High-Performance Computing with Quantum Processing Units

KEITH A. BRITT, University of Tennessee and Oak Ridge National Laboratory
TRAVIS S. HUMBLE, Oak Ridge National Laboratory and University of Tennessee

The prospects of quantum computing have driven efforts to realize fully functional quantum processing units (QPUs). Recent success in developing proof-of-principle QPUs has prompted the question of how to integrate these emerging processors into modern high-performance computing (HPC) systems. We examine how QPUs can be integrated into current and future HPC system architectures by accounting for functional and physical design requirements. We identify two integration pathways that are differentiated by infrastructure constraints on the QPU and the use cases expected for the HPC system. This includes a tight integration that assumes infrastructure bottlenecks can be overcome as well as a loose integration that assumes they cannot. We find that the performance of both approaches is likely to depend on the quantum interconnect that serves to entangle multiple QPUs. We also identify several challenges in assessing QPU performance for HPC, and we consider new metrics that capture the interplay between system architecture and the quantum parallelism underlying computational performance.

CCS Concepts: • **Computer systems organization → Quantum computing**

Additional Key Words and Phrases: Quantum computing, high performance computing, accelerator

ACM Reference Format:
Keith A. Britt and Travis S. Humble. 2017. High-performance computing with quantum processing units. J. Emerg. Technol. Comput. Syst. 13, 3, Article 39 (March 2017), 13 pages.
DOI: http://dx.doi.org/10.1145/3007651

1. INTRODUCTION

High-performance computing (HPC) has historically taken advantage of new processing paradigms by leveraging special-purpose accelerators. This includes the use of algorithmic logic units and floating-point units in early processor architectures as well as more recent vector processors capable of single-instruction multiple-data (SIMD) parallelization. Current interest in graphical processing units (GPUs) is another example of the ongoing trend in accelerator use for HPC development. A primary motivation for the accelerator paradigm is that low-level processes can take advantage of specialized hardware while minimizing changes to overall program structure [Kindratenko et al. 2008]. This approach isolates the need for program or algorithm refactoring to those workloads specific to the hardware accelerator [Schneider 2015]. A secondary motivation is that the accelerator model offers an opportunity to take advantage of emerging technologies while also mitigating the technical risk to system development.

This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.
Authors' addresses: K. A. Britt and T. S. Humble, P.O. Box 2008, M.S. 6015, Oak Ridge National Laboratory, Oak Ridge, TN 37830; emails: {brittka, humblets}@ornl.gov.

Current limitations on processor frequency, communication bandwidth, physical scale, energy consumption, and hardware reliability make it advantageous for HPC designers to anticipate new technologies and to leverage architectures that support innovative platforms. Future efforts to realize HPC beyond the current exascale target are likely to require such innovations [Ang et al. 2010; Ashby et al. 2010; Geist and Lucas 2009].

The search for technology paths that lead to performance beyond exascale may require alternative computational models. This is because some problems are better suited to computational models other than the standard Turing machine model. In particular, quantum computing has attracted significant interest due to theoretical results that exponential reductions in algorithmic complexity of some problems are possible relative to the best-known conventional algorithms [Simon 1997]. This includes the factorization of integers, a staple of public-key cryptography [Shor 1997]; *ab initio* calculations of electronic structure in chemistry and physics [Kassal et al. 2011]; and scattering amplitudes of particles in high-energy physics [Abrams and Lloyd 1997]. These algorithmic speedups are achieved by leveraging the unique features of quantum mechanics, namely superposition, entanglement, and intrinsic randomness. The basic principles of quantum computing have been demonstrated in small-scale experimental systems, and there is an on-going, global effort to develop large-scale quantum computing platforms.

The opportunities afforded by quantum computing represent a challenge to the HPC accelerator model, which previously has been practiced exclusively within the setting of the classical, deterministic Turing model. By contrast, many quantum algorithms lack clearly defined kernels that can be off-loaded to a quantum computational accelerator. The mixture of computational models also stymies efforts to leverage conventional notions of SIMD and multiple-instruction multiple-data (MIMD) parallelism. Parallel computing typically makes use of domain decomposition [van Nieuwpoort et al. 2001], whereas quantum algorithms frequently intentionally avoid this type of problem partitioning [Bauer et al. 2016; Kreula et al. 2015]. In addition, domain decomposition exposes an interface between classical and quantum computational models that is not yet well defined. Translation between computational models may be theoretically possible but making these interfaces efficient and robust is an outstanding concern.

As constraints on the physics underlying quantum computation limit how these resources may be used, it is unclear if and how emerging quantum processors will become compatible with existing or future HPC platforms. We analyze the integration of these quantum processing units (QPUs) for modern HPC architectures. We place an emphasis on conceptual differences between the conventional and quantum computing models that may be expected to challenge integration. Our analysis examines two pathways that lead to several abstract machine architectures. We identify those distinguishing features that QPUs may be expected to exhibit and the dimensions that will be most useful for characterizing their performance metrics within a hybrid system.

The article is organized as follows. In Section 2, we characterize the features of a QPU, briefly summarize its operating principles, and identify requirements for operation that influence HPC integration. In Section 3, we examine three multi-processing models for adopting QPUs into conventional HPC systems and the architectures that arise from them. In Section 4, we describe the need for both standardized as well as unique performance metrics to characterize HPC with quantum accelerators. We offer final remarks in Section 5.

2. QUANTUM PROCESSING UNIT

We define a QPU to be a computational unit that uses quantum computing principles to perform a task. As the operating principles of a QPU are based on quantum mechanics, there are several unique features that do not have analogs in conventional computing platforms. Foremost, a QPU stores computational states in the form of a quantum mechanical state. While a quantum state is formally defined as a unit vector in a finite-dimensional Hilbert space [Nielsen and Chuang 2000], it must also be interpreted as the data processed by the QPU. The simplest and most frequently used example is a qubit, which expresses a state within a two-dimensional Hilbert space. These states are stored in quantum physical systems. For the purpose of clarity, we define a quantum register as an addressable array of two-level quantum physical systems. We will refer to an individual system within the register as a quantum register element, and we will assume that each register element can store a qubit of information. We may refer to the size of the register by the number of qubits that it can store, for example, an n-qubit register.

The computational space available to a quantum register scales exponentially with its size. Like an n-bit register, an n-qubit register is capable of representing all 2^n computational states. However, the quantum register is also capable of representing superpositions of these states simultaneously, some of which cannot be expressed classically, using a phenomenon known as entanglement. Fundamentally, entanglement is a limitation on the ability to describe states of a register solely by specifying the value of each register element. This is in stark contrast to classical models of computation and leads to a description of what has been called the "inherent parallelism" of quantum computing [Deutsch 1985]. This inseparability of register states manifests as perfect correlations during computation, and many quantum algorithms take advantage of entanglement to realize computational speedups [Childs and van Dam 2010].

Operations on the quantum register are realized using gates. Like conventional computing, quantum gates correspond with well-defined transformations of the computational state. When the register is prepared in a superposition state, operations effectively act on multiple computational states in parallel. This may be viewed as a quantum variant of conventional SIMD processing. However, quantum computing makes use of gates that are either unitary transforms of the register elements or projective measurements. Only the latter gates prepare the state of the quantum register in a well-defined classical value, for example, either a 0 or 1 for each register element. For a unitary gate, the value of the register remains in a superposition of computational states and serves as an intermediate computation. Ultimately, the solution to a computation is recovered when a projective measurement gate is applied to the register. The resulting bit string must then be stored in a classical register within the QPU.

Quantum computational models define how the registers and gates within a QPU realize quantum computation. While all the models offer identical computational power from a complexity perspective, they do differ with respect to hardware implementations and principles of operation. For example, the quantum circuit model is closely related to the conventional representations of classical circuits, as it uses sequences of discrete gates acting on registers to generate a series of computational states. By contrast, the adiabatic quantum computing model uses a continuous, time-dependent transformation of the interactions between register elements to evolve the computational state toward a solution. For example, recent special-purpose processors within the adiabatic quantum computing model implement quantum optimization using a single instruction that is tunable in duration [Johnson et al. 2011]. Across all computational models, QPUs require precise control over the quantum physical degrees of freedom defining register elements. There is an ongoing effort to demonstrate proof-of-principle registers and gates within a variety of physical systems, including silicon donor systems [Saeedi et al. 2013], trapped ions [Monroe and Kim 2013], and superconducting circuits [Devoret and Schoelkopf 2013]. A specific focus has been on realizing high-fidelity implementations that can support fault-tolerant operation. In addition, there has been some work to design the physical layout and instruction architectures for certain

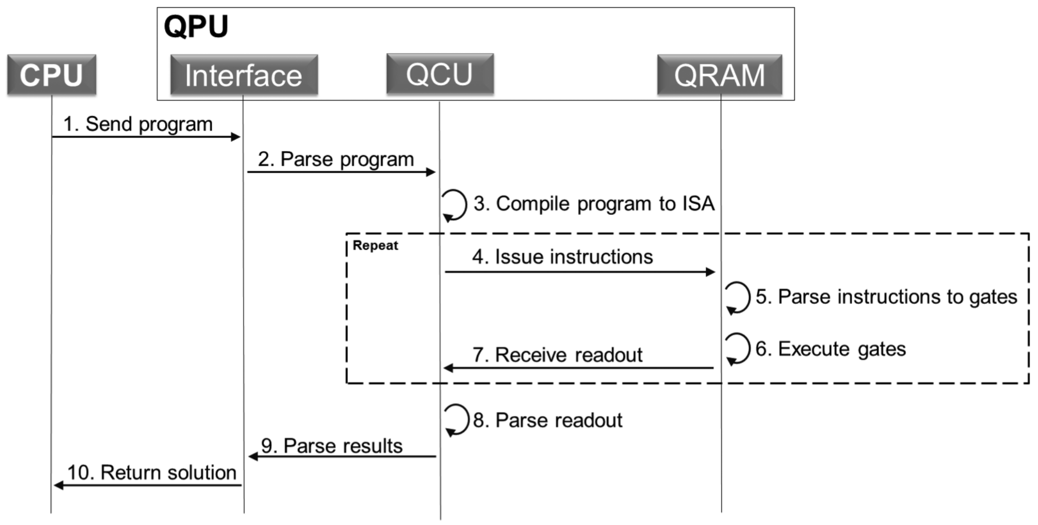

Fig. 1. A sequence diagram modeling the interactions between a host CPU and a QPU. The QPU interface defines the internal QRAM model and drives gates operating on the register. The QCU parses the incoming instructions and the outgoing response according to the computational model and device physics.

technologies [Meter and Oskin 2006; Thaker et al. 2006; Van Meter et al. 2010; Jones et al. 2012; Metodi et al. 2011; Hill et al. 2015; Humble et al. 2014].

A typical usage case for a QPU begins by preparing the quantum register in a well-defined initial state and then applying a sequence of gates that may act on individual or multiple register elements. This is commonly referred to as the quantum random access memory (QRAM) model, first articulated by Knill [1996]. We define a QPU to include a QRAM that applies low-level gates to register elements, a quantum control unit (QCU) that parses programs into instructions, and a classical controller interface that defines how the central processing unit (CPU) within a host system interacts with the QPU. The exact sequence of gates is determined by the host program, which the QCU parses into an intermediate representation using an instruction set architecture (ISA). The ISA represents a set of high-level instructions that are available for programming the QRAM. Instruction sequences are generated when a compiled program is decoded by the QCU, after which the instructions are parsed by the QRAM into gates, that is, machine codes, that are specific to the QPU technology base. At present, there are a variety of technologies under consideration for quantum computing, and they each support different sets of gates that are native to their respective technology bases. The unique ISA defining each technology base indicates the need for abstraction of the QCU interface. The standardization of the QCU interface will not only promote robust QPU design across different technologies but also assist integration of these devices into larger system designs.

Applying the QRAM model within the QPU context defines an interface with the classical controller allocated to the host CPU (see Figure 1). The CPU tasks the QPU by submitting a program and then waits for the reply. These tasks must represent quantum computational workloads that can be parsed out by the QCU, where the interface may be implemented in software or hardware. Development of these interfaces is still an open question, although recent progress has been made in defining quantum programming languages for this purpose [Selinger 2004; Green et al. 2013; Wecker and Svore 2014; Abhari et al. 2012]. These languages offer exposure to the gates and registers needed for programming low-level quantum algorithms, and we expect future generations will likely grow to addressing additional data structures and instructions. Interfaces that encapsulate and mask quantum hardware details from the software developer are especially important for maintaining existing applications across a variety of HPC environments. As an example, an application program interface that requires software developers to integrate quantum processing instructions directly into the code base will prevent its adoption (due to the burden of code rewriting).

In addition to local operations driven by a host CPU, a QPU may also interact directly with other QPUs. This may be necessary to communicate a computational state between QPUs or to prepare both registers in a mutually entangled state. These operations require the presence of a quantum network, which uses quantum physical systems to communicate quantum states between registers. However, a notable feature of quantum computing is that intermediate computations cannot be copied during the communication. This is a consequence of the no-cloning theorem from quantum mechanics that limits the precision with which arbitrary quantum states can be duplicated. Instead, communication between QPUs must use either direct transmission or teleportation [Nielsen and Chuang 2000]. Direct transmission transfers the value of the first register over the quantum network by using a mobile quantum carrier. On receiving the transmission, the second QPU swaps this state into its register. This approach most closely resembles conventional read-write communication in an HPC network. But quantum communication also supports teleportation, which allows for the value of a first register to be transferred to a second register without passing through the quantum network. Instead, teleportation uses pre-existing entanglement between the QPUs to perform the data transfer. This does also require transmission of classical side-channel information from the first QPU to the second; however. this classical information is generally much less than the information needed to describe the value of the quantum register.

3. QPU INTEGRATION STRATEGIES

The simplified CPU-QPU execution model presented in Section 2 offers a variety of different integration strategies for the development of large-scale hybrid computing systems. A significant obstacle to integration is the physical hardware requirements of current experimental quantum computing devices. Many technologies that could be used to realize a fully functional QPU currently require bulky and costly infrastructure. This includes the use of dilution refrigerators to suppress thermal noise, electromagnetic shielding to avert ambient energy, and ultra-high vacuum enclosures to prevent device contamination. In addition, most devices require relatively complex electronic and optical control systems that must cross the physical barriers to the processor. An exemplary system schematic is shown in Figure 2.

We anticipate that QPU requirements will ease with future device development and refined engineering principles. For example, recent work on ultra-cold operation of field-programmable gate arrays (FPGAs) to drive silicon qubits suggests there is a path toward integration of the QPU control interface within the dilution refrigerator [Hornibrook et al. 2015]. Similarly, progress in the miniaturization of electronic controls for linear optical quantum chips hints at scalable operations in the future [Carolan et al. 2015]. However, these devices still remain far from the typical hardware environments on which modern HPC systems have been built, namely, room-temperature operation, direct interaction with the host CPU, and easily managed footprints for individual processors. Consequently, integration opportunities for QPUs naturally separate into loosely and tightly bound systems.

In the loose integration path, QPUs remain as isolated operational elements that must interact with a host HPC system using a network interface. This is effectively a client-server model as shown in Figure 3 where the quantum computing (QC) server may either be on a dedicated network or part of a larger computational grid. In this

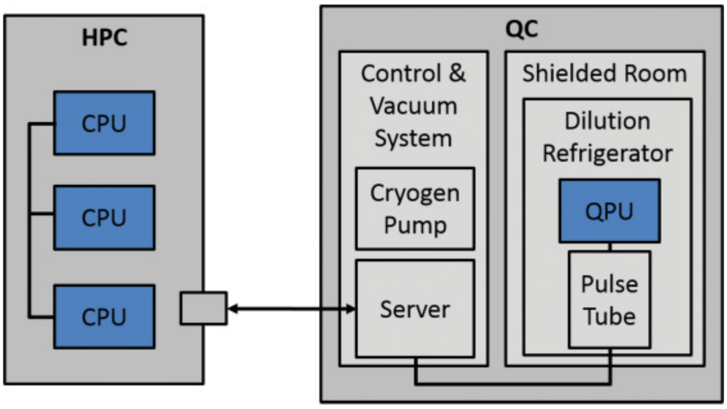

Fig. 2. Asymmetric multi-processor architecture for integrating a stand-alone quantum computer (QC) with an HPC system. We highlight components of the QC system that represent the substantial infrastructure required for interfacing and controlling the QPU.

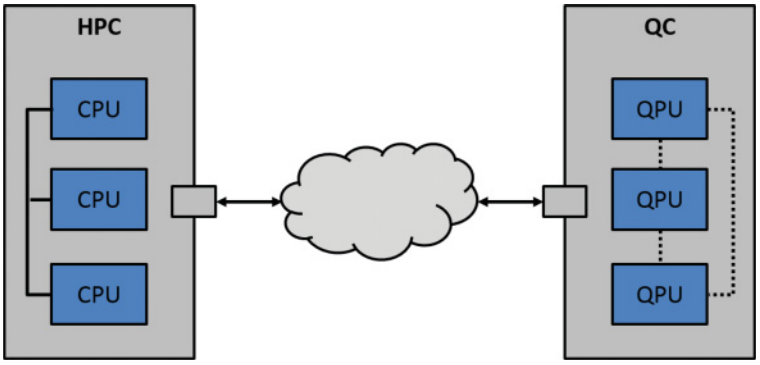

Fig. 3. An asymmetric multiprocessor model employing a quantum computing (QC) server, for example, a form of cloud-based quantum computing. The dashed lines indicate a quantum interconnects between QPUs while solid lines indicate classical interconnects. The concept of QC as a service offers increased flexibility and ease of use at the expense of communication latencies. Latencies will contribute to overall execution timing and, depending on problem and program structure, could partially negate quantum computational advantages.

asymmetric multi-processor model, the network communicates requests between the host (client) system and the QC server. As indicated in Figure 3, the QC server may host multiple QPUs and these may interact via a quantum interconnect. However, the entry point into the system remains the primary bottleneck. This connection can be streamlined when both systems are within a local area network, but access to each QPU must still be provisioned by the QC control system. This control system may appear as a switch that forwards program data to individual QPUs, or it may more intelligently route programs based on QPU usage and demand.

The demands of the client-server model in Figure 3 force a tradeoff between the communication latency and the computational speedup gained from using a QPU instead of a CPU (or some other local resource). This tradeoff is advantageous when the communication time is offset by the QPU speedup, but this will depend on problem type as well as size. Moreover, evaluation of the model is complicated by the communication patterns arising from multiple CPU nodes and any latency they may experience from resource competition. Therefore, the client-server model is likely to be broadly useful

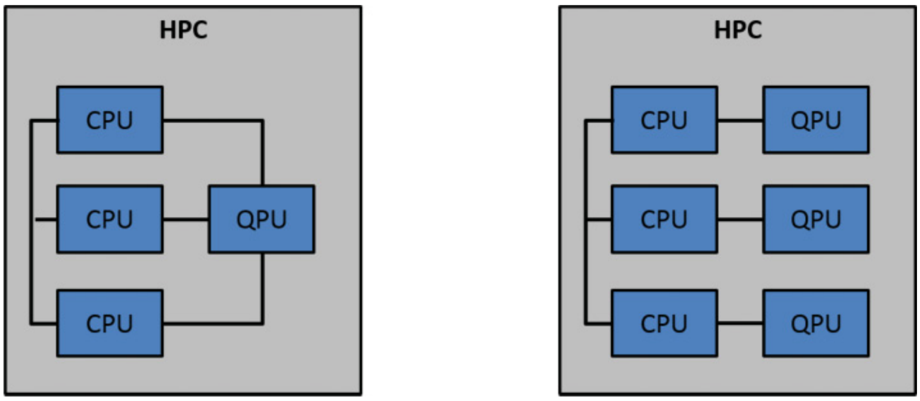

Fig. 4. (Left) A shared resource model in which a single QPU is accessed by multiple CPU nodes. (Right) A standard accelerator model in which QPUs are attached to nodes hosted on a classical interconnect. The absence of quantum networking between QPUs restricts the scaling with respect to the quantum resources and enforces a classical domain decomposition paradigm.

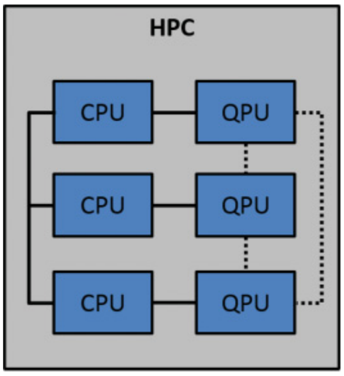

Fig. 5. An accelerator model with QPUs incorporating a quantum interconnect that supports both quantum and classical parallelism. QPUs may be addressed individually or collectively through the coordinated CPU elements.

only when the computational gain over conventional approaches is significant. This adds emphasis to the importance of the underlying quantum algorithm.

The loosely integrated client-server model also supports the alternative use case of cloud-based quantum computing. In this setting, the QC server is a rare resource in demand from multiple users simultaneously. For a system containing q QPUs, the server can support classical MIMD parallelism with each node performing an isolated job. As a measure of server capacity, the dimension of the server Hilbert space is $q2^n$ given an n-qubit register on each node. By contrast, the presence of a quantum interconnect linking the individual QPUs offers a Hilbert space of 2^{nq}. This exponential increase in server capacity with node number is not a guarantee of computational speedup. The cloud-based quantum computing model may be especially attractive for blind quantum computing, which permits a user to submit a job request without revealing details about either the data or instructions [Broadbent et al. 2009].

The tight integration path is shown in Figures 4 and 5 and represents a progression toward more sophisticated accelerator models. The goal of this design is to move the QPU closer to the host node in order to eliminate communication latency and maximize computational speedup. This design assumes the hardware requirements for QPUs can be eased to the point that a single, tightly connected single-system model can be created. As mentioned above, this will require multiple advances in the classical infrastructure used by current experimental devices. Figure 4 (left) represents a first design based on a shared resource model that permits multiple CPU nodes to interact with a single QPU node. Like the server model, a single QPU is responsible for managing requests for multiple CPUs and requires a robust classical controller interface. This communication design also represents a bottleneck but the tighter integration alleviates some of the overhead required in the loose model. In addition, data from multiple CPUs can be aggregated by the QPU. This use case may appear when pre-processing of the input for the QPU can be parallelized across CPU nodes. Alternatively, if the QPU is part of a MIMD or data streaming model, then the redirection of QPU results to another node may be useful.

A more pronounced example of the accelerator model is presented in the right panel of Figure 4. This design most closely matches that used for integrating GPU accelerators into modern HPC systems as each CPU node is tightly integrated with a dedicated QPU. This greatly simplifies the QPU interface. This hardware model also naturally matches many existing program data access patterns, in which top-level memory management is driven by domain decomposition with low-level data movement restricted to single nodes. In this sense, the design is motivated by an initial application of classical parallelism and subsequently followed by quantum parallelism. While this model offers the appeal that it would minimize the refactoring required of existing source codes, it also restricts the amount of quantum parallelism available.

The accelerator design allows for running multiple quantum algorithms simultaneously on independent QPUs. In the near term, we envision QPU size to be the significant factor limiting problem size, so applications that can take advantage of this loose integration design are the applications most likely to see significant benefits early from the integration of QPUs into HPC systems. However, this is not the only approach to using the accelerator architecture. As a form of this type of domain decomposition, divide-and-conquer offers a classical approach to parallelism that is needed to justify a network of isolated QPUs. For example, recent work by Peruzzo et al. have shown variational eigensolvers for computational chemistry can accommodate a divide-and-conquer strategy for calculating the lowest-energy ground state [Peruzzo et al. 2014]. However, for many quantum algorithms, parallelizable divide-and-conquer strategies have not yet been developed and may not even be possible. This poses a problem when a single QPU does not contain enough qubits in its registers to support the specified size of a desired algorithm instance. The additional QPUs available in the accelerator architecture do not enable the user to scale the problem size up, because there is no entanglement across QPUs. Instead, quantum algorithms that utilize divide-and-conquer approaches are needed [Peruzzo et al. 2014; Yung et al. 2014].

Figure 5 represents an accelerator model in which individual QPUs are interconnected. The quantum interconnect establishes quantum communication between each QPU and offers the possibility of generating entangled states between registers. Like the conventional interconnect used to establish communication between CPU nodes, the quantum interconnect will require additional switches and possibly routers to perform robust communication [Dasari et al. 2016]. In this tightly integrated limit, a collection of interconnected QPUs may be abstracted as a single QRAM accessed through interfaces at multiple CPU nodes. This design offers the benefit of maximizing the potential quantum parallelism by allowing for a single quantum algorithm to be distributed across multiple QPUs (without a divide-and-conquer strategy). This tight integration design creates two partitions, the first being the set of QPUs, which hold all the data used by an algorithm, and the second being the set of CPUs, which hold all the instructions used by the same algorithm.

The interfaces between CPUs and QPUs, while varying in their implementation, are well understood to be classical in nature, meaning that CPUs will pass classical states to the QPUs, which will in turn be transformed into quantum states via a digital-to-analog converter (DAC) and QPUs need only pass back classical states to CPUs (see Figure 1). The introduction of a quantum interconnect, however, presents the opportunity for the communication of sustained superposition and entangled states. As mentioned above, at a high-level, this could be implemented as a single QRAM interacting with multiple QPUs attached to their respective CPUs. Alternatively, the concept of *flying qubits* [DiVincenzo 2000] is well understood today, and numerous physical implementations have been implemented using optical technology, which may serve as a data passing medium of qubit states between multiple QPUs.

4. PERFORMANCE METRICS

An important aspect in evaluating the merits of QPUs for use in any of the system integration strategies is identifying performance metrics that are both well defined and meaningful. Basic metrics for conventional computing typically focus on register size, word size, floating point operations per second (FLOPS), and so on. More elaborate measures take account of multi-threaded processor pipelines, memory access speeds, and communication latency. Processing models that include SIMD and MIMD parallelism are also judged according to use cases and targeted problem sets. Each metric can also be monetized, for example, FLOPS per watt and FLOPS per unit currency, to address specific stakeholder interests.

Despite a long history of evaluating conventional HPC systems, the unique features of QPUs pose new challenges in measuring performance. First and foremost is the difference in the underlying computational model. For example, QCUs and QRAMs will certainly require clocks to execute the instructions and machine codes acting on the quantum register, but the speed of the clock is not directly proportional to the speed at which gates are applied. Instead, the gates themselves are complicated sequences of control signals applied to the physical system and they cannot be made arbitrarily fast. The speed of gate execution is complicated further by the use of fault-tolerant (FT) instruction protocols. These protocols protect against errors but necessarily require the use of additional primitive gates and qubits [Nielsen and Chuang 2000]. Fault-tolerant instruction protocols may vary with program sequence as well as data locality. Since the context of the program ultimately determines the speed at which gates are executed, the quantum FLOPS analogy for measuring QPU performance is poorly defined.

Nevertheless, QPU performance can be measured (see Table I). The most widely discussed metric of QPU performance is number of qubits, which in most cases is correlated to the problem size that can be executed on the QPU. An increased number of qubits does not necessarily speed up the performance of an algorithm as the computational complexity of the algorithm may be constant or the increased number of qubits used by an algorithm may not mean a longer sequence of gates is used to execute the algorithm. Thus an increased number of qubits available on a QPU may simply enable moving from fixed-point variables to floating-point variables for results with greater resolution. Alternatively, for quantum algorithms that can be executed in a divide-and-conquer fashion, the number of qubits on a QPU may play a major role in algorithmic performance as fewer executions of sub-problems will need to be executed as the number of qubits, and, consequently, the size of the sub-problems, grows.

Another measure is the relative overhead required by the FT instruction protocols for the ISA. At the scope of the QCU, the cycles per instruction can be extracted. Moreover, the longest duration gate within a QRAM can serve as a worst-case measure of performance, while the shortest gate reflects best time cost. The timing difference between these instructions offers a measure of the spread in QRAM performance on

Table I. Metrics of Performance for a QPU

Metric	Significance
Number of Qubits	Problem size capabilities
FTQEC Qubit Overhead	Number of additional qubits required to implement FTQEC
FTQEC Gate Overhead	Number of additional gates required to implement FTQEC
Clock Cycles Per Gate	Basis for determining quantum computation time
FTQEC Latency	Additional computation time needed to incorporate FTQEC
Number of Gates	Number of gates used to complete circuit
Sample Rate	The lifecycle frequency to obtain a result and reset a circuit to obtain an additional result
Entanglement Latency	Time needed to initially entangle a set of qubits
Entanglement Rate	Time needed to refresh or reestablish entanglement between a set qubits
Intra-QPU Connectivity	The number of qubits a single qubit can entangle with directly
Inter-QPU Connectivity	The number of qubits a single qubit can entangle with indirectly or at a remote site
Teleportation Rate	The rate at which quantum data can be exchanged
ISA Complexity	The diversity of gates that can being executed

any random gate instance. The same measures can be applied to the complete set of QPU instructions to compile a snapshot of worst- and best-case timings.

These definitions do not require reference to a particular technology, as they rely on abstraction of the QRAM and QPU interfaces. This is advantageous for making performance comparisons against different technologies, which may have widely different physics and control signals. Different QPU technologies may also employ vastly different ISAs in a manner that is reminiscent of reduced instruction set computing (RISC) versus complex instruction set computing (CISC) designs. For example, a QPU based on the adiabatic quantum computing model may use a single, time-continuous gate to implement an instruction that requires a significant amount of classical processing overhead to ready the data for quantum processor execution, whereas the same instruction would be realized in the circuit model using a lengthy sequence of discrete primitive gates with minimal classical processing overhead. A comparison between these different QPUs that simply references gate duration and spread forgoes these important details. Additionally, task-specific QPUs will necessitate special-purpose metrics to differentiate between how problems are solved. Restrictions on processor behavior can be useful, however, for purposes of forming comparisons provided that the context is well specified.

An additional consideration for measuring QPU performance is that quantum algorithms are often probabilistic. This introduces the notion of repeated sampling of the readout register in order to collect sufficient statistics for reporting a final result. Depending on the level of instruction abstraction, the programmer may handle this type of sampling, or it may arise within the QPU itself. For those use cases where quantum behavior is intended to be hidden from the user, for example, with high-level languages and libraries, QPU performance will be impacted by these classical pre- and post-processing steps. The use of statistical sampling to derive the final result requires confidence levels that determine the number of required samples and therefore total duration of the program. Neither the instruction or gate metric that we have proposed would measure this aspect of QPU performance. Instead, this becomes an element of benchmarking against certain problem sets and solution goals. Algorithmic complexity statements will offer some guidelines on the scaling of these slowdowns, but experimental tests will be needed to identify the variance in total function performance.

Performance of the quantum interconnect is also a major factor in overall system behavior. The rate at which QPUs establish entangled registers may be initially ignored, since these operations can occur offline from the program execution. However, in a high-performance setting, the rate of entangling and preserving entanglement between nodes represents a potential bottleneck for the system. Idle registers in a QPU require active error correction to mitigate against noise, and any delays in the quantum interconnect will add to this overhead.

It therefore seems very likely that QPUs will need to use some form of network management controller that interacts with the quantum programs wishing to execute using entangled registers. The network manager will be responsible for ensuring availability of entangled registers when requested by a program. This will require coordination among the interconnect, the error-corrected registers, and the program instructions, such that the manager refreshes entanglement between QPUs only when needed by the program. However, faults and latencies in the both the QPUs and the interconnect will complicate these instructions and may eventually lead to communication failure. Therefore, the performance of the interconnect, and especially the entangling operations, is likely to be a major factor in overall system behavior.

5. CONCLUSIONS

As QPUs mature from basic scientific testbeds to robust devices, they will likely be adopted for both application-specific and general-purpose computing. We have investigated several possible abstract architectures for integrating QPUs into HPC systems. We have examined both loose and tight integration designs, which differ primarily in the communication infrastructure and runtime environment needed to host the QPU. The relative performance of each design is expected to depend on how well the quantum algorithm and its programming model offsets the costs of this communication as well as the intended use case.

We have also emphasized that one of the most important aspects of future HPC with QPUs is the quantum interconnect. It has long been appreciated for massively parallel processing systems that the communication backbone between nodes plays a significant role in performance. This has been underscored in recent years with awareness that communication costs may often be bottlenecks in application scaling. It is clear that a quantum interconnect can enhance system functionality by enlarging the set of accessible register states, but it remains unclear if the interconnect would provide a net benefit. This is because the entanglement established between nodes by the interconnect would expose the system to a potentially more serious fault model, in which correlated errors lead to a cascade of failures across QPUs. Appreciable attention has been placed on fault-tolerant ISAs within the context of local computation, and similar techniques for managing distributed QPUs will be an important issue moving forward. Fault models for these architectures and protocols for mitigating against these types of failures are needed.

Existing metrics for conventional computing do not capture all aspects of the QPU behavior, and we have suggested several new features that need to be tracked when tuning system performance (see Table I). This includes the overhead in fault-tolerant ISAs as well as the spread in instruction timings. However, some instructions may be so complex that their performance can only be measured in very restrictive settings, for example, as special-purpose QPUs. The comparison of these metrics with each other offers a quantitative means of assessing the value of QPU-enabled systems, but only when they can be related to existing system metrics, for example, FLOPS, and so on. Putting quantum metrics at the same level of inspections as those of CPU-based measurements will require more detailed execution models for the entire system.

REFERENCES

Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara andKen Brown, Massoud Pedram, and Todd Brun. 2012. *Scaffold: Quantum Programming Language*. Technical Report. Retrieved from ftp://ftp.cs.princeton.edu/techreports/2012/934.pdf

Daniel S. Abrams and Seth Lloyd. 1997. Simulation of many-body fermi systems on a universal quantum computer. *Phys. Rev. Lett.* 79, 13 (Sep. 1997), 2586–2589. DOI:http://dx.doi.org/10.1103/PhysRevLett.79.2586

James Ang, Keren Bergman, Shekhar Borkar, William Carlson, Laura Carrington, George Chiu, Robert Colwell, William Dally, Jack Dongarra, Al Geist, Gary Grider, Rud Haring, Jeffrey Hittinger, Adolfy Hoisie, Dean Klein, Peter Kogge, Richard Lethin, Vivek Sarkar, Robert Schreiber, John Shalf, Thomas Sterling, and Rick Stevens. 2010. *Top Ten Exascale Research Challenges*. Technical Report. DOE ASCAC Subcommittee Report.

Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, Tony Mezzacappa, Parviz Moin, Mike Norman, Robert Rosner, Vivek Sarkar, Andrew Siegel, Fred Streitz, Andy White, and Margaret Wright. 2010. *The Opportunities and Challenges of Exascale Computing*. Technical Report. Summary Report of the Advanced Scientific Computing Advisory Committee Subcommittee.

Bela Bauer, Dave Wecker, Andrew J. Millis, Matthew B. Hastings, and Matthias Troyer. 2016. Hybrid quantum-classical approach to correlated materials. *Phys. Rev. X* 6, 3 (Sep. 2016), 39. DOI:http://link.aps.org/doi/10.1103/PhysRevX.6.031045

A. Broadbent, J. Fitzsimons, and E. Kashefi. 2009. Universal blind quantum computation. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. 517–526. DOI:10.1109/FOCS.2009.36

Jacques Carolan, Christopher Harrold, Chris Sparrow, Enrique Martn-Lpez, Nicholas J. Russell, Joshua W. Silverstone, Peter J. Shadbolt, Nobuyuki Matsuda, Manabu Oguma, Mikitaka Itoh, Graham D. Marshall, Mark G. Thompson, Jonathan C. F. Matthews, Toshikazu Hashimoto, Jeremy L. OBrien, and Anthony Laing. 2015. Universal linear optics. *Science* 349, 6249 (2015), 711–716. DOI:http://dx.doi.org/10.1126/science.aab3642

Andrew M. Childs and Wim van Dam. 2010. Quantum algorithms for algebraic problems. *Rev. Mod. Phys.* 82, 1 (Jan 2010), 1–52. DOI:http://dx.doi.org/10.1103/RevModPhys.82.1

Venkat R. Dasari, Ronald J. Sadlier, Ryan Prout, Brian P. Williams, and Travis S. Humble. 2016. *Programmable Multi-Node Quantum Network Design and Simulation*. Proc. SPIE 9873, Quantum Information and Computation IX, 98730B (May 19, 2016). DOI:10.1117/12.2234697

D. Deutsch. 1985. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. Lond. Ser. A* 400 (July 1985), 97–117. DOI:http://dx.doi.org/10.1098/rspa.1985.0070

M. H. Devoret and R. J. Schoelkopf. 2013. Superconducting circuits for quantum information: An outlook. *Science* 339, 6124 (2013), 1169–1174. DOI:http://dx.doi.org/10.1126/science.1231930

D. P. DiVincenzo. 2000. The physical implementation of quantum computation. *Fortschr. Phys.* 48 (2000), 771783. DOI:http://dx.doi.org/doi: 10.1002/1521-3978

A. Geist and R. Lucas. 2009. Major computer science challenges at exascale. *Int. J. High. Perform. Comput. Appl.* 23 (2009), 427436. Issue 4.

Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'13)*. ACM, New York, NY, 333–342. DOI:http://dx.doi.org/10.1145/2491956.2462177

Charles D. Hill, Eldad Peretz, Samuel J. Hile, Matthew G. House, Martin Fuechsle, Sven Rogge, Michelle Y. Simmons, and Lloyd C. L. Hollenberg. 2015. A surface code quantum computer in silicon. *Sci. Adv.* 1, 9 (2015), e1500707. DOI:http://dx.doi.org/10.1126/sciadv.1500707

J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb, S. J. Pauka, H. Lu, A. C. Gossard, J. D. Watson, G. C. Gardner, S. Fallahi, M. J. Manfra, and D. J. Reilly. 2015. Cryogenic control architecture for large-scale quantum computing. *Phys. Rev. Appl.* 3 (Feb. 2015), 024010. DOI:http://dx.doi.org/10.1103/PhysRevApplied.3.024010

T. S. Humble, A. J. McCaskey, R. S. Bennink, J. J. Billings, E. F. DAzevedo, B. D. Sullivan, C. F. Klymko, and H. Seddiqi. 2014. An integrated programming and development environment for adiabatic quantum optimization. *Comput. Sci. Discov.* 7, 1 (2014), 015006.

M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, and others. 2011. Quantum annealing with manufactured spins. *Nature* 473, 7346 (2011), 194–198.

N. Cody Jones, Rodney Van Meter, Austin G. Fowler, Peter L. McMahon, Jungsang Kim, Thaddeus D. Ladd, and Yoshihisa Yamamoto. 2012. Layered architecture for quantum computing. *Phys. Rev. X* 2, 3 (2012), 031007.

Ivan Kassal, James D. Whitfield, Alejandro Perdomo-Ortiz, Man-Hong Yung, and Alán Aspuru-Guzik. 2011. Simulating chemistry using quantum computers. *Annu. Rev. Phys. Chem.* 62, 1 (2011), 185–207. DOI:http://dx.doi.org/10.1146/annurev-physchem-032210-103512

Volodymyr Kindratenko, George K. Thiruvathukal, and Steven Gottlieb. 2008. High-performance computing applications on novel architectures. *Comput. Sci. Eng.* 10, 6 (2008), 13–15. DOI:http://dx.doi.org/10.1109/MCSE.2008.149

Emmanuel Knill. 1996. *Conventions for Quantum Pseudocode*. Technical Report. Technical Report LAUR-96-2724, Los Alamos National Laboratory.

J. M. Kreula, S. R. Clark, and D. Jaksch. 2015. *A Quantum Coprocessor for Accelerating Simulations of Non-equilibrium many Body Quantum Dynamics*. arXiv:1510.05703 [quant-ph].

Rodney van Meter and Mark Oskin. 2006. Architectural implications of quantum computing technologies. *ACM J. Emerg. Technol. Comput. Syst.* 2, 1 (2006), 31–63.

Tzvetan S. Metodi, Arvin I. Faruque, and Frederic T. Chong. 2011. *Quantum Computing for Computer Architects*, (2nd ed.). Morgan & Claypool Publishers. DOI:http://dx.doi.org/10.2200/S00331ED1V01Y201101CAC013

C. Monroe and J. Kim. 2013. Scaling the ion trap quantum processor. *Science* 339, 6124 (2013), 1164–1169. DOI:http://dx.doi.org/10.1126/science.1231298

Michael A. Nielsen and Isaac L. Chuang. 2000. *Quantum Computation and Quantum Information*. Cambridge University Press.

Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. OBrien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nat. Commun.* 5 (Jul. 2014). DOI:http://dx.doi.org/10.1038/ncomms5213

Kamyar Saeedi, Stephanie Simmons, Jeff Z. Salvail, Phillip Dluhy, Helge Riemann, Nikolai V. Abrosimov, Peter Becker, Hans-Joachim Pohl, John J. L. Morton, and Mike L. W. Thewalt. 2013. Room-temperature quantum bit storage exceeding 39 minutes using ionized donors in silicon-28. *Science* 342, 6160 (2013), 830–833. DOI:http://dx.doi.org/10.1126/science.1239584

Barry I. Schneider. 2015. The impact of heterogeneous computer architectures on computational physics. *Comput. Sci. Eng.* 17, 2 (Mar 2015), 9–13. DOI:http://dx.doi.org/10.1109/MCSE.2015.39

Peter Selinger. 2004. Towards a quantum programming language. *Math. Struct. Comput. Sci.* 14 (8 2004), 527–586. Issue 04. DOI:http://dx.doi.org/10.1017/S0960129504004256

Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509. DOI:http://dx.doi.org/10.1137/S0097539795293172

Daniel R. Simon. 1997. On the power of quantum computation. *SIAM J. Comput.* 26, 5 (1997), 1474–1483. DOI:http://dx.doi.org/S0097539796298637

Darshan D. Thaker, Tzvetan S. Metodi, Andrew W. Cross, Isaac L. Chuang, and Frederic T. Chong. 2006. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. *SIGARCH Comput. Archit. News* 34, 2 (May 2006), 378–390. DOI:http://dx.doi.org/10.1145/1150019.1136518

Rodney Van Meter, Thaddeus D. Ladd, Austin G. Fowler, and Yoshihisa Yamamoto. 2010. Distributed quantum computation architecture using semiconductor nanophotonics. *Int. J. Quant. Inf.* 8, 01n02 (2010), 295–323.

Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal. 2001. Efficient load balancing for wide-area divide-and-conquer applications. *SIGPLAN Not.* 36, 7 (Jun. 2001), 34–43. DOI:http://dx.doi.org/10.1145/568014.379563

Dave Wecker and Krysta M. Svore. 2014. *LIQUID: A Software Design Architecture and Domain-Specific Language for Quantum Computing*. Retrieved from http://arxiv.org/pdf/1402.4467v1.pdf.

M.-H. Yung, J. Casanova, A. Mezzacapo, J. McClean, L. Lamata, A. Aspuru-Guzik, and E. Solano. 2014. From transistors to trapped-ion computers for quantum chemistry. *Sci. Rep.* 4, 3589 (2014). DOI:http://dx.doi.org/doi:10.1038/srep03589

Received November 2015; revised September 2016; accepted October 2016