accelerators for high-performance computing systems," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–7, 2017.

[6] A. Geist and R. Lucas, "Major computer science challenges at exascale," *Int. J. of High. Perform. Comput. Appl.*, vol. 23, p. 427436, 2009.

[7] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright, "The opportunities and challenges of exascale computing," tech. rep., Summary Report of the Advanced Scientific Computing Advisory Committee Subcommittee, 2010.

[8] J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, A. Geist, G. Grider, R. Haring, J. Hittinger, A. Hoisie, D. Klein, P. Kogge, R. Lethin, V. Sarkar, R. Schreiber, J. Shalf, T. Sterling, and R. Stevens, "Top ten exascale research challenges," tech. rep., DOE ASCAC Subcommittee Report, 2010.

[9] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," in *Proc. R. Soc. Lond. A*, vol. 400, pp. 97–117, The Royal Society, 1985.

[10] D. P. DiVincenzo *et al.*, "The physical implementation of quantum computation," *arXiv preprint quant-ph/0002077*, 2000.

[11] C. M. Dawson and M. A. Nielsen, "The Solovay-Kitaev algorithm," *arXiv preprint quant-ph/0505030*, 2005.

[12] K. A. Britt and T. S. Humble, "Instruction set architectures for quantum processing units," in *Lecture Notes in Computer Science*, vol. 10524, pp. 98–105, Springer, 2017.

[13] M. Schlosshauer, "Decoherence, the measurement problem, and interpretations of quantum mechanics," *Rev. Mod. Phys.*, vol. 76, pp. 1267–1305, 2005.

[14] A. Streltsov, G. Adesso, and M. B. Plenio, "Colloquium: Quantum coherence as a resource," *Rev. Mod. Phys.*, vol. 89, p. 041003, Oct 2017.

[15] E. T. Campbell, B. M. Terhal, and C. Vuillot, "Roads towards fault-tolerant universal quantum computation," *Nature*, vol. 549, no. 7671, p. 172, 2017.

[16] A. J. McCaskey, E. F. Dumitrescu, D. Liakh, M. Chen, W. Feng, and T. S. Humble, "Extreme-scale programming model for quantum acceleration within high performance computing," *arXiv preprint arXiv:1710.01794*, 2017.

[17] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, p. 020501, 2018.