How can we accelerate quantum resource estimation learning and development? Introducing the first public GraphQL API for quantum resource estimation. A software engineering approach.

Y. V. Bermudez
Universidad ORT Uruguay

Introduction

While currently, quantum researchers have access to various software tools and frameworks for quantum computing, estimating the resources required for implementing quantum algorithms remains a complex and often opaque process. For example, creating a Benchq estimation involves installing Python, Julia, PySCF, Jabalizer, and other packages, which makes the process of getting started with resource estimation quite difficult for the average learner. Azure QRE is available only through Visual Studio Code and cannot be integrated with other applications.

We have identified several problems: no readily available code review integrations to QREs (i.e using Github Actions or Gitlab), a lack of Q# or QASM playgrounds to learn quantum development, and also missing documentation and guides regarding how quantum resource estimators work internally and how small differences in variables change results.

This challenge is intensified by the diversity of hardware platforms and error correction schemes, which significantly increase the amount of inputs estimators can receive as parameters. Learners at the moment cannot easily tweak these parameters to instantly see results in an intuitive and pedagogical manner.

For the industry to progress faster, and to bring these technologies closer to the average developer or the “quantum curious”, we need a core library and a standardized API. This will allow us to:

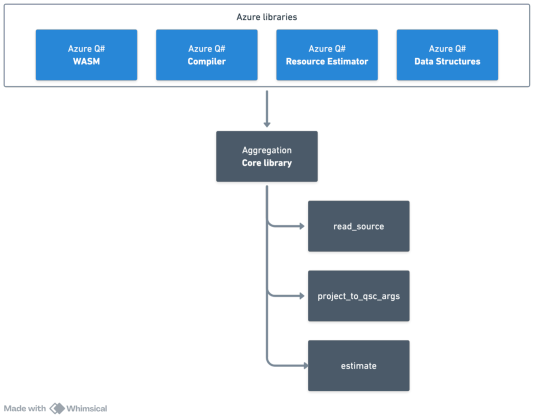
- Connect different quantum resource estimators to aggregate and compare estimation output (Azure QRE vs Benchq).
- Allow these quantum resources to be integrated into any code editor. Right now, only Visual Studio Code allows plugins for QRE.
- Allow researchers to integrate QRE automations to their code review pipelines (via Gitlab or Github), instantly displaying resource estimation differences on each pull request or review.

To address this issue and to take a step further into these objectives, we present the first publicly available GraphQL (Graph Query Language) API that provides a standardized interface for quantum resource estimation. Our API allows users to input parameters related to quantum algorithms, qubit technologies, and error correction. Using these parameters, it returns detailed

estimates of physical and logical resources required for the quantum algorithm to be executed, including the number of qubits, gate operations, and runtime.

We believe that by providing transparent and easily accessible resource estimation tools, this publicly available API will significantly contribute to the development speed of quantum applications, and will also help bridge the gap between theoretical research and practical implementation.

Reverse-engineering existing QREs



To be able to create an API for quantum resource estimation, we first need to be able to use the underlying resource estimators (Azure QRE, Benchq) at a low-level. Unfortunately, these tools don't provide a usable library for developers to create their own custom applications on top of them. To solve this, we reversed-engineered the Azure QRE codebase to understand how to create a low-level integration with the following libraries:

- Azure Q# WASM module: For Typescript support. Current Visual Studio Code plugins use this module for quantum support.
- Azure Q# Compiler: To understand how the compiler arguments change depending on different quantum resource estimator inputs (i.e qubit type, QEC scheme).
- Azure QRE estimator: We are interested in the estimate_entry function from the Azure QRE library. This is our entry point to our QRE integration.
- Q# data structures: Needed for access to "PackageGraphSources" and "Project" structures. This represents a Visual Studio Code Q# project. We have engineered it so that we do not need this IDE for integration.

Researching and coming up with the core aggregation library was not straightforward. Azure's QRE code repository lacked documentation, but after arduously debugging and internally documenting each method, we reached a milestone: running an estimation through the command-line, without the help of Visual Studio Code plugins.

Running an estimation: Bell State Q# algorithm

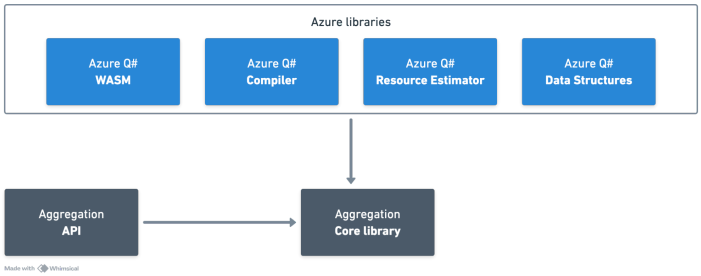
Currently, there is no existing CLI tool to execute Azure QRE estimations on the terminal. However, using our CLI module and core library, we were able to run an estimation using the following command.

```
cargo run -- ../q#/bell_state.qsc
```

See the attached link for the Bell State Q# algorithm we chose as an example. The output of this execution can be seen here. To summarize, it contained the job params (including QEC schemes, error budgets, qubit params, etc.) and the physical and logical qubits necessary for the algorithm to be executed under those constraints, among multiple other fields.

Running an estimation: Integrating our core library to an API

As we have detailed previously, the end goal of our research is to create an API for resource estimation. Having a CLI tool integrated to the core aggregation module, proved that integration with external services was possible.



For the next step, a GraphQL API was set up. A standard structure, inspired by Azure QRE, was defined as the API's request parameters. *File, label, detail, params* are the required inputs.

- File: Contains the URL for the Q# file to estimate.
- Label: Details the title of the estimation.
- Detail: Explanation of the Q# estimation that will be executed.
- Params: Qubit type and QEC schema to use.

Using this structure along with GraphQL mutations and queries, we were able to create an HTTP request for the resource estimation of the Bell State algorithm. Furthermore, the estimation output was fully customizable and filterable using the query language's syntax.

Future applications and impact

By providing a standardized, accessible interface, we anticipate increased integration of quantum resource estimation into various development workflows, educational platforms, and research tools. This could lead to more efficient algorithm design, better-informed hardware development decisions, and accelerated progress in practical quantum computing applications.

This democratization of access to quantum resource estimation tools has the potential to foster innovation, promote interdisciplinary collaboration, and ultimately accelerate the transition of quantum computing from theoretical promise to practical reality. As the field evolves, our API can serve as a foundation for more sophisticated quantum development environments and contribute to the establishment of industry-wide standards for quantum resource estimation.