QED-C®

Practical Intermediate Representation for Quantum (PIRQ)

Requirements and Near-Term Recommendations

February 2022

Acknowledgments

The contents of this report would not have been possible without the months-long diligent and collaborative efforts from the organizing committee:

Yudong Cao (PIRQ Chair, Zapata Computing)
Krystal Bouverot (SRI International)
Alex Chernoguzov (Honeywell)
Bettina Heim (Microsoft) Alan Ho (Google)
Sean Howell (Q-CTRL) Sonika Johri (IonQ)
Tom Lubinski (Quantum Circuit Inc.)
Alex McCaskey (ORNL)
Celia Merzbacher (SRI International)
Jason Necaise (D-Wave)
Bálint Pató (Google) Vic Putz (QC Ware)
Steve Reinhardt (Quantum Computing Inc.)
David Rivas (Rigetti)
Shengtao Wang (QuEra)

Special thanks also go to all who participated in the workshop, discussed requirements and contributed to the recommendations captured in this report.

Table of Contents

Executive Summary

With the emerging landscape of quantum hardware as well as quantum programming paradigms, a quantum programmer soon faces the challenge of executing general quantum programs on a variety of quantum backends in an automated, flexible, and consistent manner. A pressing question for the programmer then becomes **how to write the program once and run it on all quantum backends.** The challenge from a hardware provider point of view is equally acute, with a diverse landscape of quantum programming approaches and compilation tools. The question from the provider's perspective is **how to run all programs that are written, regardless of how they are written.**

In the classical programming world such questions around "write once, run all" and "run all that are written" also emerged in the historical evolution of computing. After all, for ten different programming languages and three different hardware processor backends, it is inefficient to produce 30 distinct compilers for each combination of programming language and hardware backend. Thus, the concept of intermediate representation (IR) developed. Roughly speaking, IR is a representation of a program that is independent of its source language and the hardware on which it is executed. Contemporary designs such as low-level virtual machine (LLVM) have become a great tool for standardizing the compilation pipeline. LLVM is set of compiler and toolchain technologies, which can be used to develop a front end for any programming language and a back end for any instruction set architecture.

This report seeks to inform a set of requirements for practical IR for supporting the needs of quantum programming in the near-term while being compatible with long-term development plans. The report includes:

- An exploration of current solutions and proposals for quantum IR.
- Ideas about IR at different levels of abstraction for different quantum computing use cases.
- Recommendations for near-term actionable milestones for the community that address the two challenges of "write once run all" and "run all that are written."

This report bridges between the classical compiler and quantum computing approaches to IR and is based in part on a workshop in July 2021 that convened experts from academia, industry, national labs, and government agencies.

Recommendations

1. Conduct an architecture study to identify required IR functionalities

Conduct an architecture study for clarifying the functionalities required for IR. The goal is to clarify the relationship between the IRs and compiler toolchains that currently exist. Such a study will help the community to agree on various levels of abstractions. Such study will also help consolidate languages, both at a low level and a high level. It is worth seeing how such architecture study impacts the process of defining the high-level and mid-level IRs for declarative quantum programming.

2. Determine reference implementations for specific quantum computing use cases

Collect concrete reference implementations for specific quantum use cases. Such reference implementations may be contributed by players across the industry and used to consolidate various IR proposals. IR requirements should be framed according to different types of use cases, with the low-level IR requirements driven by specific quantum use cases. The reference implementations will also help shed light on the priorities for incorporating the currently exotic hardware features, such as measurements beyond 2-level systems and qubit rearrangement.

3. Define abstract models for quantum computing hardware

Define appropriate abstract machine models for quantum hardware backend. This has been proposed for imperative paradigms focusing on gate-model quantum computing as well as for declarative paradigms including analog simulators and quantum annealers. There is a related need for a standard representation of hardware backend. Such machine models will facilitate the development of systems supporting a full compiler IR for backend integration, as outlined above.

4. Collaborate through QED-C

Form one or more dedicated work group(s) for specific directions that contribute to the recommendations in this report. The progress made in the work group(s) may provide the basis for a follow-up PIRQ workshop as well as the called-for industry-wide IR consolidation process. In addition, the PIRQ working group can also benefit from other efforts such as QIR Alliance under Linux Foundation which are focusing on pushing the state-of-the-art compilation techniques for quantum computing. The working group may serve as a testing ground for new developments coming from those efforts and in return, act as a platform for providing user feedback.

Introduction

The field of quantum computing is maturing as an engineering discipline. With this growth, various proposals have surfaced for building the associated software infrastructure for quantum programming in an efficient and flexible manner. A critical component is the pipeline that starts with the user's source code as input and terminates with quantum-classical, machine-readable instructions as the output (figure 1).

There is a broad range of quantum computing hardware available to users today with different backend architectures that may need to be supported. Users develop quantum programs in diverse classical programming language environments with fundamentally different paradigms (for example, languages such as Julia for scientific computing). An important question is how to design a low-level software infrastructure that is scalable with respect to these multiple target environments?

Classical compiler infrastructure typically incorporates an intermediate representation (IR) that is precisely intended for handling the multitude of both programming languages and hardware architectures. The IR provides a flexible data model that enables code optimization, transpilation to target backend hardware, and generally enhances the portability of a computer program. In programming a quantum computer, the question naturally follows: how to design a practical IR that incorporates both quantum and classical instructions? This question opens up an entire design space encompassing many tradeoffs that warrant discussion (between complexity and generality, near-term gain versus long-term consideration, for example).

QED-C convened a workshop in July 2021 with an objective to articulate a set of practical requirements for an IR that satisfies the near-term needs of the quantum community. Participants were presented with in-depth technical overviews and demonstrations of several current approaches to implementing a useful IR, providing background on techniques that have been explored. This was followed by a series of presentations on various use cases that could potentially be addressed by an IR that is well-designed.

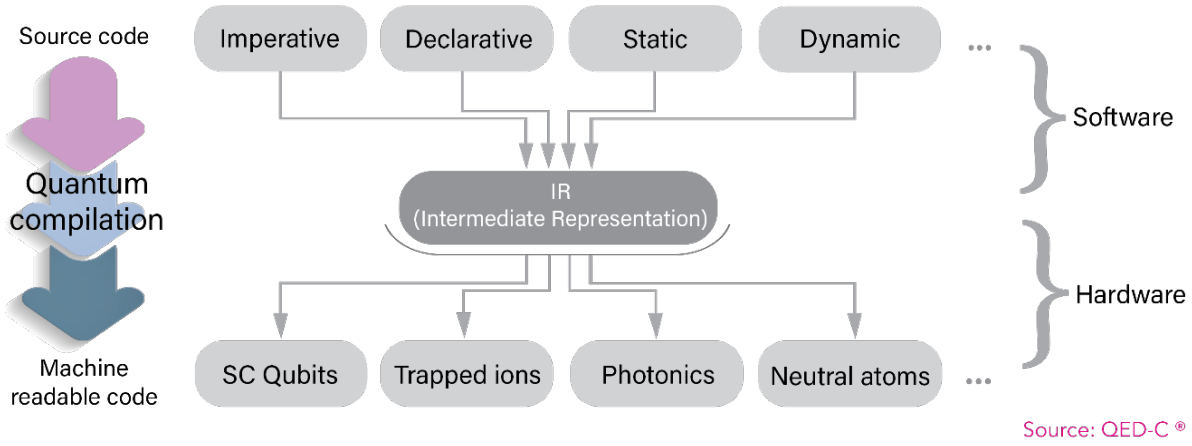

Figure 1: The role of IR in the compilation

The focus of the workshop was *not* to design concrete IR constructions or compiler solutions, but instead to provide an inclusive forum to gather participants from a diverse range of expertise that may have different notions of what an IR needs to support. See Appendix A for the detailed agenda of the workshop and Appendix B for detail on the structure of breakout discussions.

The need for practical IR can be appreciated in the context of the software development life cycle (figure 2). The development of an IR for hybrid quantum/classical kernels can also be considered in a similar light, where the very first step is to articulate a set of requirements. This report is intended to serve as a resource for the quantum community. For potential users of the IR, this report is intended to boost a sense of assurance that their needs are captured in an inclusive forum. For designers and builders of IR, this report can provide early information about what *actually* matters to the potential users. Since this report does not present concrete solutions or designs, it is largely left to the quantum community to develop IR solutions that meet certain requirements. The below requirements should provide useful guidance for the quantum community to share their efforts in IR design and development.

To further illustrate the potential benefits of PIRQ, consider a few scenarios where a lack of clarity in IR requirements can lead to unsatisfactory circumstances. Suppose Alice wants to run a quantum circuit on Bob's quantum computer:

- *Scenario 1: IR with a fixed gate set* (e.g., CNOT + single-qubit rotations). Alice compiles x natively for Bob. But the IR insists that the circuit be in a specific gate set. So the circuit from Alice gets compiled in the IR gate set first and then re- compiled back into the native gate set for Bob, leading to an overhead compared with direct execution.
- *Scenario 2: IR with limited classical operations*. For executing iterative quantum-classical algorithms such as VQE/QAOA efficiently, Alice can only send one set of circuits at a time to Bob, unless Alice sends Bob the source code directly. This severely impairs the efficiency of the algorithms.
- *Scenario 3: IR with limited software support*. Bob is a new hardware company with physical device capabilities for pulse control and gate operations; Alice wants to run a circuit on Bob's computer, but Bob doesn't have the software needed to run the IR.

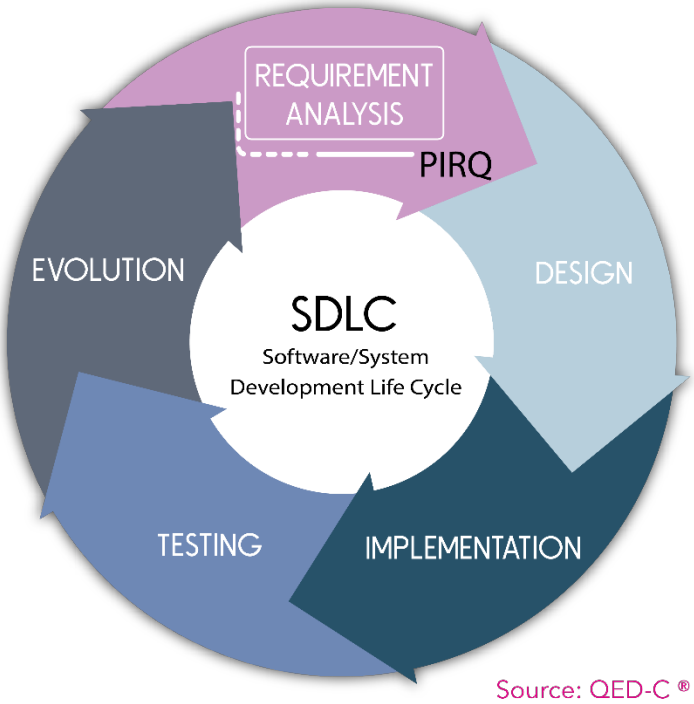

Figure 2: The emphasis of practical IR from the perspective of software development life cycle.

- *Scenario 4: IR with ambiguous technical assumptions*. Alice compiles a program from toolkit *X* to the IR for Bob's computer using architecture *Y*. The program compiles and executes but since *I*, the IR, and *Y* have different assumptions on bit ordering, angle measurement, and other communicative parameters, the results at best require extensive rework and at worst are useless.

These scenarios are examples of circumstances that can arise in the near term, which motivates the urgency of articulating and addressing the IR requirements.

Framework

The framework presented in table 1 helps to frame the requirements for IR. In the classical compilation pipeline, an intermediate representation typically undergoes a sequence of optimization steps, each of which produces an IR that is transformed from the previous step. In the process of executing quantum operations, different levels of representations may apply. For each level, there is a specific set of information that the IR needs to capture. The specific combination of the levels involved in a quantum compilation process also depends on the model of quantum computing and sometimes the specific paradigm within the model (see Table 1, where each column is for a general class of use cases of IR for quantum).

Table 1: Different levels of the IR for various quantum computing use cases.

		Gate model quantum computation			Gate-free	Quantum annealer	Analog simulator
		Quantum simulation (static, dynamic)	General quantum algorithms	Hardware-efficient ansatzes			
Level 1	1.1: Problem Hamiltonian description	✓				✓	
	1.2: Embedded Hamiltonian on hardware					✓	✓
Level 2	2.1: Quantum program description (classical control flow, classical operations, quantum operations, oracle calls)	✓	✓				
	2.2: Hardware ready quantum program (transpiled quantum operations)	✓	✓	✓			
Level 3	3.1: Error corrected gate implementations	✓	✓				
	3.2: Physical evolution to be realized on the quantum device	✓	✓	✓	✓	✓	✓

Level 1.1: Problem Hamiltonian description. This level provides descriptions for computational problems that can be expressed as a problem of finding the ground state of a "problem Hamiltonian." Here the problem Hamiltonian that the IR describes can be any *k*-local operator, and it may even go beyond a two-spin (binary) model. It does not necessarily conform to any hardware restrictions.

Level 1.2 Embedded Hamiltonian on hardware. At this level, the IR describes a Hamiltonian that conforms to the specifications of a particular physical backend. For example, the Hamiltonian may be restricted to pairwise qubit interactions with coupling coefficients represented with finite precision in a finite interval. The particles in the Hamiltonian can be qubits, spins, or elementary particles such as bosons and fermions.

Level 2.1 Quantum program description. At this level, the IR describes a sequence of operations, be it quantum or classical, and control flows (such as conditional branching, jumping and looping). The quantum operations do not have to conform to the restriction of any quantum hardware backend.

Level 2.2 Hardware-ready quantum program. At this level, the IR should be such that the quantum operations conform to the specifications of a given hardware backend in terms of gate set, qubit connectivity, and so on, making it compatible for executing at the hardware backend. Note that while this is useful to talk about as a separate phase of compilation, this level is relative to the hardware platform: one program expressed at level 2.2 might be considered as a level 2.1 for a different hardware platform if, for example, the qubit topology or native gate set is different.

Level 3.1 Error corrected gate implementations. At this level, the IR describes the details of how each quantum gate operation is to be executed on an error-corrected backend. Here one assumes that the gate set is Clifford and T gates.

Level 3.2 Physical evolution to be realized on the quantum device. At this level, the IR specifies directly the time-dependent waveform of various terms in the Hamiltonian of the quantum device.

Use case: Gate model quantum computation. For the IR discussion, there are three distinct classes of use cases that involve different sets of levels:

Quantum simulation. The goal is to simulate a physical system described by some Hamiltonian. There are two types of simulations.

1. Static simulations are concerned with finding the properties of the eigenstate(s) of the Hamiltonian. In this case, the minimal amount of information needed to specify a static simulation instance is by describing the Hamiltonian and leaving the rest to the compiler. The user may optionally wish to select the eigenstate (e.g., ground or excited), the method for extracting the properties of the state (e.g., VQE, QAOA, phase estimation), the circuit or ansatz for implementing the method, the optimizer (if any) used for training the circuit. However, at a minimum, the user only needs to provide information about the Hamiltonian with an IR at Level 1.1.

2. Dynamic simulations are concerned with realizing the time evolution under the Hamiltonian. Here the same principle applies: the user only needs to specify the Hamiltonian with an IR at Level 1.1 at a minimum. Here, algorithms for time evolution that rely on oracular calls such as those for supplying matrix elements of the Hamiltonian (such as https://arxiv.org/abs/0910.4157) or supplying a Hermitian operator as a block embedded in

a unitary operator (such as https://arxiv.org/abs/1610.06546) are excluded from this class. They will be included in the next class, general quantum algorithms.

General quantum algorithms. This class of use cases captures a majority of quantum algorithms devised in the standard gate model of quantum computation. In addition to quantum algorithms that consists of a sequence of unitary operations with measurements occurring only at the end, there are the following classes of quantum algorithms that are worth pointing out:

1. Variational quantum algorithms with only end-of-circuit measurements and iterative classical updates of circuit parameters. This includes many of the familiar paradigms such as variational quantum eigensolvers (VQE), quantum approximate optimization algorithms (QAOA), as well as some of the quantum machine learning algorithms (for instance https://arxiv.org/abs/1804.00633).

2. Quantum algorithms requiring mid-circuit measurement or qubit resetting. Examples include repeat-until-success circuits (see for instance https://arxiv.org/abs/1404.5320), quantum simulated annealing (see, for instance, https://arxiv.org/abs/0712.1008), and iterative phase estimation (see, for instance, Section 13.5 in *Classical and Quantum Computation* by A. Kitaev, A. Shen and M. Vyalyi). Here, it is essential that at least a subsystem of the quantum device retain quantum coherence while other parts are undergoing classical computation, quantum measurement, or resetting processes. Specific requirements are placed on the timeframe in which one executes the classical controls.

3. Quantum algorithms containing calls to oracles. There is an abundance of examples in this category, with prominent ones including the Grover's algorithm and the HHL algorithm. Another large collection of algorithms in this class is quantum simulation, as mentioned in the previous section.

Because all instances of algorithms in this class can be expressed as a sequence of quantum or classical operations punctuated by control flow operations such as conditional branching and looping and jumping, an IR at Level 2.1 should be able to capture the most basic set of information needed to execute the instances on concrete hardware backend.

Hardware efficient ansatzes. Here one intends to fully take advantage of the native gate set of a quantum device by treating it as a tunable state preparation and measurement apparatus. Prominent examples include variational quantum eigensolvers (https://arxiv.org/abs/1704.05018) and other variational algorithms of similar nature (see https://arxiv.org/abs/2012.09265), as well as hybrid algorithms based on variational principles (see https://arxiv.org/abs/1812.08767). In these cases, one only needs an IR at Level 2.2 since there is no need for circuit compilation.

Use case: Gate-free approaches. It has been demonstrated that by thinking outside the gate model paradigm and instead considering quantum computing processes as controlled quantum evolutions, one can reap considerable benefits in the number of quantum resources needed for solving certain problems (see, for example, https://arxiv.org/abs/2008.04302 for an illustration of such benefit for VQE). In this case, one only needs an IR at Level 3.2 where time dependence of driving terms in the Hamiltonian of the quantum device is described.

Use case: Quantum annealers. Users of these annealers program in a declarative fashion by specifying the constraints between the variables involved in the problem using a Hamiltonian. The ground state of

the Hamiltonian encodes the desired computation, which includes its output. To support such declarative programming, the minimal set of information needed is the description of the Hamiltonian, which can be captured by an IR at Level 1.1. It is also possible to directly encode a computation with an IR at Level 1.2 if the user would like to perform the embedding of the problem Hamiltonian onto the connectivity of a given hardware backend. In the cases where the user desires to directly manipulate the low-level physical evolution of the device, such as the annealing schedule of a quantum annealer, an IR at Level 3.2 will be used.

Use case: Analog quantum simulators (See presentation here and a review, for example). The gate-based approaches inevitably incur overhead when compiled on specific hardware. For a number of different quantum hardware platforms, it is possible to go to a much larger scale in terms of the number of qubits and the time scale of the dynamics if one uses them as analog quantum simulators exploiting the native Hamiltonian of the hardware. For these analog quantum simulation use cases, the quantum system is not necessarily fully programmable, so one does not need to use an IR as general as that on Level 1.1, but instead adopt an IR on Level 1.2 which maps closely to the physical system. Within the scope of the tunability of the system, control signals can also be represented using an IR on Level 3.2.

Analog quantum simulation often finds applications in condensed matter physics, high-energy physics, and material sciences. In recent years, both the scale and programmability of various quantum simulator platforms have increased dramatically, such as technologies based on neutral atoms in optical tweezers (e.g., https://arxiv.org/pdf/2012.12281.pdf and https://arxiv.org/abs/2104.04119), neutral atoms in optical lattices (e.g., https://arxiv.org/abs/1805.09819 and https://arxiv.org/abs/1810.03584), and trapped ions (e.g., https://arxiv.org/abs/1912.07845). Users of these analog quantum simulators may specify the (time-dependent) parameters of the native Hamiltonian for the specific hardware using an IR on level 1.2. The underlying physical particles can be represented by qubits, spins, bosons, or fermions.

Requirements for practical IR for quantum computing

This chapter reviews the requirements by level and application, as shown in figure 3.

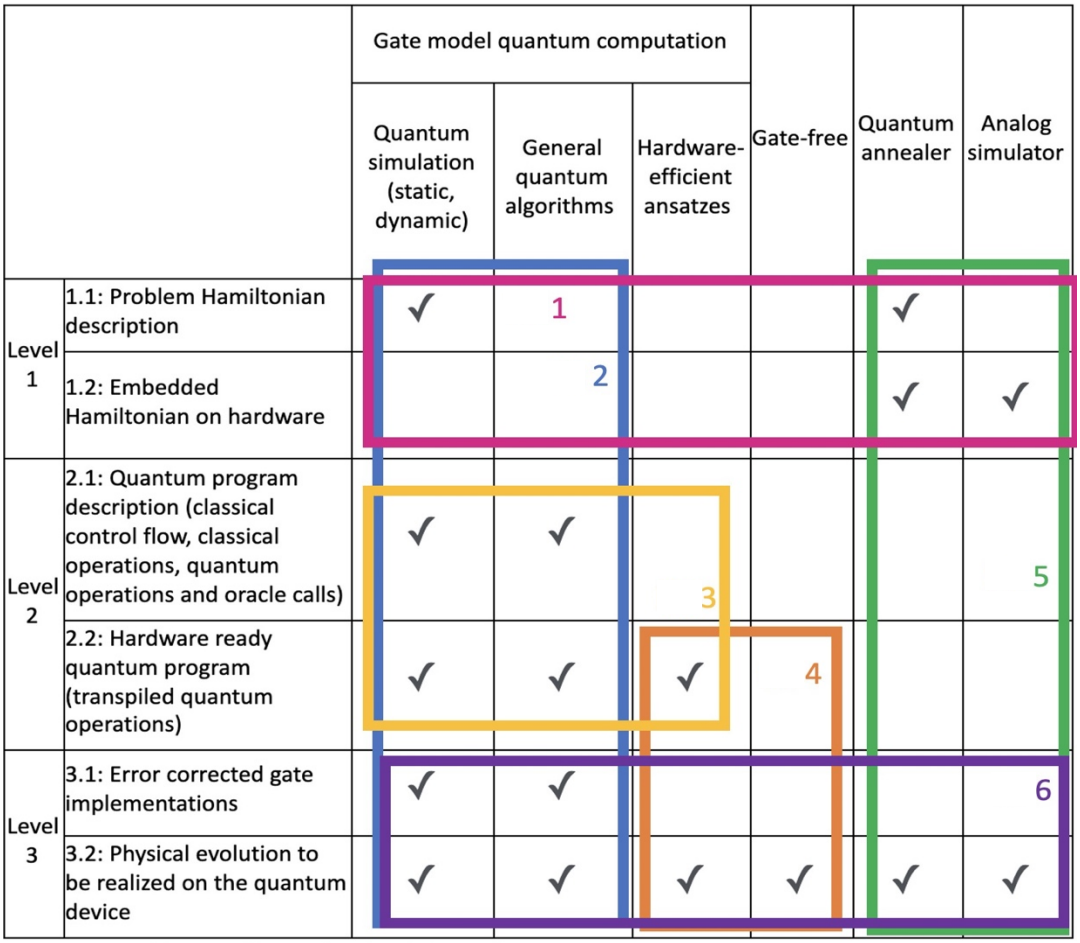

Figure 3: Levels and applications relevant to practical IR for quantum.

1. Declarative quantum programming

Declarative quantum programming is concerned with the declarative aspect of quantum programming, where a program is described by "what needs to be done" and leaves the "how" to the compilers and the hardware backend. A typical example is constraint programming, where a program only specifies the constraints among a set of variables and leaves it to the compiler to generate an assignment of the constraints that maximally satisfy those constraints. Such a declarative model is also echoed in quantum annealing and analog quantum simulation paradigms.

Requirements already fulfilled by existing IR proposals

A declarative API could be great for computing practitioners who do not want to learn about quantum computing in detail, which will be the overwhelming proportion of quantum computing (QC) users even

if not the overwhelming proportion of initial QC users. *Declarative programming* could be broadened from its usual definition to include side-effect-free computation that is independent of instruction order. The OpenFermion API is considered a declarative API, though it is specific to the domain of quantum chemistry. The D-Wave QUBO/Ising model API is also considered a declarative API, though it is specific to D-Wave hardware among quantum processors (QPUs) and a particular Hamiltonian.

A declarative API will be required that may be separate from any underlying IR, though it would use underlying IRs. There will need to be multiple levels of IRs under a declarative API.

The declarative APIs (OpenFermion and Ocean) could meet the goal of portability across processor architectures and types but do not currently. Focusing more explicitly on the Hamiltonian as the portable construct would be valuable. *Portability* is required across different circuit-model, measurement-based, quantum-annealing, and topological implementations.

While the use of low level virtual machine (LLVM) by three of the Session 1 presentations is encouraging in some ways, risk arises due to a) it not being clear that LLVM is a good match for quantum compiler needs, b) the lack of coordination among those efforts, and c) it being a large development burden.

Requirements that need to be fulfilled

Qubit topology, noise, and errors will all be important to handle for delivering strong performance. All of these issues should not be visible in a declarative API, with the possible exception of via pragmas, but that means they need to be handled effectively by the IRs at lower levels, which is beyond today's state of the art.

Declarative API/IR users need hardware notions (e.g., qubit architecture and topology, representation of variable types, constraints) to be abstracted away and handled effectively by IR(s) at lower levels.

Similarly, declarative API/IR users need the details of algorithms like variational algorithms to be abstracted away, as QC experts may understand these, but most QC users will not. Such algorithms need to be connected by IR(s) and tools to construct addressing concepts that QC users already use in their reasoning.

Challenges to adoption

Qubit topology, noise handling, and error correction will all be important in delivering strong performance from NISQ and error-corrected QPUs. It will be difficult to represent those in a high-level declarative API/IR without the risk of "abstraction leakage."

Topological QPUs will be quite different from others and handling topological and non-topological QPUs via the same IR will be difficult.

Open questions and challenges

Balancing the need for a high level of abstraction with the need to deliver excellent performance with early QCs (even deferring the need for portability across physical gate-model implementations) will be difficult.

Near-term opportunities

The following constitute useful intermediate steps:

At a medium level in the IR hierarchy, classes of problems (e.g., QUBO/Ising model) that have a natural mapping to one or more quantum algorithms should be found and thus bridge from classical to quantum. This is similar to the way that MILP, conic programming, Prolog, and SAT expressions can be mapped from various problems and mapped to various solvers.

At a high level in the IR hierarchy, declarative languages that internally map to QCs are the recommended approach. Optimization and constrained-optimization problems are often specified declaratively already in the classical domain. A "quantum SMT" may be valuable as an implicit way to use a QC in searching. As OpenFermion already illustrates, mapping a domain-specific language to a mid-level IR could yield excellent results for that domain.

2. Gate model vertical solution

This concept involves executing highly structured quantum algorithms, which covers most quantum algorithms discovered to date. "If we were to execute an instance of an algorithm *X*, what information should the IR maintain throughout the levels?"

This section considers the role of an IR in addressing the challenges of vertical application development within the quantum computing landscape. The overarching idea that emerged is that there is a *layering process* involved. There is actually a hierarchy of IRs, starting with abstract IRs that are defined close to the source model. This pinches down to target-independent IRs, then to target-specific IRs. This can be described by the "funnel" diagram below (figure 4). At the bottom of the funnel, the IRs become target-specific. It is critical to support the idea of composition, i.e., the ability to include lower-level IR into the higher levels of IR. Figure 4 shows three levels of existing and future IRs: problem-specific, problem and target agnostic, and target specific.

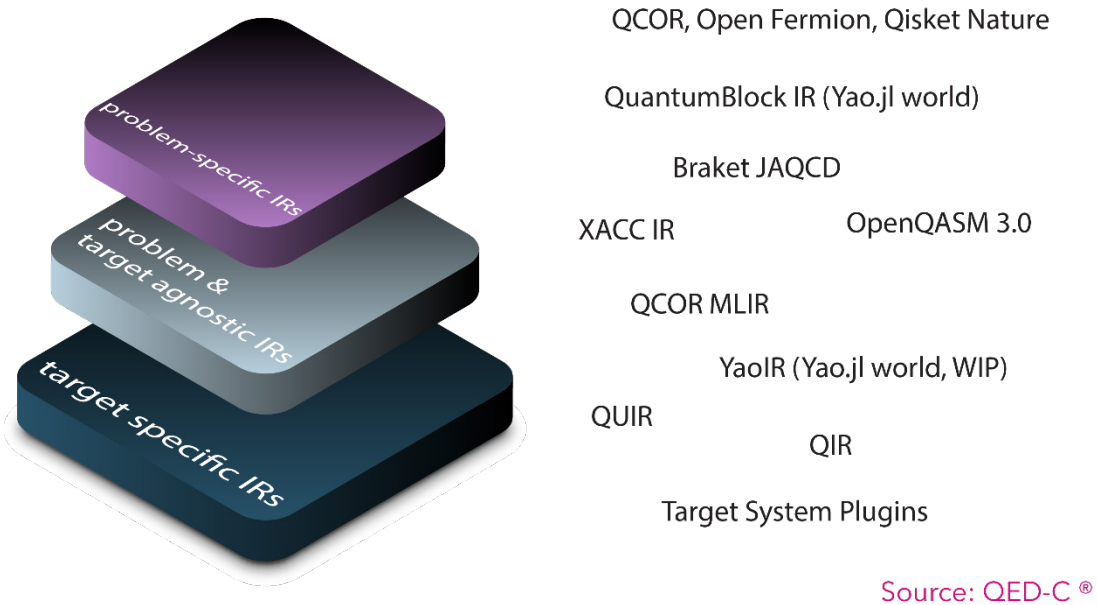

Figure 4: The 3 levels of existing and future IRs.

One key requirement is the need for hierarchical or multi-level intermediate representations. Different analysis and transformation strategies may be present at varying levels of IR abstraction. Providing an IR stack that progressively lowers from one representation to another will prove extremely useful in enabling developers to contribute transformations and operations at levels that best work for the situation at hand. Multi-Level IR (MLIR) is a key player here, in that it directly enables the design of IR extensions for the desired level of abstraction. Moreover, it provides robust mechanisms for lowering one representation to another.

Further study is required on an intermediate representation or for concrete projects that may emerge. Researchers today need to be able to punch through abstractions and have available for the concrete implementation of tools to facilitate the utilization of all the quantum computing resources currently available. Different communities will express their problems in problem-specific IRs. These need to be translated down to a problem-independent, target-independent level and then compiled down to a target-specific representation for execution.

There is consensus among many members of the community that multiple IRs are here to stay. These are in wide use and active development by the community. Our efforts should be focused on exploring the common terminology that all IRs share. There is an opportunity to study ways to enhance communication between the IRs toward interoperability. This would include an analysis of the techniques needed to maximize the coverage of the tools that are developed to perform conversions between the IRs.

Level 1: Problem-Specific IRs - The Problem Description

There are multiple problem- specific requirements for an IR. For example, there may need to be different IRs for problems centered around Hamiltonian Representation. Other problems may be better suited for an IR tailored for variational algorithms or classically centered algorithms like Shor's.

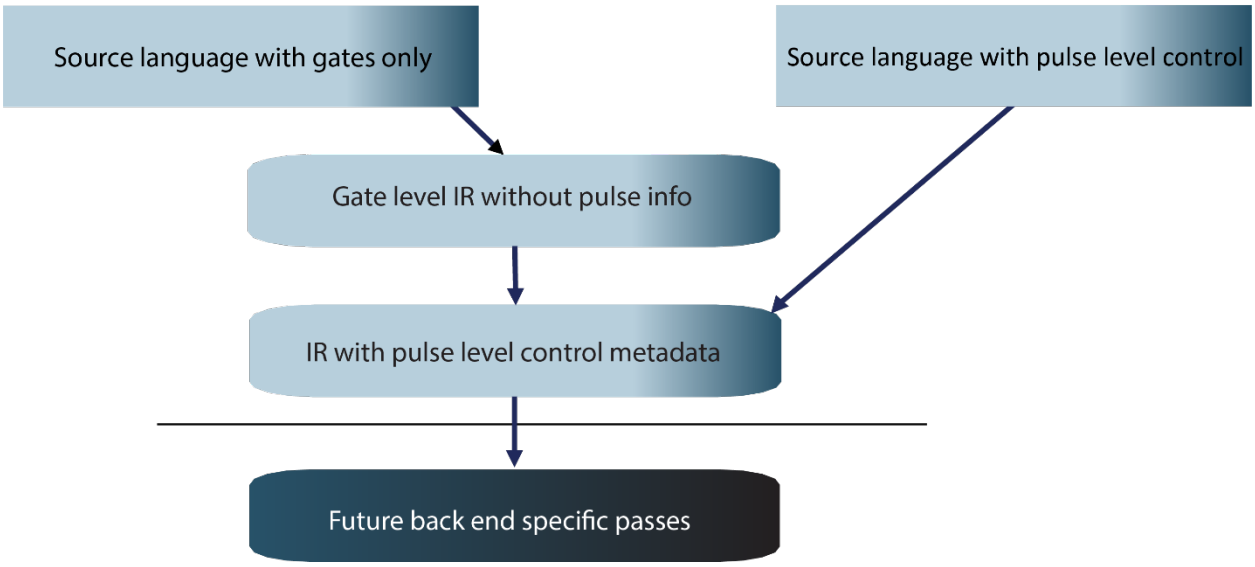

Figure 5: How gate-level and pulse-level descriptions can enter at different levels of IR.

At this level, one needs to avoid domain-specific nodes in the IR. This will adversely impact the flexibility and extensibility of the IR or force constant extension as new domains are considered. High-level problem descriptions may be best served as runtime-level data structures or libraries, as opposed to a native aspect of the compiler IR. Recently published work has put forward an IR for Pauli operation representations (https://arxiv.org/pdf/2109.03371.pdf), while other work has kept this at the library level for usage by programmers at runtime (https://arxiv.org/abs/1909.02457).

These types of high-level domain-specific problem descriptions are expressed in runtime libraries in projects like QCOR, Qiskit Nature, and OpenFermion. There is further need for nodes in the IR that enable typical quantum code synthesis. Things like a trotterization primitive (QCOR $e^{i\theta}$ instruction) will prove extremely useful and span several domains.

Level 2: Problem and Target-Agnostic IRs - Quantum-Classical Description

There is a need to generically represent circuits, gates, qubits, the full gate model, and to define custom gates (with pluggable calibrations). It is important to be able to specify parameters to enable optimized execution and to minimize latency of execution. Compilers should be able to take target calibrations as metadata. With regard to building a quantum error correctable system, it should be possible to express a full error correction scheme including classical control, feed forward, encoding, decoding, and so on.

It will be critical to consider the entire quantum-classical machine model in the development of a robust IR for quantum computing. This will include CPUs, FPGAs, ASICs, and the QPU itself. The interaction of these components and the mechanisms for transforming the IR to suit the machine model will be key.

One may need to consider parallelism and classical thread execution at the low levels of interaction between host CPU and QPU. Co-routines may play a role and at the lowest levels HDL may provide a potential programming paradigm. Leveraging the LLVM may prove extremely powerful in the context of quantum computing. The LLVM ecosystem has been in development for nearly 20 years and provides robust tooling for IR expression, transformation, linking, code generation, and so on. Instead of reinventing the wheel, there is a research opportunity to leverage these tools in new ways for available quantum coprocessors. The community should leverage the framework as is and avoid extending it with new instructions (a notoriously difficult thing to do in LLVM). Finally, in the development of an IR, one should avoid developing a monolithic system. Instead, there is an opportunity to focus on modularity, developing components with well-defined interfaces, and enabling developers to plug-and-play various components for tailored compilation workflows.

Level 3: Target-Specific IRs - Low-level EC and Physical Evolution

With regard to target-specific requirements, many important factors need to be considered. These include an awareness of qubit connectivity characteristics to enable optimal compilation and execution. Similarly, users will need access to pulse-level control of the quantum hardware, especially in the early stages of the development of quantum computing (figure 5). It is also essential that the target-specific IR has the ability to represent fault-tolerant computational primitives to define quantum error correction protocols. Additional areas to consider include surface code, lattice surgery, and braiding patterns that may be enabled on some hardware targets.

At the lowest levels, the IR should be able to express *synchronicity and timing*, as well as signal representations. Once again, this is viewed in the context of a multi-level IR, and higher levels should be

progressively lowered to some physical analog level of abstraction. Key concepts at this level for the quantum-classical machine model are typically analog signals of some type and electronic control system invocations. Therefore, *retaining timing information* in the IR representation itself will prove critical, as these signals are composed and created at specific runtime points. Programmers' needs should be accommodated at varying levels of abstraction. The IR should enable one to develop programming approaches that let one define code at the high level, but also at the physical analog level as well.

Summary and opportunities

The current IRs (e.g., OpenQASM/QUIR**,** MSFT QIR, XACC/QCOR) are sufficient at the moment. These are in use by some portion of the user community. There are active integration projects underway, and this process should be encouraged.

The goal is to steer the efforts of independent, invested parties toward a common goal without precisely dictating a desired outcome. As the efforts proceed, the community will need to be mindful of the evolution of quantum hardware in areas such as error correction.

These observations suggest the following opportunities to develop a practical IR for quantum computing:

The community could define the target quantum-classical abstract machine model at scale. A number of language and IR approaches start from this perspective and define the abstract machine the programmer is targeting. This directly influences the semantics of the IR and programming approach and could streamline further requirements gathering and use case modeling for a robust quantum IR. There is need to discuss both logical and physical program representations (high- and low-level IR abstractions) and different user types (low- and high-level programmers). The *enumeration of machine model profiles* may prove critical as well. These profiles would describe what the target hardware is capable of and ultimately dictate what the compiler-generated IR can do. The enforcement of these profiles will prove critical, and compiler engineers can follow automated translation pathways or robust diagnostics for error reporting.

A "PIRQ v2 workshop" is recommended to study and determine whether interoperability would be valuable across the existing IRs. A workshop could also attempt to identify valuable collaborative efforts among QED-C members that will help achieve interoperability between the major IR proposals.

3. Gate model focusing on circuits

Quantum circuit representations are a central object of current quantum programming practice.

Requirements already fulfilled by existing IR proposals

The current state of quantum programming is largely ad hoc. Several languages have been developed by the industry to allow constructing quantum programs (circuits). With the growing abilities of QPUs, there is a growing need to define hybrid quantum-classical programming models. The programming model needs to be supported by the tools and be built on an interoperable foundation.

In some ways, some IR and compiler efforts are far ahead of the "higher-level" toolkits, which could eventually be compiled down to them such as Qiskit, Cirq, et al. The state of current implementations of

IR that handle compilation to various targets, admixture of classical and quantum operations, control flow, etc. are a good start. Indeed, such implementations cover almost all of the basics for a (circuit-based) quantum IR: they encode gate-based operations cleanly and unambiguously, have provision for measuring quantum state and returning it, and can mix classical control-flow and operations with quantum operations.

A layered approach to intermediate representations may be very valuable, much as the layer model was extraordinarily successful for modeling network communications. Several such layers were proposed, from a very high level, "pure gate" approach assuming perfect hardware to layers involving specified automatic resource allocation/optimization, hints for allocation/optimization, CPU/FPGA machine code, down to microcodes and pulse levels. There may not be a single "common IR" (although the runaway success of IRs such as LLVM, the Java Runtime Environment, or the Common Language Runtime suggest that such a layer may well be feasible, with layers above compiling down to it and layers below depending on it).

It would be beneficial to have a target-agnostic high-level layer where allocations would be compiled away, whereas in lower levels such decisions would increasingly be "baked in" to the code. A hardware-agnostic quantum IR can be produced by language-specific compilers, before being consumed by subsequent compilation stages and tools in the pipeline that apply optimization passes and transformation passes to ultimately convert it into a backend-specific format including the set of control system commands required to execute it on a QPU. Ordinarily an IR is viewed as a depiction of the upper layers' intent (in other words, a codified form of an executable program). However, it is important to consider how to expose details of the devices— including hardware features and control features—to create an "abstract model" of a quantum device that would inform upper layers of a compilation stack.

Multi-level quantum IRs will enable multiple ecosystem vendors to cooperate on building quality tools focusing on front-end language development, mid-level optimizations, and low-level QPU-specific adaptations. For example, the XACC IR can be used, to an extent, for interoperability between toolkits. Portions of a program can be generated from various QC toolkits, compiled into the IR, and used by other modules of the same program. Several quantum IR proposals that could evolve into an IR representation at different stages have been introduced by various industry players and are under development. Some of the representations more closely resemble a traditional IR built on LLVM technology foundation. Others are promoting the languages developed to program specific QPU hardware.

Requirements that need to be fulfilled

If a multi-level IR strategy is adopted, the interfaces between IR levels need to be clearly specified. The scope of an IR at each level needs to be defined to avoid overlaps or gaps. While specifying boundaries between IR levels, a balance needs to be struck between completeness and the ability to extend to future architectures.

To be useful, any quantum IR needs to be semantically well-defined. Quantum gate operations need to be unambiguous and directly mapped into their mathematical representation. Semantic meaning is key to a multi-vendor compilation pipeline that enables optimization passes from different parties to operate on the same IR. The IR should support defining custom gate sets, both basic and compound.

IR levels should be structured focusing on separation of concerns. High-level IR should be focused on supporting end-user programming languages and paradigms. This representation should avoid tight

coupling to the QPU details and focus instead on the quantum information science aspects – gates and classical control flow. One use case is the “high-level theorist’s language”, which would allow users to describe algorithms without dealing with the fine details of qubit mappings, resource management, and the like. This is a toolkit problem rather than an IR problem (unless one regards such a toolkit as a “high-level layer” in an IR stack), but the middle layer to which all upper-level programs should compile to must be able to support and implement anything imaginable in such a high- level language.

Low-level IR should be focused on transforming the high-level IR into QPU-specific representation. All IR levels may need to consume some amount of QPU metadata describing QPU constraints including connectivity, native gate set, control system capabilities, timing of different quantum, and classical operations. This metadata is expected to be provided by the QPU vendors in a standard format for defining resource constraints.

Due to the fast-paced development of the quantum industry, the IR must be sufficiently flexible to support new QPU architectures based on known and yet unknown physical qubit implementations. Profiles are a common approach to enable tailoring a lower-level representation to specific QPU architectures. A generic IR specification may be too broad or forward-looking for any given QPU to support. QPU profiles could be used to define IR subsets applicable to a specific QPU without loss of generality. The IR itself does not need to be intimately aware of all QPU capabilities and overly prescriptive. It may leave the details of quantum operations to the QPU vendors to implement.

When transforming a user program into IR, it is necessary to specify in what environment different parts of the program will run. The three execution environments are quantum, real-time classical, and near real-time classical. Quantum execution environment performs purely quantum operations on the qubits. Real-time execution environment performs calculations synchronously with the quantum circuit execution and within the lifetime of the qubits. Typical examples of real-time calculations are quantum error correction (QEC), iterative phase estimation (IPE), and repeat-until-success (RUS) circuits. Near real-time environments are useful for driving quantum algorithm execution, for example, VQE or QAOA. The relationship of near real-time execution environment to the quantum execution environment resembles traditional CPU-GPU relationship where the CPU is coordinating the main program execution flow and QPU acts as a co-processor or accelerator. The program compilation pipeline needs to be able to target the right environment for different parts of the program. Quantum IR should support explicit execution environment targeting.

To support quantum-classical programs in real-time environments, an IR should provide mechanisms for synchronization of quantum measurements and classical computation based on the results of the measurements. This can be done via synchronization primitives and instruction grouping. General ability to control instruction timing and parallelism at the hardware IR level could enable fine-grained user control over the program execution.

Open questions and challenges

A question to be answered by the quantum industry is whether quantum IR needs to reinvent classical computation parts, specifically for use in a real-time environment. LLVM IR serves classical computation well. MLIR extends LLVM into heterogeneous computing environments difficult to model with traditional LLVM IR. A combination of the two can provide a solid background for the development of quantum dialect. Relying on existing classical technologies would facilitate the ability to integrate classical languages with quantum more easily. It should be possible to write the classical part of the

hybrid quantum/classical program in a familiar classical language.

One notable omission, in perhaps all existing implementations, is the absence of hints for optimization and performance. For example, it is often possible to allocate a qubit without any requirements for accuracy, coherence times, and such. A useful analogy for these cases is the implementation of floating-point operations in classical computing languages, where one can often request a default precision (analogous to "any fidelity of qubit") but can also specify, in some cases, a requested precision to have some confidence the program would be successful. This would assist the upper layers of IR by giving confidence that the program would be successful and assists the lower layers with issues such as mapping to physical qubits. In an extreme case, one could encode a section of a program with multiple implementations, "the best" of which could be selected based on the capabilities of the selected machine.

Similarly, hints for parallelization and concurrency are important and will become more so if distributed/networked QC backends are produced. Qiskit handles this with "synchronization barrier gates" which stop execution until all instructions prior to the synchronization gate are executed, which is one option; but many more possibilities exist, such as assignable time slots for instructions. This extends to the idea of virtualization, which could allow, e.g., a 10-qubit machine to simultaneously execute 6-qubit and 4-qubit programs and decide whether or not primitives should exist in an IR to handle such cases (the inevitable noise of crosstalk prevents much success here in the NISQ era, but presumably this will get better; even then, if the "fidelity hints" mentioned above allowed a large enough tolerance, perhaps it would not matter).

Resource allocation and release is also an area that is not handled well by existing implementations. Qubits must in general be manually allocated and freed; many routines would require the use of ancilla qubits, and it would be useful for developers to have the option of manual allocation (possibly with quality hints, as above) or to have this abstracted. In a sense, this is not the job of an IR (the borrow/release semantics of Rust, the garbage collection of Scheme can both be handled by LLVM), but when one speaks of different layers of IR representations at various levels of abstraction, resource management may well fit into one of the layers. In almost no modern programming languages does one start a program by declaring how much memory it will use, but very often modern quantum toolkits need this up front. This places an unnecessary cognitive load on the developer. The "live time" of ancilla qubits could feed into optimization hints.

It has not been determined where error data fits into the IR—either by specifying desired qubit fidelity or with measurement data that is propagated through, and whether that should be handled at an IR level at all.

Opportunities

The idea of "write once, run anywhere" is has not been fully realized even in cases such as the Java Runtime Environment or Common Language Runtime), so the idea of "one IR to handle all use cases" may be unrealistic. A great challenge in adoption is the idea that the IR should support "any imaginable program". The different models of QC implementation (implementation of qubits, continuous variable methods vs traditional state vectors, and so forth) may indeed make settling on one IR impractical while the industry is in the throes of trying many different ideas.

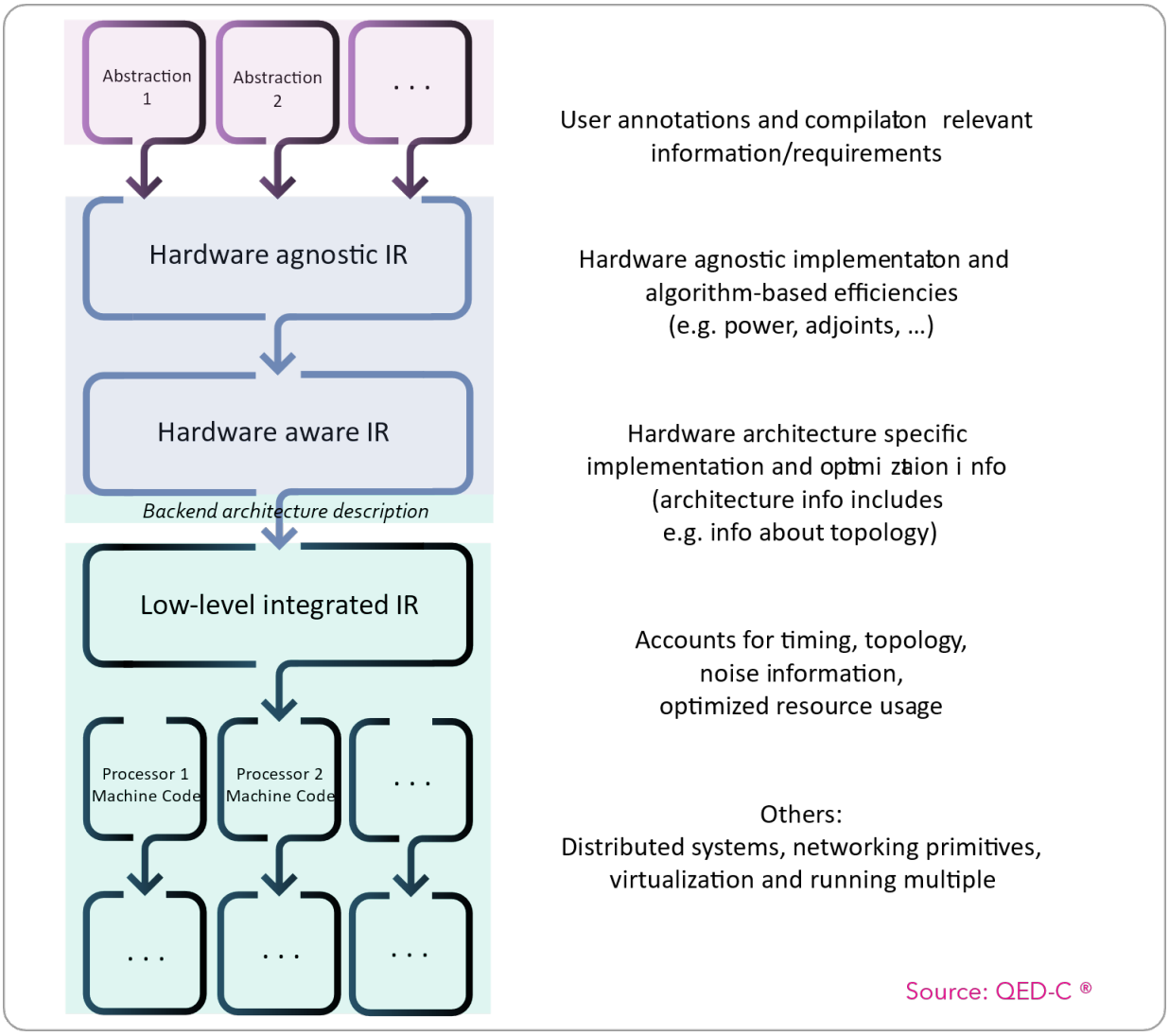

Figure 6: Summary of discussions across multiple levels of abstraction.

Due to the nascent state of the quantum industry, it would be helpful to start an iterative process for defining quantum IR levels, their scope, and interfaces. A lightweight community-based standardization process could be used to establish consensus and provide the forcing function to define an interoperable set of IR specifications. Given the place of IR in the quantum toolchain, it is best to focus first on the middle layers, which are hardware and programming paradigm agnostic (figure 6). This layer is the most vital to the whole IR stack and is more likely to enable interoperability in the future.

While working on IR specifications, it is critical to keep in mind that IR is part of the compiler toolchain. If cost considerations become a barrier to the adoption of any standard because existing frameworks have invested too much time and effort into their existing IR stack to contemplate switching to or supporting a new standard, creating a toolset to facilitate such a process would be desirable. To be useful, IR needs to be adopted by all ecosystem players from QPU to software vendors. Focusing on tools and early integration of IR proposals with the existing toolsets could enable an easier road to IR adoption.

In case the desire for commercial advantage hinders adoption of any standard, a potentially workable process for making progress on quantum IR could consist of the following iterative phases:

1. Define IR requirements, levels, scope. Establish focus groups to focus on different IR aspects. Answer the main question: can a useful interoperable IR be defined?
2. Solicit proposals from the industry. Let the existing industry players propose IR candidates.
3. Evaluate proposals against requirements, consolidate when possible.
4. Issue draft specifications for industry-wide review. In this step, IR is tested in the real world by ecosystem players. The feedback is collected and incorporated into requirements and updated proposals.

At some point in this iterative process, enough consensus is reached, and a specification supported by a large part of the community would be issued.

4. Hardware native approaches

Practical techniques for IR may take advantage of as many native features of the hardware as possible. This section focuses on the IR needs for executing the hardware-efficient ansatz as well as gate-free approaches.

This particular topic concerns two levels of IR: Level 2.2 (hardware ready program) and Level 3.2 (physical evolution to be realized on the quantum device). Therefore, the discussion on IR requirements is also framed on the two levels. The general principle that should be upheld is that if the user wishes to specify how a quantum program is to be run down to the lowest level of hardware details, the IR and associated compiler toolchain should not pose any obstacles in doing so.

At Level 2.2, the quantum components of a program are described as a sequence of elementary operations (gates) that can be executed on a given quantum hardware backend. This is currently supported by many solutions including OpenQASM, Quil, and Cirq. At Level 3.2, the IR needs to specify the quantum program at the pulse level. The most generic way to specify such pulse-level program is by a time-dependent Hamiltonian which contains terms with time-varying coefficients, namely

$$H(t) = \sum_{i=1}^{N} a_i(t) P_i(t)$$

where $aa_{ii}(tt)$ is a time-dependent coefficient and $PP_{ii}(tt)$ is a term in the Hamiltonian describing an interaction that can be realized physically on quantum hardware. This level of abstraction contains details of quantum hardware architecture, and it remains to be determined whether it should be exposed to a generic compiler toolchain or should remain proprietary with respect to each hardware manufacturer.

Opportunities

An outline of a set of examples and potential expressions that IR should be able to support is a necessary starting point. This could take the form of a repository of concrete use cases. The set of use cases would then help to implicitly define the requirements for an IR. In the repository, these algorithms should be distilled down to concrete steps instead of just being expressed at the theoretical level, with

the goal that each of these steps is supported by IR. The quantum algorithm zoo (https://quantumalgorithmzoo.org) already contains several known quantum algorithms which could be used as a starting point. More example implementations of specific features such as mixed descriptions of gates and pulses, as well as parameterized pulse descriptions, are also required. The ability to mix gates and pulses in the same representation allows the user to have the flexibility for optimizing the circuit implementation down to the lowest-level details without being burdened by excessively expanding every single gate into the same level of pulse details. This is currently supported in, for instance, XACC runtime. However, concrete examples are needed.

There is a need for transformers between the different IR proposals that are currently being proposed. The various IRs currently are seen to lack validation/verification and formal verification tools. There is also a need for adding debug statements/flags to IR. The lowest level needs logging and pulse capture. There should be more specificity and granularity into which layer of the funnel (discussed above) the existing IRs fit into and how they connect to each other. This will help identify gaps in the existing toolchains.

Standardized representations of the quantum hardware backend are required at the level of device Hamiltonian. This includes an Instruction Set Architecture (ISA) that outlines the available operations and their formats, machine configuration (topology, interconnects), and calibration data as applicable (e.g., fidelities, gate implementations). The most complete description would be of the form $H(t)$ as described above. However, useful partial descriptions can also be derived.

In order to develop an IR, one must consider the relevant algorithms. For instance, quantum simulation requires low-level control, whereas generally gate-free algorithms may not. For example, one may organize relevant algorithms as:

Algorithm	Gate-based	Mixed	Hamiltonian
ctrl-VQE			x
QAOA on Ising Model	x		x
Chemistry Simulation (GSE)	x	x	x
Quantum Simulation (time evolution)	x		x
Bosonic Sampling		x	x

Table 2: Classification of quantum algorithms

Here, “Gate-based” refers to algorithms that should be fully expressible as gate operations in the IR. Mixed refers to algorithms where single qubit ops are gate-based, but entanglement is hardware efficient (pulse-based or adiabatic). Hamiltonian refers to algorithms where only the Hamiltonian needs to be specified in the IR, such as QUBO or Ising expressions with adiabatic computation.

The mixed-use case indicates that an IR may need to describe both a Hamiltonian and gate-based system and can translate between them. Such a mixed IR where a particular representation is picked keeping efficiency in mind may be very useful in the short term, but it is not clear if it will scale well. It may be most useful in the transition to a fully error-corrected quantum computer but may become obsolete later.

Tools like Verilog from the classical world may provide some precedent for how the quantum IR at this level should look. There are cases in classical computing where a mix of digital and analog processes need to be supported, such as when dealing with sensor measurements for temperature and pressure, while still keeping room for specialized accelerators. An analogy is the combination of quantum sensing with quantum computing. For example, when dealing with cosmic rays, which is sensitive to detection by a quantum sensor, one might need adaptive error correction.

A final question is about supporting experimental quantum research. It is unclear whether the goal of IR should be to support current experiments in quantum science as well.

5. Annealing and analog models

Annealing and analog models are declarative quantum programming models; however, they are unique approaches to executing quantum programs. In contrast to focusing on the frontend, as in section (1) above, this section considers IR requirements for annealing and analog models in a "full-stack" context.

This section focuses on the use cases of quantum annealing and analog quantum simulation and examines existing IR solutions to determine whether they satisfy these target use cases. The prevailing view is that no single proposal simultaneously satisfies all the basic requirements for these use cases. Therefore, we consider minimum IR requirements, near-term IR requirements, and future IR requirements.

Minimum IR requirements. This captures the list of basic requirements for the simplest use case to run successfully. First and foremost, the IR needs to be able to *explicitly express effective Hamiltonians*, which is identified to be the most critical gap between existing IRs and a satisfactory IR for these use cases. In addition, the IR should be able to express different spin statistics including qubits, bosons, fermions, and multi-level spins. For some hardware platforms, such as cold atom platforms, the IR should capture flexible geometries in 1D, 2D, or 3D, including even non-lattice geometries, and be able to communicate these geometries to a lower-level, hardware-specific IR for better compilation and optimization. For the pulse-level IR, *continuous* pulse shapes and *global* pulses should be expressible. For verification and interoperability purposes, the IR needs to hold sufficient information and be expressive enough to support the *verification and validation* of Hamiltonians and the *transformation* between different effective Hamiltonians.

Near-term IR requirements. In addition to allowing the simplest use cases to run successfully, this list of near-term requirements enables the simplest use cases to run better. For quantum annealing, an IR should provide support for use cases beyond the Ising model. The IR should be able to support different measurement capabilities and measurement bases, such as measurement beyond 2-level systems and time-of-flight measurements in some analog simulation use cases. The integration of classical control flow is critical for some use cases relevant to this section but also for many circuit-based applications. There are two types of classical control flow integration that are necessary to aim for in the near term. The first is to support hybrid variational algorithms, which only require shot-by-shot classical flow and don't require classical control flow to be built-in within the coherence time. The second is to support floquet driving use cases, where classical control flow is needed within the coherence time, but no run-time classical-quantum communication is required. A bonus for near-term IR development targets is to create standards for device capabilities for different platforms and machines.

Future IR requirements. This includes developments on IR beyond the basic requirements for quantum annealing and analog simulation use cases. One of the requirements common to all use cases is the ability to support fast classical-quantum communication with the coherence time, e.g., for mid-circuit measurements. This is necessary for quantum error correction and would be useful for some annealing

and analog simulation use cases as well. An additional IR expressivity need is to support the capability of qubit rearrangement within the coherence time, which is a technology being developed for trapped ions and neutral-atom systems. For some specific use cases such as quantum chaos or quantum supremacy sampling, it is also useful for the IR to support the implementation of a classical ensemble of Hamiltonians. Lastly, a good IR could hold sufficient information to communicate error terms in the quantum device between the hardware and the user.

None of the existing IRs satisfy all the preceding, basic requirements to support use cases of quantum annealing and analog quantum simulation. The IBM IR infrastructure is perhaps the closest to be able to support these use cases. OpenPulse is suitable for low-level, pulse-level instructions, and OpenQASM is designed for gate-based approaches. No existing IR is able to express and validate Hamiltonians; Hamiltonian formulation needs to be explicit in the IR for it to be effective. The DWave IR, Dimod, supports Ising Hamiltonian simulation, but it is not expressive enough to include Hamiltonians beyond Ising models or classical-solution models.

For standard, circuit-based applications, some IR designers have proposed an MLIR approach that makes use of LLVM infrastructure. This includes QCOR, QIR, QUIR, and so on. The MLIR-based approaches are also applicable to quantum annealing and analog quantum simulation use cases. They are especially suitable for handling hybrid hardware integration in the backend, including the QPU, CPU, GPU, and FPGA. Real-time classical loops can be supported in the infrastructure. The concept of progressive lowering between IRs could potentially allow a unified framework for circuit-based and non- circuit-based use cases. There may be an opportunity to extend the current existing IRs to explicitly support Hamiltonians, which will be a major step in supporting annealing and analog simulation use cases. In the lower level, close to the hardware, pulse-level IR would be suitable for these use cases as well, but it also needs to inherit Hamiltonian descriptions from a higher level.

In summary, no existing IRs satisfy all basic requirements for quantum annealing and analog quantum simulation use cases. To support those use cases, current IRs need to explicitly incorporate Hamiltonian descriptions.

6. Low-level hardware features

The growing diversity of the hardware landscape and progress in developing and implementing quantum error correction codes motivate consideration of “low-level” hardware features. For instance, what does the IR need to have to accommodate the needs for error correction, error mitigation, and pulse engineering?

This section concentrates on the current state and needs of the community that is developing hardware. For the most part, the discussion is restricted to level 3 in the model in Table 1.

Critically, there are two different views of the IR problem with overlapping terminology, namely:

a. **Low-Level Development Language:** A language for the developer and an interchange format for program/experiment specification.

   Examples: Quil, OpenQasm

b. **Compiler IR:** A part of a system of tools and representations to support the construction of back-ends to compilers and compiler frameworks. Essentially this is the glue between an Abstract Syntax Tree (AST) and a machine-level representation that a compiler would output. Examples: QIR, QUIR, XACC

Furthermore, there are differences in the semantics being represented in an IR of either type.

- **Quantum Gate Model:** This is relatively well understood and supported by examples in both types of IR.
- **Classical:** These are classical instructions required to specify the implementation of an algorithm.

  In the first case, classical instructions are used to support decision-making and computation with a circuit execution (e.g., error correction protocol implementation); in the second case, supporting classical computation more general hybrid quantum-classical algorithms (e.g., VQE, QAOA).

  - **Dynamic control** – Tightly coupled and intended to be executed within the coherence time of the related qubits.
  - **Hybrid applications** – For example, classical computing is used to create and update subsequent execution of a circuit.
- **Physical System:** This is different for different modalities:
  - **Quilt-T and OpenPulse** capture this for superconducting qubits (e.g., transmons).
  - **Ion traps** also require support for notions of motion and cooling and even physical swaps.

There is a fundamental tension between standardizing on a single representation across modalities—even for hardware vendors of similar modalities, versus expressing differences in features and fundamental system architecture due to high rates of innovation and the state of maturity of quantum computing.

This issue becomes particularly difficult for specifying a low-level development language (as opposed to a compiler IR). At the current state of development practice, it is difficult to express different hardware-specific hardware optimizations in a purely generic way and expect optimization to flow down to the hardware without fairly complex and nuanced compilation techniques (e.g., circuit parameter updates). When you add dynamic control instructions and a way to express physical system computation (e.g., pulse control), hardware-independent expression becomes very difficult.

For example, there are varying degrees of support among vendors today (including non-support) for functions like:

- Mid-circuit measurement
- Dynamic control
- Programming at the physical system level
- Features to support efficient execution, e.g., dynamic update of circuit parameters, active reset, conditional reset).

It is worth also mentioning that the state of compiler IRs for quantum is also quite nascent and so standardization is hard. While many agree that LLVM is a very good model for specifying classical instructions, there are issues moving to quantum:

- There is conceptual conflict where entanglement and the no-cloning theorem prevent separating a quantum “value” from its physical resource, e.g., memory vs value semantics in MLIR and LLVM for describing qubits.

- At some point, however, the computation is specified in the IR and one or more synchronized schedules of instructions need to be produced for execution on the hardware. Tools like LLVM do not do a good job at capturing this.

- A traditional compiler will often produce a single package that can be loaded and deployed by an operating system loader and in a format suitable for a language and system Application Binary Interface (ABI). In a modern quantum computing system, this can result in a distributed systems problem requiring multiple targets and ABIs.

Opportunities

Increased focus on systems supporting a full compiler IR for back-end integration and language development environment should be prioritized. These include QIR, XACC, and QUIR. Doing so will spur the development of languages for pure quantum and hybrid quantum computation. With a set of two or three of these in place, a hardware vendor can more easily support a broader collection of languages.

One thing does seem certain, there is much more work to be done in language development for quantum programming. A small collection of compiler frameworks to support this development would be a significant boon to the industry. With this in mind, there is no *particular* need to standardize on the IRs here, except at the classical level. The LLVM IR is a robust and well-respected tool for this and, despite the limitations outlined above, is the preferred solution whenever appropriate.

Additionally, various proposals for a low-level development language have been expressed, including QUIL and OpenQasm3, and continued work to attempt to consolidate features among them is valuable.

Finally, one development language is probably not a good solution. However, a high-level gate model interchange standard would be very valuable and easily incorporated into any of the existing and likely future languages.

Recommendations

Based on the preceding sections and the opportunities and requirements therein, the following high-level recommendations are distilled for future efforts in the area of practical intermediate representation for quantum computing.

Recommendation 1: Conduct an architecture study to identify required IR functionalities

Conduct an architecture study for clarifying the functionalities required for IR. The goal is to clarify the relationship between the IRs and compiler toolchains that currently exist. Such a study will help the community to agree on various levels of abstractions. The study will also help consolidate languages, both at a low level and a high level. It is worth seeing how such an architecture study impacts the process of defining the high-level and mid-level IRs for declarative quantum programming.

Recommendation 2: Determine reference implementations for specific quantum computing use cases

Collect concrete reference implementations for specific quantum computing use cases. Such reference

implementations may be contributed by players across the industry and used to consolidate various IR proposals. IR requirements should be framed according to different types of use cases, with the low-level IR requirements driven by specific quantum computing use cases. The reference implementations will also help shed light on the priorities for incorporating the currently exotic hardware features, such as measurements beyond 2-level systems and qubit rearrangement.

Recommendation 3: Define abstract models for quantum computing hardware

Define appropriate abstract machine models for quantum computing hardware backend. This has been proposed for imperative paradigms focusing on gate-model quantum computing as well as for declarative paradigms including analog simulators and quantum annealers. There is a related need for a standard representation of hardware backend. Such machine models will facilitate the development of systems supporting a full compiler IR for backend integration.

Recommendation 4: Collaborate through QED-C

Form one or more dedicated work group(s) for specific directions that contribute to the recommendations in this report. The progress made in the work group(s) may provide the basis for a follow-up PIRQ workshop as well as the called-for industry-wide IR consolidation process.

Conclusion

Overall, while the current proposals of IR for quantum computing cover a substantial amount of the full stack ranging from source code to quantum hardware, additional work is needed. This includes clarifying how the different proposals relate to each other architecturally, how they can be applied to specific QC algorithm use cases, as well as how appropriate abstract machine models can be defined for quantum backends. The above recommendations will help advance practical intermediate representation for quantum.

Appendix A: PIRQ workshop agenda

<table>
<tr><th colspan="3">Agenda for July 2021 PIRQ Workshop (all times in ET)</th></tr>
<tr><td colspan="3">Pre-Workshop: Monday, July 25 Pre-read materials</td></tr>
<tr><td colspan="3">You will receive a document before the workshop. You are highly encouraged to review the document before the workshop, so you have the right context to evaluate demonstrator proposals in Session 1. The document also contains information about IR levels and use cases to be discussed in Session 2.</td></tr>
<tr><td colspan="3">Session 1: Thursday, July 29 Existing proposals of IRs for quantum</td></tr>
<tr><td>10:30 am – 11:00 am</td><td colspan="2">· Brief introduction (Yudong Cao)
- Goal of the workshop
- Workshop agenda
- Participation from attendees</td></tr>
<tr><td>11:00 am – 11:45 am</td><td>IBM Demo
(Presenter: Ali Javadi)</td><td rowspan="5">· Interactive demos with questions from participants answered live
· Demonstrators include components:
- OpenQASM 2 or 3 (DAG),
- Cirq,
- quil IR,
- QCOR (MLIR),
- XACC IR,
- QIR,
· Each demo focuses on a concrete quantum program instance and demonstrates at least one of the following:
- (“write once, run all”) How to generate IR from source code written in a programming language and execute it on two different quantum backends to produce the same outcome
- (“run all that is written”) How to generate IR from two different programming frameworks (language/library) and execute it on a quantum backend to produce the same outcome</td></tr>
<tr><td>12:00 pm – 12:45 pm</td><td>Google Demo
(Presenter: Bálint Pató)</td></tr>
<tr><td>2:00 pm – 2:45 pm</td><td>Rigetti Demo
(Presenter: David Rivas)</td></tr>
<tr><td>3:00 pm – 4:45 pm</td><td>Microsoft/ORNL joint Demo
(Presenter: Alex McCaskey, Bettina Heim, Stefan Wernli, Thien Nguyen)</td></tr>
<tr><td>5:00 pm – 5:45 pm</td><td>D-Wave Demo
(Presenter: Jason Necaise)</td></tr>
</table>

<table>
<tr><td colspan="3">Session 2A: Friday, July 30 IR at different levels and use cases</td></tr>
<tr><td colspan="3">Goal of this session is to solicit participant feedback and opinion on specific questions about the requirements of IR at different levels and use cases.</td></tr>
<tr><td>10:30 am – 11:00 am</td><td colspan="2">Opening (Yudong Cao, Zapata)
- Outline dimensions of discussion
o Different levels and sub-levels of IR
o Different use cases of quantum computing</td></tr>
<tr><td>11:00 am – 11:15 am</td><td>General gate model quantum algorithms (Presenter: Chris Grenade, Microsoft)</td><td rowspan="6">· Short 10–15 minute presentations from QED-C ecosystem participants to present examples of use cases of interest to them
· Key points:
- Follow Chatham House Rule for confidential information
- Focus on requirements rather than solutions</td></tr>
<tr><td>11:15 am – 11:30 am</td><td>Gate model quantum simulation (Presenter: Eugene Dumitrescu, ORNL)</td></tr>
<tr><td>11:30 am – 11:45 am</td><td>Hardware-efficient circuit ansatz (Presenter: Ntwali Toussaint, Zapata)</td></tr>
<tr><td>11:45 am – 12:00 pm</td><td>Gate-free approaches (Presenter: Nick Mayhall, Virginia Tech)</td></tr>
<tr><td>12:00 pm – 12:15 pm</td><td>Quantum annealer (Presenter: Scott Pakin, LANL)</td></tr>
<tr><td>12:15 pm – 12:30 pm</td><td>Analog simulator (Presenter: Shengtao Wang, QuEra)</td></tr>
<tr><td>12:30 am – 1:00 pm</td><td colspan="2">· Participants select which breakout room topic they would like to join.</td></tr>
<tr><td>Afternoon</td><td colspan="2">· Organizing committee decides breakout session structure based on participant selection and feedback. Each breakout discussion should focus on one breakout room topic.</td></tr>
</table>

Session 2B: Monday, August 2 ***IR at different levels and use cases – Breakout groups***	
10:50 am – 11:00 am	Opening (What to expect today)
11:00 am – 12:30 pm	· Breakout group discussions (recorded) - Each breakout room focuses on a set of (level, use case) combinations, ideally moderated by one OC member - Group participants provided with: 1) set of references representative of use case being considered, 2) set of questions to be answered by the group: o Does any of the current solutions demonstrated in Session 1 satisfy the needs of IR at this particular level for this particular use case? o What are the requirements that the solutions demonstrated in Session 1 can already fulfill? o What are the IR requirements that accommodates future advances in quantum computing? How do future advances impact the requirements? What hardware evolution do we need to take into account? o What is your interest in IR? How could IR help with your work? o What additional requirements does the IR at this level need to fulfill for this particular use case? o What requirements are there for the IR at this level to interface with the other levels? o Are there any additional reference(s) that should be considered in the discussion? - Participants generate short presentation (2–3 slides via template) and also short report by end of breakout session to answer the questions above - Each group selects one representative to present results to the entire workshop
1:30 pm – 2:30 pm	Breakout group presentations

Session 3: Tuesday, August 3	***Near-term, actionable milestones***
Goal of this session is to have an open discussion about near-term, actionable milestones for IR. As a conversation starter, we propose the following "strawman" proposal: Phase 1: An IR that can be transformed into existing representations Phase 2: Reverse transformations from the existing representations to the IR. This assumes that the transformation in step 1 is lossless Phase 3: An IR that can be executed on different quantum hardware backends Phase 4: An IR that is able to capture the full quantum-classical program expressibility. This includes classical-quantum feedback such as repeat-until-success circuits, quantum error correction, which concerns operations inside quantum coherence, as well as feedback loops such as variational algorithms	
10:30 am – 11:00 am	· Opening: Introduce four milestones outlined above, answer any questions from participants
11:00 am – 11:45 am	· Breakout sessions: - Any comments/feedback/improvement on the proposal (Phase 1–4)? If you can design the milestones from scratch, what would they be? o Generate short presentation (2–3 slides) to answer this question o Each group selects one representative to present results to entire workshop - Think about what can be done near-term, outline prototyping activity
11:45 am – 12:30 pm	· Breakout group presentations
12:30 pm – 12:35 pm	· Closing remarks
12:35 pm – 1:30 pm	· Open discussion: Workshop officially over, but you can linger for discussion if desired

Appendix B: Workshop breakout structure

The workshop opens in Session 1 with demonstrations of existing proposals for intermediate representations for quantum computing. In Session 2 of the workshop, the participants were polled on their interest to participate in discussions around different breakout room topics (check-marked in figure 7). The organizing committee then grouped the participants according to their interests. Each group of participants was directed into a focused breakout session to discuss the IR requirements for the specific set of IR levels and applications. In Session 3 of the workshop, the participants resumed the same breakout session grouping to discuss near-term milestones. To facilitate the discussion around IR requirements, breakout rooms were organized as shown in figure 7.:

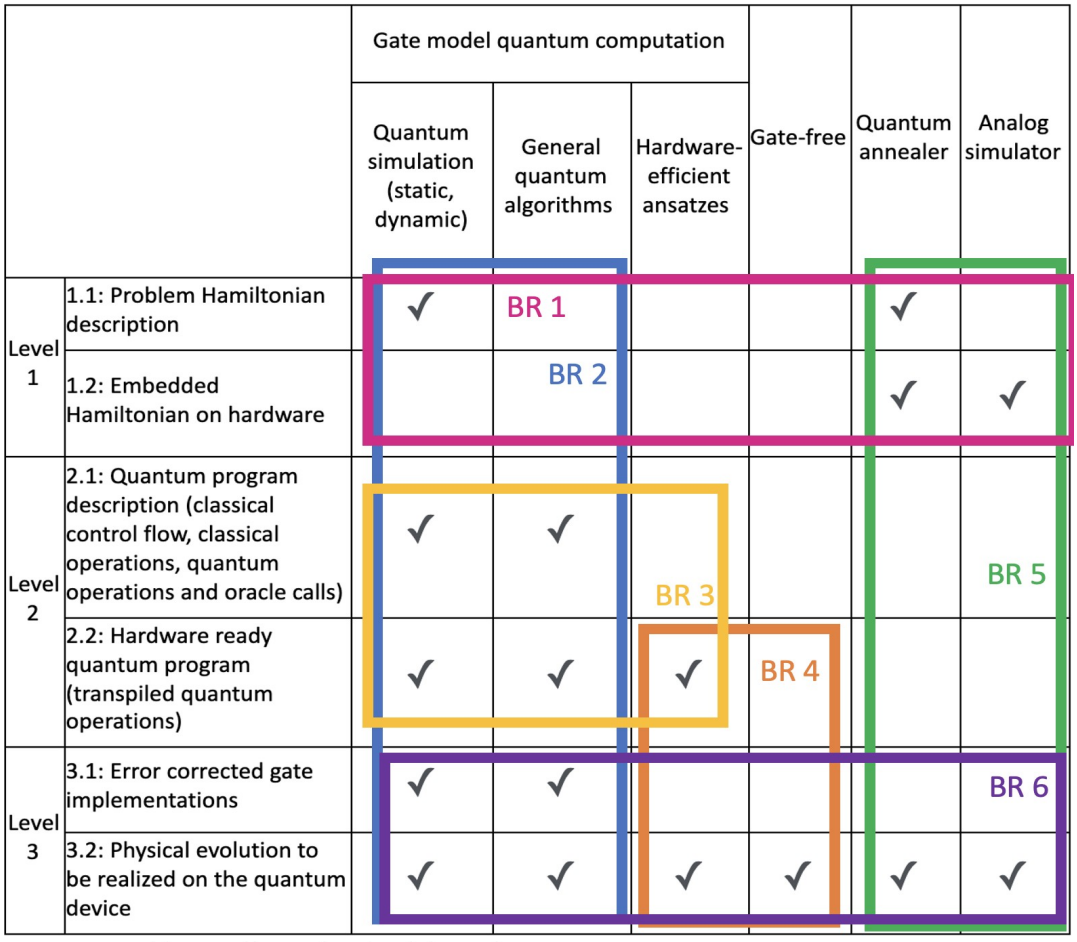

Figure 7: Breakout rooms superimposed on levels and applications relevant to practical IR for quantum.

BR1: Declarative quantum programming. This breakout room is concerned with the declarative aspect of quantum programming, where a program is described by "what needs to be done" and leaves the "how" to the compilers and the hardware backend. A typical example is constraint programming, where a program only specifies the constraints among a set of variables and leaves it to the compiler to generate an assignment of the constraints that maximally satisfy those constraints. Such a declarative model is also echoed in quantum annealing and analog quantum simulation paradigms.

BR2: Gate model vertical solution. This breakout room is concerned with executing highly structured quantum algorithms, which covers most quantum algorithms discovered to date. The idea is to have a space for participants with expertise at different levels of IR to come together and have a focused session about "if we were to execute an instance of an algorithm *X*, what information should the IR maintain throughout the levels?"

BR3: Gate model focusing on circuits. From the motivating scenarios described in the Introduction, it is apparent that quantum circuits are a central object concerned in the current quantum programming practice. The goal of this breakout room is to have a focused discussion just about quantum circuit representations. A good starting point would be to address these scenarios.

BR4: Hardware native approaches. This breakout room focuses on practical techniques that are meant to take advantage of as many native features of the hardware as possible. In contrast to BR2, this breakout room will focus on the IR needs for executing the hardware-efficient ansatz as well as gate-free approaches.

BR5: Annealing and analog models. While both are declarative quantum programming models, their unique approaches to executing quantum programs warrant them a separate breakout room. Unlike BR1, which focuses on the frontend, this breakout room allows for discussions about IR requirements for annealing and analog models in a "full-stack" context.

BR6: Low-level hardware features. With the growing diversity of the hardware landscape, as well as the progress in developing and implementing quantum error correction codes, it makes sense to have a breakout room dedicated to these "low-level" hardware features. For instance, what does the IR need to have to accommodate the needs for error correction, error mitigation, and pulse engineering?