## Tool Pipeline View

### **Goal**

**Survey the landscape of intermediate representations (IRs), compilers, and toolchains used across quantum and HPC platforms. Identify shared patterns, critical gaps, and collaboration opportunities in toolchain development.**

### **Session Chairs**

* **ORNL:** Daniel Chaves Claudino
* **IonQ:** Coleman Collins

### **Prompts**

* Which quantum intermediate representations (IRs) (QIR, OpenQASM, MLIR) are in active use, and for which pipeline stages?
* What strengths and limitations have teams experienced with each IR in circuit optimization, mapping, or error mitigation?
* What tools have been developed for circuit manipulation? What are these tools' input/output representation formats?
* How are different groups handling multi‑IR support—via translators, envelopes, or bespoke pipelines?
* What vendor or open‑source tools support each IR (e.g., QIR, OpenQASM, MLIR)?
* Where do tool‑chain gaps still exist?
* Is there appetite for a circuit representation that's more amenable to circuit transformation? and what features would it need? Can MLIR fill this gap?
* How will the IRs need to change as we move to the FT era? Will they scale up with circuit size?
* If providers decide to be opaque below the circuit level, what tools are needed so that higher levels can do their job
* As we progress how can we make sure that the interfaces and tools we develop for research and development don't make our lives difficult when we get to the FT era?
* What can we learn from classical side? ML? Programmability of hardware? Draw parallels between ML and QC. How ML was evolving: IR, programmability, tools development. Get a history of ML: pytorch, jax, tensorflow, why did pytorch win?
* What data will need to be transferred from the HPC to the quantum resource, and how large will the dataset be? Will compression be required?
* **What are the 2 highest priority requirements for the tool pipeline view?**

Tool pipeline session #1: Motivation

What problems are we trying to solve/achieve? Highest priorities/requirements?

Start from the lessons learned from classical compilation

Two motivations: how to use the tools to accelerate the field? Figuring out the boundary between the software and hardware. Low-level constructs are adopted by hardware.

Boundaries define the interfaces

We should not write the software we’ll be using in the long-term

A capability: every compiler stack should start thinking about error correction

A requirement: compliance with high latency demands

How to move the interfaces with the software-hardware boundaries?

What information do tools need?

There are no good ways for way layer of the stack to track changes ion another layer

Find the right/relevant information to track from the devices and how to properly represent the information

HPC/ML engineers can help with how to curate/manipulate data. Needs to be modeled beyond unstructured learning

Thoughts on gamification for quantum computing?

Compilation to help applications: dialects? DSLs? E.g., Quantinuum’s guppy. Respecting the fact that QC is mostly pythonic, so embedded DSLs would make sense.

While this foster proliferation of languages/dialects, IRs will be key to ensure interoperability

Progressive lowering can help compilation by dealing specific compilation issues/domains in isolation

Tool pipeline session #2:

What information do we need to and from HW?

What information does the compiler need from the QPU?

Better organization/structuring of the output of the QPU for post-processing

At the highest level, the answer is independent from the algorithm, whose performance is dependent on the HW

Quantinuum has identified drawbacks in MLIR

File size in current representations/IRs is not manageable

Can IRs be human-friendly and manageable?

For quantum ISA, where/who should implement it? Should it be logical, physical, or both?

Do we expect support only for “gates” or arbitrary unitaries, which will necessitate more complex compiler? Support for open quantum systems/new channels?

There is a need for static analysis tools suitable for QIR or other IRs

QC lacks formal verification methods and tools

What interfaces will be needed for IRs dealing with logical qubits?

Session #3

What are current and future limits of portability?

Time as a differentiator: what can be run in different modalities?

Application determines requirements, which can affect portability

There needs to be a level of agreement across different platforms for it to be portable

Portability requires abstraction layers, e.g., ZX calculus, which increases productivity

Performance portability across resources can help validation and quantum computer time allocation

How much is validation from a classical computing standpoint valid?

Can we think of portability beyond just quantum computing/circuits to an entire program/workflow?

There is an overlap in terms of what vendor APIs cover

Portability can take many forms: compilation, performance, etc…

Performance dimensions: resources, time-to-solution, energy, fidelity, correctness

Portability vs cutting-edge applications/highly codesigned: what’s the timeline?

Maybe invest in portability as the applications allow to avoid siloing/niche communities, bringing more people in

What questions need to be addressed in the frontend that everyone agrees on in terms of portability?

IRs, ISAs, less black-box/more easily tuned compilation

What can be learned from similar fields in classical computing?

Look into Mojo for inspiration

In what language (e.g., python, Julia, etc) to express quantum algorithms?

What are the trade-offs in adoptions IRs with each individual hardware?

How can we parallelize QIR?

How can tool development keep up with updates?

Diagnostics?

Session #4

The SW/HW boundary changes with provider/business model

Ability to run benchmarks

Industry standard IR

IRs that can highlight vendor differentiators, this with JIT can offer more flexibility

IR to conform to an abstraction layer that allows the IR to be extensible, can leverage code that is not IR-adherent

Should HPC facilities own/dictate large portions of the stack?

Tension between homogeneity vs heterogeneity

In the road to FT, what changes and what stays the same?

Along the way, there should a canonicalization of QEC codes that are proven to work in most/all HWs and also canonicalization of decoders

Each HW will likely converge onto a specific type/set of codes, and algorithms will map differently onto different HW/codes

Proper abstraction layers should carry over to FT computation

Gates are probably not the best representation

What topics will the community gravitate toward? LLVM is growing, OpenQASM seemingly on the decline, MLIR