Extended Abstract for the First International Workshop on Integrating High-Performance and Quantum Computing

qbOS: a Python framework for the development of coprocessing quantum-classical applications

Seyed N. Saadatmand
Quantum Brilliance Pty Ltd
Canberra, ACT, Australia
publications@quantum-brilliance.com

Simon Yin
Quantum Brilliance Pty Ltd
Sydney, NSW, Australia
publications@quantum-brilliance.com

Michael L. Walker
Quantum Brilliance Pty Ltd
Sydney, NSW, Australia
publications@quantum-brilliance.com

Marcus W. Doherty
Quantum Brilliance Pty Ltd
Canberra, ACT, Australia
publications@quantum-brilliance.com

Maciej Cytowski
Pawsey Supercomputing Research Centre
Kensington, WA, Australia
maciej.cytowski@pawsey.org.au

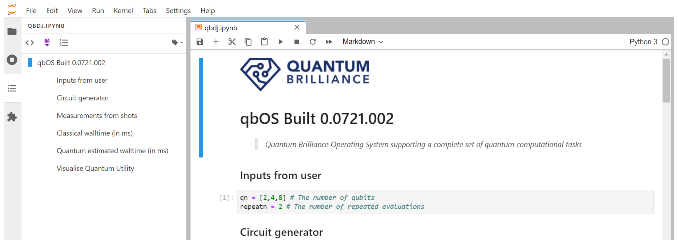
Ugo Varetto
Pawsey Supercomputing Research Centre
Kensington, WA, Australia
ugo.varetto@pawsey.org.au

***Abstract*—Over the last decade, the quantum software and hybrid programming landscape have seen significant progress: high-level language libraries now support a broad range of quantum hardware, scalable classical simulators have appeared, and assembly languages and compilers are approaching maturity. Most notably, the development of XACC, a hardware-agnostic hybrid programming framework, has opened the door for building unique user interfaces, supporting virtually all backends and integration with existing HPC infrastructure. In this proceeding, we introduce qbOS, Quantum Brilliance's XACC-based Python application development framework, which is optimized for the diamond quantum accelerator architecture. On qbOS, developers can build, run, and estimate the runtime of their quantum applications by connecting to third-party classical or Quantum Brilliance's scalable backends. We detail qbOS integration and some initial performance results, identifying a quantum usefulness threshold, on Pawsey's supercomputing systems.**

***Index Terms*—Quantum Computing, Quantum Software, Hybrid Programming, Python, XACC, Quantum Accelerators, QPU**

I. Introduction

The notable advancement in manufacturing noisy quantum processing units (QPUs) on a variety of physical platforms [1] and their immediate availability through cloud platforms, such as IBM Quantum and Amazon Braket, has led to the rapid growth of quantum software [2]. Quantum software enables hybrid quantum-classical programming, compilation, control, hardware runs, co-processing and simulations of desired algorithms. Open-source Python software development kits (SDKs) with access to QPUs, such as `qiskit`, are now well-established in the community. Some SDKs also provide access to efficient classical simulators from state-vector to exa-scalable tensor network backends (for examples refer to AER and TNQVM simulators' documentations), enabling circuit simulations of size 10s to 100s of qubits. These tools have played an important role in finding quantum advantage crossover in celebrated quantum supremacy studies [3], [4] and present immensely valuable educational opportunities.



File Edit View Run Kernel Tabs Settings Help

QBDJ.IPYNB

- qbOS Built 0.0721.002
  - Inputs from user
  - Circuit generator
  - Measurements from shots
  - Classical walltime (in ms)
  - Quantum estimated walltime (in ms)
  - Visualise Quantum Utility

qbdj.ipynb — Markdown — Python 3

QUANTUM BRILLIANCE

qbOS Built 0.0721.002

Quantum Brilliance Operating System supporting a complete set of quantum computational tasks

Inputs from user

```
[1]: qn = [2,4,8] # The number of qubits
     repeatn = 2 # The number of repeated evaluations
```

Circuit generator

Fig. 1. A screenshot example from qbOS UI providing Jupyter notebooks.

Among available platforms, diamond quantum computing [1] is distinct due to offering a pathway toward room-temperature miniaturised QPUs. Quantum Brilliance (QB) [5] has set a target of providing scalable NV-centre-based PCIe-form-factor 50-qubit quantum accelerators in the next few years, which mandates efficient quantum-classical task management and co-processing (i.e. QPUs as heterogeneous accelerators to HPC). The above-mentioned software tools can no longer address the full development and testing requirements of such hardware. The situation greatly improved when the open-source system-level coprocessing-model framework of XACC [6] was introduced. Among many important features such as high-level APIs, frontend language compilation, and extensive backend support, XACC provides a quantum-aware intermediate representation gluing the layers together.

In this proceeding, we introduce Quantum Brilliance Operating System or qbOS, a proprietary XACC-based Python framework. This software is available for licensing and includes R&D versions. Pawsey provides ongoing "Quantum Pioneers" programs[1] granting researchers and SMEs access to qbOS and enabling research on quantum chemistry, finance,

[1]See Pawsey's announcement. Note qbOS was formerly known as Quantum Brilliance Quantum Emulator (QBQE for short). Documentations for the older versions are publicly available via v1.0 and v2.0.

logistics, natural language processing, noise and co-processing modelling. In developing this software, QB has employed a variety of open-source tools, enabling efficient integration with Pawsey's HPC infrastructure as detailed in Sec. II. While `qbOS` is built upon a C++ XACC core, it uses binding tools to present a full-stack Python framework at the frontend: this allows access to a wide range of Python programming functions and provides programming tools to access and control classical simulators and in upcoming versions, for QB's QPUs.

At time of writing, the `qbOS` user interface (UI) can be accessed via either Pawsey's Nimbus or AWS cloud instances in the form of preconfigured Docker containers providing access to Jupyter and Python consoles as shown in Fig. 1. This deployment model can be immediately extended to other cloud systems or individual machines. `qbOS` offers a range of unique features, some not readily available even in its parent XACC library. For example, users can accurately estimate the runtime of algorithms on QB's future generation QPUs: in addition to the runtime of the actual quantum algorithm, `qbOS` accounts for the time needed to transfer the data to the QB hardware controller and to initialise each qubit for each run and to read each qubit at the end. We demonstrate this using a well-known quantum algorithm in Sec. III.

II. Integration with Pawsey's systems

`qbOS` was integrated with Pawsey's HPC and cloud systems to provide both software and services, namely

1) Introductory level: An online learning service through Quantum Brilliance's user acceptance testing and training course covering the basics of XACC kernel-level programming tools and `qbOS` features,
2) Intermediate level: `qbOS` Jupyter UI, (cf. Fig. 1) delivered via Pawsey's Nimbus Cloud, is a cloud service based on OpenStack, used to serve Jupyter from containerised virtual machines, and offers high-level programming tools[2],
3) Advanced level: `qbOS` offers MPI-aware command-line executables for advanced developers, which can run scalable quantum simulations using `TNQVM` backends. `qbOS` was integrated with Singularity containers on two HPC systems at Pawsey suitable for MPI and CUDA-aware executions: Magnus (a Cray XC40 class system) and Topaz (an Nvidia V100 and Infiniband capable system).

III. A quantum application development example

Quantum utility is the threshold where a quantum system outperform a purely classical machine of the similar weight, power, and size in a specific application. For certain edge applications, quantum accelerators may need to outperform handheld devices to become commercially useful and quantum utility provides the right benchmark to identify this threshold.

Here we demonstrate a short Python script, available soon, which upon running with `qbOS` generates comparison (see Fig. 2) of the runtime of QB's hardware to that of a crudely similar size-weight-power Pawsey machine for the Deutsch-Jozsa (DJ) algorithm [7] – a contrived problem where QPUs have known exponential advantage over deterministic classical solvers. The code is readily adaptable to other tasks. The DJ's oracle always implements a constant function and is wrapped according to the simple string $s = 000\ldots$. Our script creates the `OpenQASM` circuit required for the quantum algorithm, classical routines, and plotting functions. We have verified using the `AER` simulator of `qbOS`, where the simulated noise is hard-coded to match the hardware, that the unique correct outcome is achieved up to reported fidelity. Within such fixed constraints and from accurate extrapolation it is clear that QPUs can outperform the classical solver at 35 or more qubits.

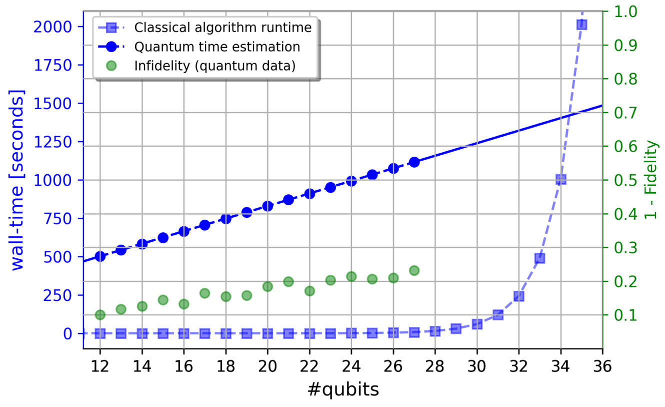


Fig. 2. DJ algorithm quantum utility: runtime comparisons of a classical algorithm (sequential querying, single-thread, Pawsey Nimbus cloud) and estimated quantum hardware (1024 shots, upcoming QB's PCIe-form-factor 50-qubit QPUs, simulated on Pawsey Topaz bare-metal). Green data indicate $(1-\text{Fidelity})$ in-between achieved distribution and the exact one, all counts in s, for noisy quantum simulations. The solid line is a precise linear fit to predict the crossover due to difficulties in exact simulations. Errors are negligible.

The integration of `qbOS` with Pawsey provides a complete quantum computing service: this allowed us to demonstrate that a 35-qubit diamond-chip quantum accelerator outperforms similar-size portable classical devices in DJ applications.

[2]Most notable application here is running complex quantum instructions through a few lines of `qbOS`' high-level Python routines: e.g. "`import qbos as q; t=q.qbqe(); t.qb12; t.random=2; t.run()`" will generate and run a random 12-qubit quantum circuit with depth = 2.

References

[1] Nathalie P. De Leon *et al.*, "Materials challenges and opportunities for quantum computing hardware," Science, vol. 372, no. 6539, April 2021.
[2] David Matthews, "How to get started in quantum computing," Nature **591**, 166-167 (2021).
[3] Frank Arute *et al.*, "Quantum supremacy using a programmable superconducting processor", Nature **574**, 505-510 (2019).
[4] Yulin Wu *et al.*, "Strong quantum computational advantage using a superconducting quantum processor," arXiv:2106.14734 pre-print (2021).
[5] Maurizio Di Paolo Emilio, "Synthetic Diamond Technology Could Make Quantum Practical", EET ASIA, March 2021.
[6] Alexander J. McCaskey, Dmitry I. Lyakh, Eugene F. Dumitrescu, Sarah S. Powers, Travis S. Humble, "XACC: A System-Level Software Infrastructure for Heterogeneous Quantum-Classical Computing," arXiv:1911.02452 pre-print, November 2019 [Documentation available at https://xacc.readthedocs.io/].
[7] David Deutsch and Richard Jozsa, "Rapid solutions of problems by quantum computation," Proceedings of the Royal Society of London A. 439 (1907): 553–558, 1992.