IEEE Quantum Week – Sept 2024

Summary:

Improvements in quantum hardware, networking, software, and algorithms continue towards a goal of 100s of error-corrected qubits in 5 years for scientific usefulness ("research *with* quantum"), and 1000s in 10 years for more general commercial advantage (e.g. CFD). In the meantime, applications which simulate quantum systems will be early adopters. Slow but accurate quantum computations can be used to train AI which then guides fast GPU-based estimations. Today's quantum research community is small, cross-functional, and collaborative.

Hardware & Utility Roadmap:

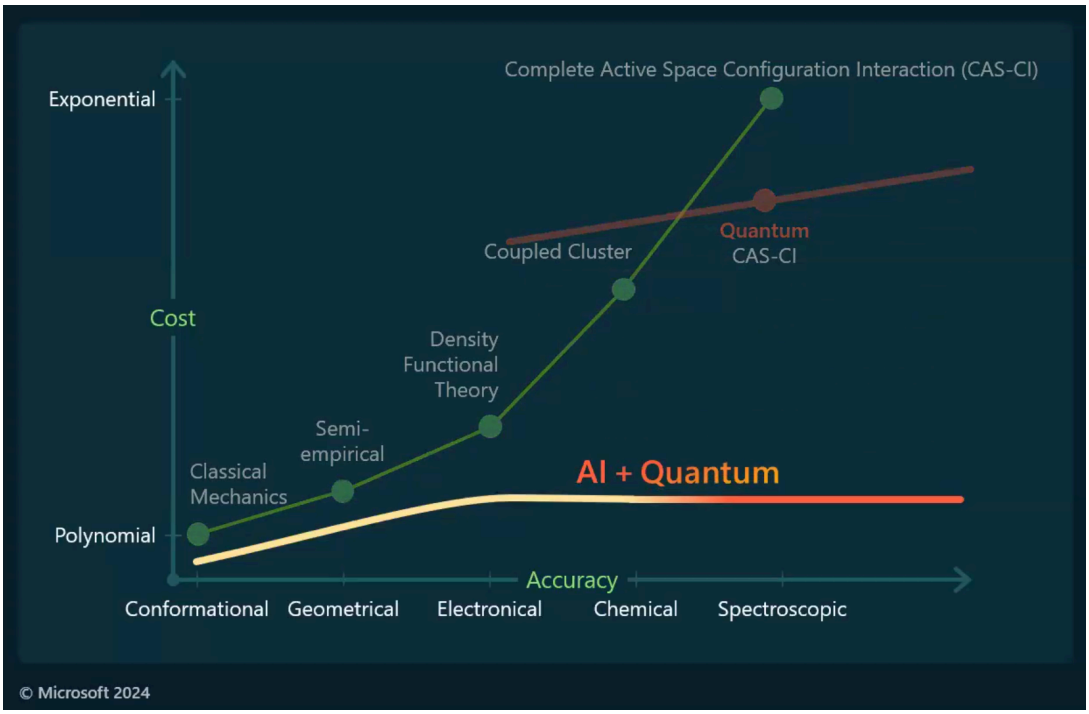
Due to the fantastic theoretical speeds of quantum computers relative to their classical counterparts and the potential to advantageously move beyond just modeling natural quantum systems, it seems worth it to keep monitoring the technology as it evolves over the next 5 years achieving scientific advantage at around 100 error-corrected qubits, through to perhaps commercial advantage in 10 years at 1,000 reliable qubits. This is reflected in the roadmaps of the major hardware players (e.g. IBM, Quantinuum) in their solo or joint ventured efforts, although it's reasonable to assume there will be technical and business winners and losers, and that perhaps in interim phases of development different hardware designs will be more useful for certain problems (e.g. annealing). Comparative performance metrics are being defined and developed, and we've previously called out the ongoing DARPA effort to provide metrics guidelines and a bake-off.

Applications:

Certain scientific domains will also be more amenable in these next few years - molecular biologics, quantum chemistry, material science - and we see some vendors (e.g. IonQ, Microsoft) aligning for those use cases. We see early use of quantum algorithms for image segmentation, recasting the NP-Hard problem as a QAOA in log(N) qubits, more early use of LBM for CFD, and several cases where a problem which is not itself a quantum system being solved on a quantum computer by creative "rephrasing" of the problem (e.g. model as a Hamiltonian and Trotterize).

HPC Integration:

We also see the beginnings of using an HPC + QAOA approach to explore larger multi-variate design spaces. The role of quantum computing in the enterprise portfolio of HPC was well illustrated (below) for a chemistry use case in the Microsoft keynote and amplified by others - quantum computing might be initially slower than GPU-accelerated classical computing but can provide more exact results which might also be dually used to train an AI which then guides the GPU-accelerated algorithm. HPC can also be used for simulators, and while the usefulness of generic simulators is reaching a limit, simulators customized for the application may extend the utility, as may algorithm-assisted circuit cut, distribute, and restitch.



Software as Science:

Towards productive science, there's the sense that the quantum tech stack is onerous, and that a SME-RSE (Subject Matter Expert and Research Software Engineer) partnership is necessary to enable the evolution from "research about quantum" to "research with quantum". Here the SME and the RSE may potentially and iteratively express the solution in different yet equivalent and transparent manners, the SME in domain-specific and algorithmic terms, the RSE in one of several implementation-specific forms. Code becomes research, code is reproducible result.

Implementation:

Python continues to dominate the front end, with Rust in the backend, enabled by LLVM-based compilers for the variety of quantum devices. These toolchains are necessarily customizable to interoperate with specific hardware and to provide plugin algorithmic optimizations based on some early intermediate-representation standards (e.g. QASM). We're seeing the need for compilers to retain more meta-information rather than classically strip it, and then while "progressively lowering" using that metadata to inform deeper kinds of optimizations later in the toolchain. Small codebases at this stage means that there need be no long-term commitment to any given framework, and it seems anyway that their similarities are stronger than their differences. End state would be domain-specific expressions for algorithms which self-compile into multiple executables automatically targeted at one of several available runtime compute types - CPU, GPU, QPU, etc. - and a workflow to orchestrate them. While we think a focus on pure runtime performance at this stage is premature, some vendors (e.g. Quantum Machines) claim performance while dodging the larger workflow problem and tying the quantum computer to a specific HPC node. Oak Ridge is starting to work on some of the potential scheduling variations.

Next Steps:

1) id early opportunities, 2) extend collaborator network, 3) grow cross-functional team

Short of basic quantum research, scientific applications are currently often classical-quantum hybrids, and it seems expedient for demonstration of utility to bend to the predominant workflow paradigm of a given early-adopting scientific domain. Collaboration is key - papers tend to have long lists of inter-enterprise collaborators - there is an understanding many current ideas won't pan out, and collegial pivots will be needed. At this stage, researchers can obtain benefit from work in the quantum space without disclosure of confidential IP, enabling broad collaboration, and at this stage a foundation of collaborative work is needed to compete for funding sources. The role of the research software engineer is also key to a working team which includes hardware and subject matter experts.

Appendix - QC Landscape:

IBM's Qiskit is currently the dominant circuit-oriented programming framework, with end-user writing in Python, services in Rust, an extensible and interoperable transpilation toolchain, and performant simulators which can make use of cuQuantum for NVIDIA. Most of the hardware vendors provide backends for Qiskit, and some software companies are starting to provide value-add toolchains based on intermediate representations like QASM. IBM is also very consistent with their educational programs and online materials and is a good starting point for learning quantum computing theory and practice. Xanadu's PennyLane is similar in many ways but less pervasive. Microsoft and their collaborators prefer their Q# language, not based on QASM. In the future we see programming paradigms evolving to include higher-level domain-specific and algorithmic constructs. Here's a brief non-exhaustive list of some of the major players and their software - we'll let the industry, and DARPA, continue to define the comparative metrics:

- Alice & Bob - "cat qubit" hardware; small qubit counts but bit flips stable for minutes; backend for Qiskit
- Amazon - multi-vendor classical & quantum platform; AWS backend for Qiskit; also has own simulator
- Atom - grid of 100 non-QEC neutral atom qubits; use Qiskit "cold atom" provider or Q#
- D-Wave - annealing; own "Ocean" Python SDK, plugin for Qiskit
- Google - superconducting hardware; own "Cirq" Python lib and block-centric Qualtran language for estimation
- IBM - 133 non-QEC superconducting qubits today, in the cloud; 100s of QEC qubits in 5 years, 1000s in 10; Qiskit
- Intel - silicon spin qubits; own "QuantumSDK"
- IonQ - trapped ion qubits; backend for Qiskit; app focus on molecular biologics and chemistry; see M. Yamada
- Microsoft - Q# language; Azure multi-vendor cloud platform, akin to AWS; software-tier QEC; future topological qubits
- Quantinuum - 56 n-way connected trapped ion non-QEC qubits, 1000s by 2029; partnered with MS for QEC
- QuEra - 256 non-QEC today, 30 QEC neutral atom qubits in 2025; own "Bloqade" SDK
- Rigetti - superconducting qubits; backends for AWS / Qiskit, Azure / Q#
- Xanadu - photonics hardware; PennyLane Python lib interops with Qiskit and adds photonics and gradient features