

Fig (1) Overview

Motivation

- Quantum Computing (QC) is more efficient for specific problems than classical computing
- Using QC in a High-Performance Computing (HPC) environment can be useful for those problems → A Hybrid HPC/QC Computing model
- Resource management for such a hybrid model is important!

Background

- PMIx Reference Runtime Environment (PRRTE) →
 - Open-source, scalable, and flexible runtime environment designed for HPC and parallel computing
 - PMIx interface offers interoperability (SLURM, PBS and others)
 - Detects and recovers from process and node failures, ensuring continuous and reliable application execution → Fault Tolerant
 - Supports rapid job launching, dynamic spawning, and process management, facilitating efficient execution of parallel applications
- PRTE Distributed Virtual Machine (PRTE-DVM) →
 - A specific deployment mode of PRRTE that provides a lightweight DVM for running parallel applications
 - The DVM, identified by a unique `<dvm-uri>` is responsible for all MPI tasks

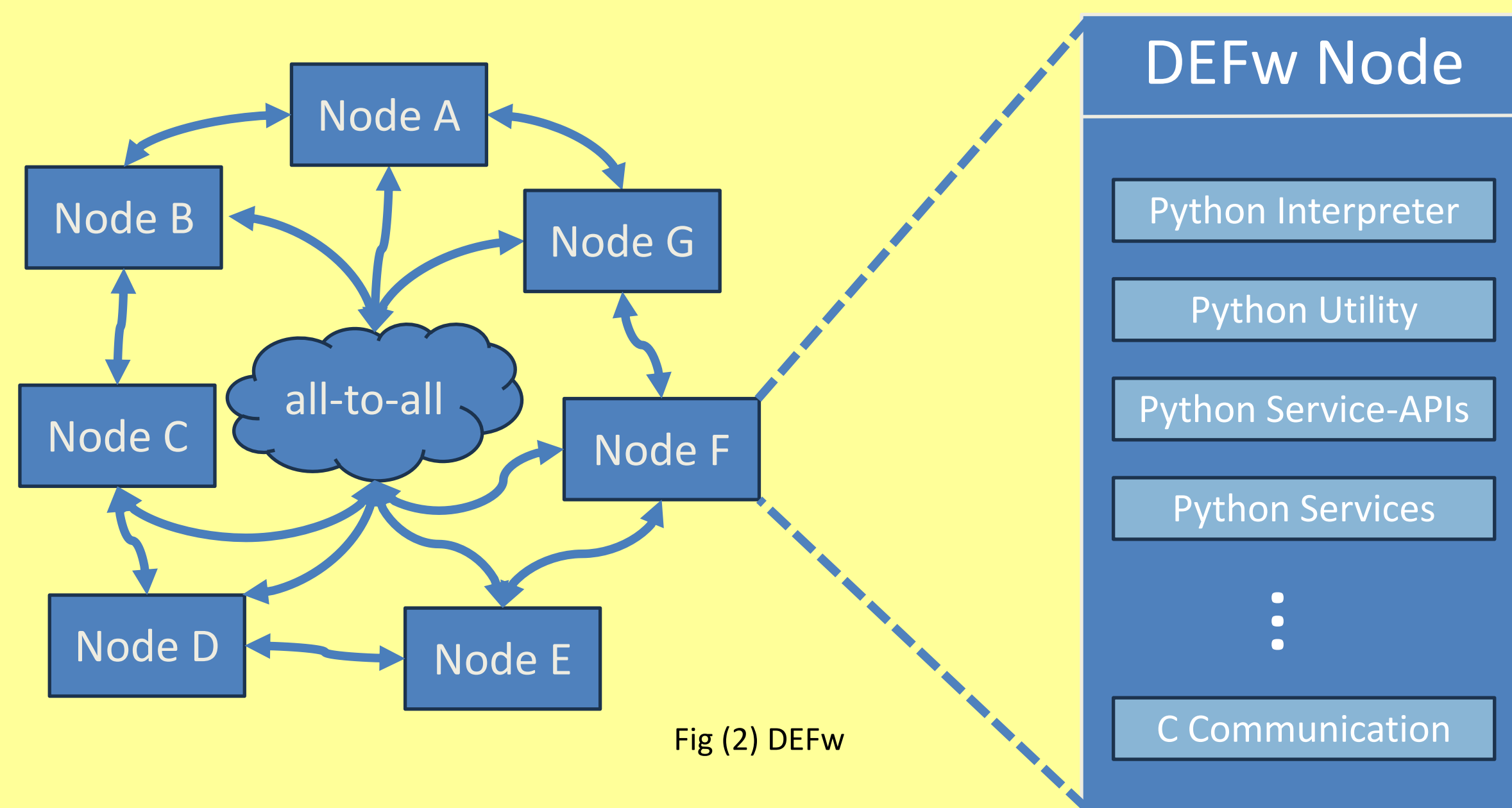


Fig (2) DEFw

- Distributed Execution Framework (DEFw) → A framework to manage the DVM and the QFw components
 - DEFw manages all the nodes in the heterogeneous group and enables inter-service communication via Remote Procedure Calls (RPCs) abstracting it away from each service

QFw Components

- Resource Manager
 - Keeps track of services running in the DEFw environment
 - Respawns services if needed
- QPM-API (Quantum Platform Manager)
 - Service managing quantum resources
 - “Declares” QPM API methods → see Fig (1)
 - Supports both synchronous and asynchronous operations
- QPM-API-Implementation
 - A QPM service (e.g., an HPC Simulator– TN-QVM[1], NWQ-Sim[2]) or actual quantum hardware (not implemented currently) → easily integratable → QFw’s plugin architecture)

qfw python library

- Provides functions to connect to QFw and retrieve a resource manager instance, which then gets an QPM API
- QFWSimulator → Implements Qiskit[3] BackendV2 interface

```
1 # backend_instance = AerSimulator(method="auto")
2 backend_instance = QFWSimulator(simulator="nwqsim")
```

- Pennylane[5] → Leverages the `pennylane-qiskit` plugin

```
1 import pennylane as qml
2 from qfw.qiskit_integration import QFWSimulator
3 # dev = qml.device('qiskit.aer', wires=2)
4 qfw_backend = QFWSimulator(simulator="nwqsim")
5 dev = qml.device('qiskit.remote', wires=2, backend=qfw_backend)
```

QFw Features

- QFw decouples frontends (E.g. Qiskit, Pennylane etc) from backends (E.g, TN-QVM, NWQ-Sim, Qiskit-AER, etc)
- QFw enables application scaling to multiple nodes without needing additional code changes → see Fig (3) below

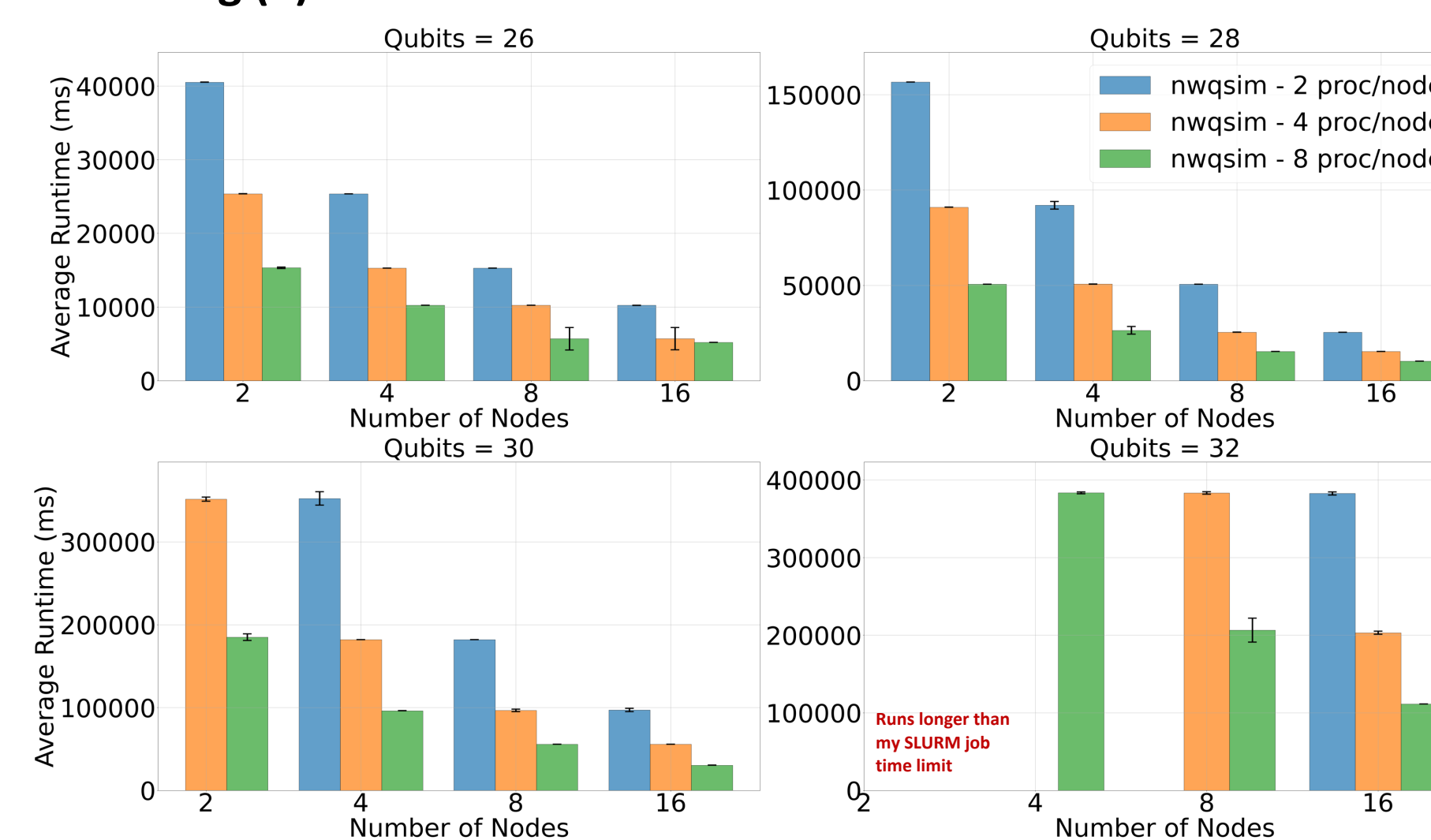


Fig (3) GHZ Scale Tests via QFw with NWQ-Sim

- QFw can also run multiple different circuits in parallel
 - Enables better resource sharing, also useful for certain applications like Distributed-QAOA[6]
- QFw enables hybrid application runs → see Fig (4) & Fig (5)
- QFw enables Hamiltonian simulations with just one line code change compared to qiskit-aer → see Fig (6)
- QFw has some overhead → see Fig (7) → we are working on improving it

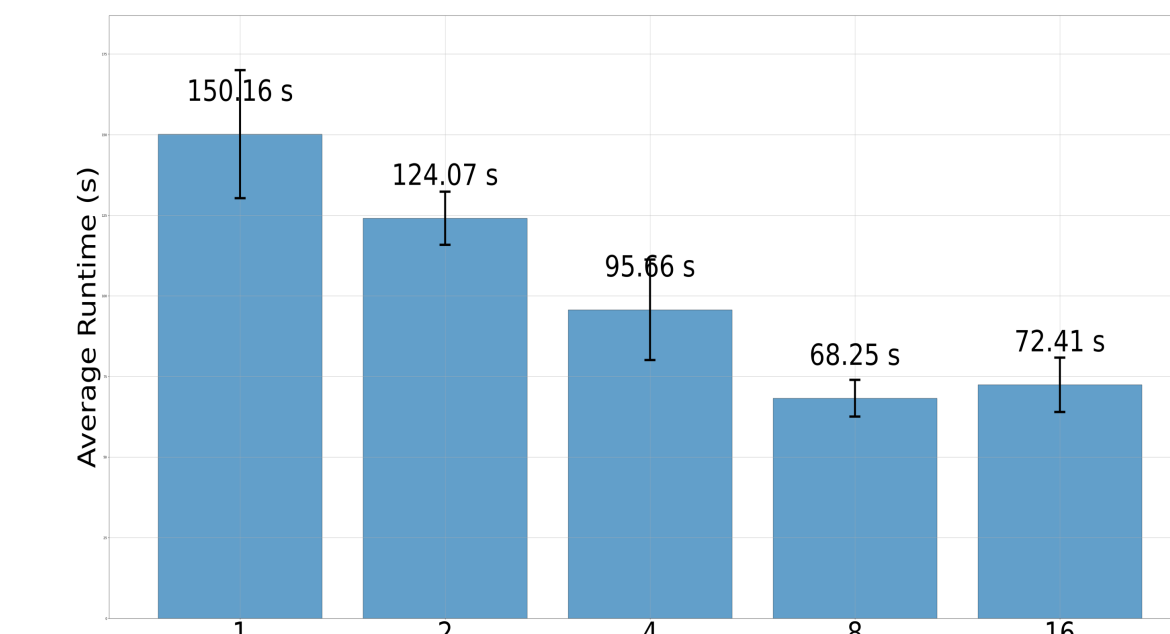


Fig (4) Metamaterial Design[4] QAOA (QUBO 20) via QFw-NWQ-Sim

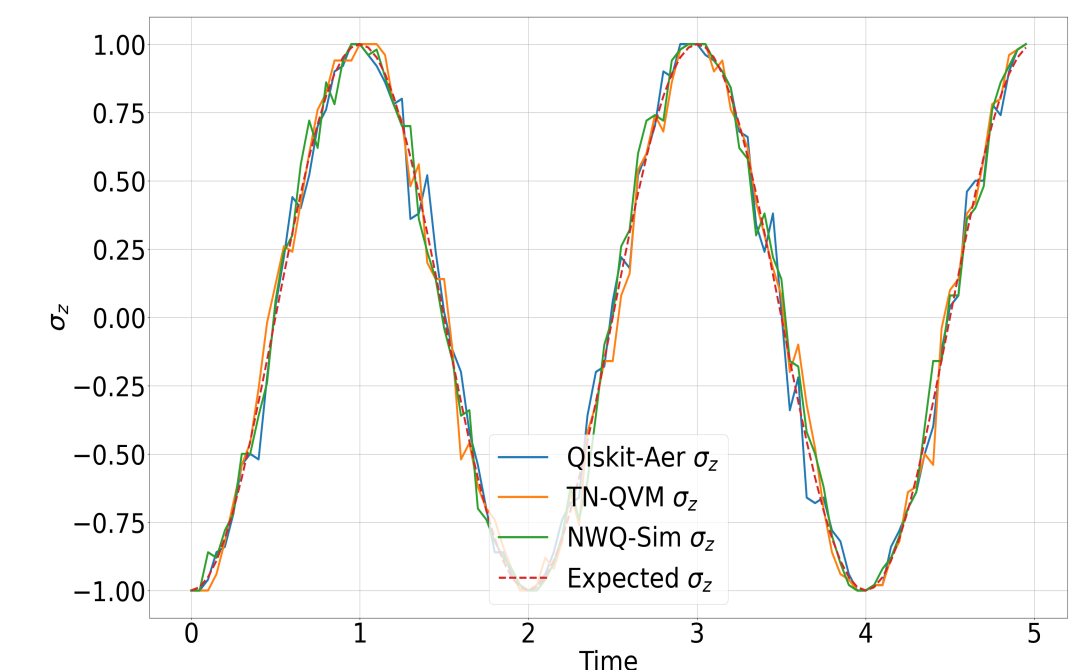


Fig (6) Vacuum Oscillation Hamiltonian (N_Sites = 16) via QFw

QUBO size	Actual Solution	Qiskit-Aer	QFw/NWQ-Sim
4	-1.070966	-1.070966	-1.070967
8	-2.367085	-2.367085	-2.367085
10	-5.102878	-5.102878	-5.102878
16	-3.974519	-3.974519	-3.974519
20	-0.934529	-0.83677	-0.92976

Fig (5) Metamaterial Design[4] QAOA solutions

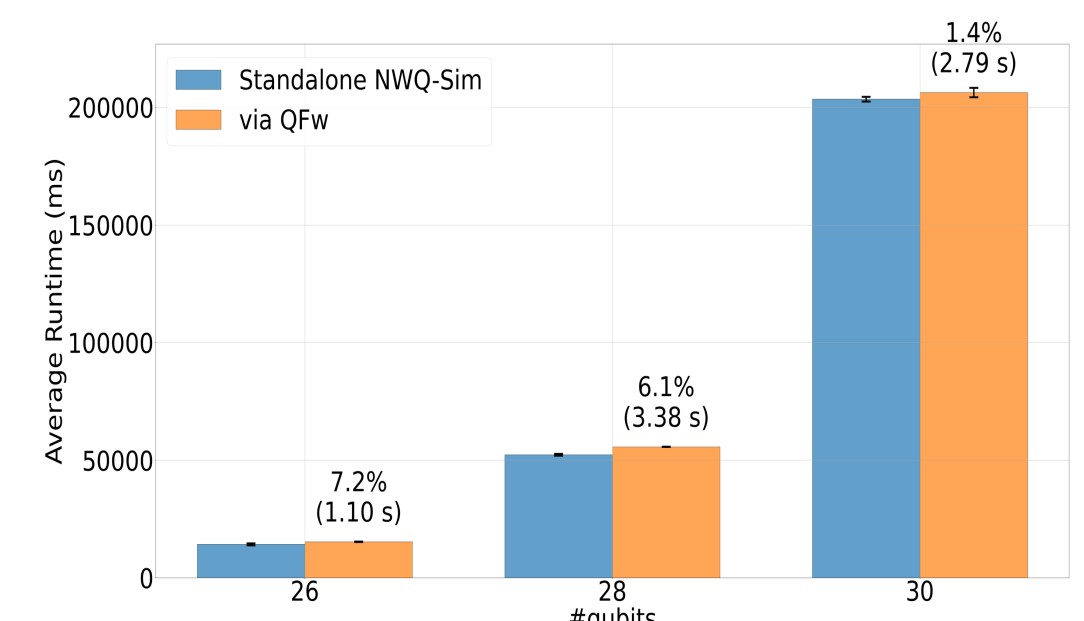


Fig (7) QFw Overhead (GHZ)

Future

- Hardening and optimizing QFw
- More QPM-API-Implementations → other simulators (qiskit-aer, QTensor), actual quantum hardware!
- Different runs in a hybrid workflow could use different QPMs!
- More hybrid applications → E.g., Distributed QAOA[6]
- Auto selection of QPMs based on circuit meta-data
- Smart resource management to avoid wasteful SLURM allocation for QFw (heterogeneous group 1)

References

[1] McCaskey, Alexander J. Tensor Network Quantum Virtual Machine (TNQVM). Computer software. https://www.osti.gov/servlets/purl/1340180_Vers.00. USDOE. 18 Nov. 2016. Web.

[2] Li, A., Fang, B., Granade, C., Prawiroatmodjo, G., Hein, B., Rotteler, M., & Krishnamoorthy, S. (2021). SV-Sim: Scalable PGAS-based State Vector Simulation of Quantum Circuits. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.

[3] Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., & Gambetta, J. M. (2024). Quantum computing with Qiskit. <https://doi.org/10.48550/arXiv.2405.08810>

[4] Kim, S., & Suh, I.-S. (2024). Performance Analysis of an Optimization Algorithm for Metamaterial Design on the Integrated High-Performance Computing and Quantum Systems. <https://arxiv.org/abs/2405.02211>

[5] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M. S., Alonso-Linaje, G., AkashNarayanan, B., Asadi, A., Arrazola, J. M., Azad, U., Banning, S., Blank, C., Bromley, T. R., Cordier, B. A., Ceroni, J., Delgado, A., Matteo, O. D., ... Killoran, N. (2022). Pennylane: Automatic differentiation of hybrid quantum-classical computations. <https://arxiv.org/abs/1811.04968>

[6] Kim, S., Luo, T., Lee, E., & Suh, I.-S. (2024). Distributed Quantum Approximate Optimization Algorithm on Integrated High-Performance Computing and Quantum Computing Systems for Large-Scale Optimization. <https://arxiv.org/abs/2407.20212>