FY2024 MERIT / MERIT-WINGS Self-Directed Joint Research

Development of a Comprehensive Programming Language for Measurement-Based Quantum Computation

Masato Fukushima [A], Yuki Watanabe [B]

[A] Ideguchi Laboratory, Department of Physics, Faculty of Science

[B] Oka Laboratory, Department of Physics, Faculty of Science

Submission Data：2025/01/15

Abstract

This study aims to develop a practical software platform based on the Measurement-Based Quantum Computation (MBQC) theoretical framework, covering all steps from resource-state design to measurement-pattern generation. The software constructed in this research serves as an “MBQC compiler,” converting resource states into measurement patterns that can be executed on real devices or simulators. By integrating with Graphix—an MBQC “machine-language” layer software suite—demonstrations on quantum devices such as IBM and Quandela, as well as simulators like Aer and Perceval, will be possible. This approach will make designing, verifying, and optimizing computational resources on actual quantum devices easier.

About the Authors

[A]Masato Fukushima: Specializes in optical measurements (mid-infrared photothermal microscopy). In this project, contributed to the implementation of the end-to-end workflow, from design of graph state through measurement pattern generation.

[B]Yuki Watanabe: Specializes in condensed matter theory (time-periodic driving of classical stochastic processes, non-Hermitian physics). In this project, contributed to implementing the ZX-Calculus algorithms.

Research Background

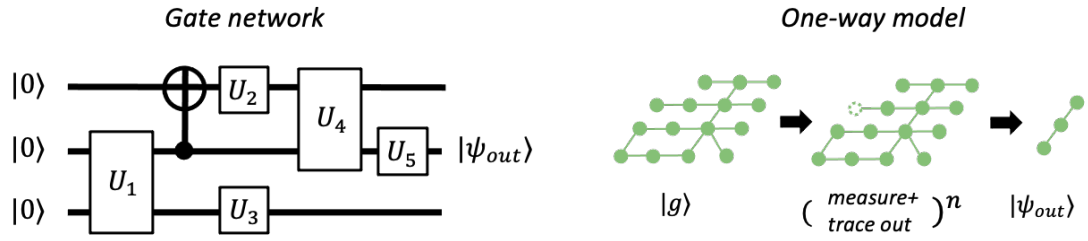


Fig. 1. Overview of the MBQC model. Quantum computation proceeds by the series of single-qubit measurements performed on entangled qubits (graph state). Universal quantum computation can be realized by performing measurements and feedforward operations on an input state [1].

Measurement-Based Quantum Computation (MBQC) is a theoretical framework for universal quantum computation, realized by combining quantum measurements on a graph state—a type of stabilizer state—and feedforward operations that depend on the measurement outcomes (graph state + measurements will be referred to collectively as the “computational graph” in what follows) [1] (see Fig. 1). Unlike the circuit model,

MBQC achieves universal quantum computation using only quantum measurements on a graph state and feedforward operations. This feature is particularly attractive for optical quantum computers, where a universal gate set requires nonlinear processes that are fundamentally probabilistic.

Moreover, MBQC underpins the concept of blind quantum computation [2], a protocol enabling secure cloud-based quantum computing without requiring the client side to possess advanced quantum resources. Consequently, MBQC holds both theoretical and practical promise. In recent years, it has also been shown that ZX Calculus—originally proposed for gate-count optimization in quantum circuits—can be applied to optimize MBQC computational graphs [3]. At a lower, device-oriented level, measurement patterns [4] have been established as a language to describe the sequence of qubit operations. By exploiting the commutation rules of the measurement commands, these measurement patterns can be re-ordered and optimized to align with the execution order on physical devices or simulators.

However, practical software infrastructure that leverages the hierarchical structure and degrees of freedom inherent in the MBQC theoretical framework, and their practical links to real-device execution or simulations, are missing. While software based on quantum circuit models, such as Qiskit, can in principle describe MBQC by incorporating feedforward operations, they require a substantial overhead, making efficient programming of MBQC challenging. MCBeth [5], a software for simulating measurement patterns, only implements the reordering of measurement commands and is not suitable for handling graph transformation operations permitted in MBQC, such as those performed by ZX Calculus. Although ZX Calculus can be handled by PyZX [6], it focuses solely on optimizing computational graphs and lacks a compiler component that converts them into measurement patterns required for execution on actual devices. Given these circumstances, there is a demand for a comprehensive software platform that covers everything from the design of computational graphs to the optimization of measurement patterns.

Research Objectives

In this study, we aim to develop software that comprehensively supports everything from the design of computational graphs to the optimization of measurement patterns, faithfully based on the theoretical framework of Measurement-Based Quantum Computation (MBQC). This software functions as an "MBQC compiler," which intuitively designs resource states in the form of computational graphs—an abstract representation of quantum computations in MBQC—and rapidly and securely converts them into measurement patterns executable on actual devices. By providing a foundation that transforms computational graphs into formats suitable for numerical simulations and real-device executions, this software seeks to facilitate the application and verification of computational resources on quantum devices.

Value and Features of This Research

In this study, we constructed an architecture that hierarchically integrates (1) a layer for optimizing computational graphs, (2) a layer for efficiently performing feedforward operations, and (3) a layer for generating measurement patterns (Fig. 2). This architecture enables the software to function as an "MBQC compiler," directly compiling computational graphs—the abstract representations of computations—into measurement patterns equivalent to an "assembly language" in MBQC. While gate-based models have established software foundations that translate sequences of quantum gates into hardware instructions, MBQC centers around measurements and feedforward operations, making simple circuit transformations inadequate. The software developed in this work can directly handle these MBQC-specific computational resources, which is useful for both theoretical and experimental research. In other words, the software bridges MBQC theory between theoretical concepts and device-executable MBQC language, serving as a research tool to support the design of new quantum algorithms and resource optimization studies.

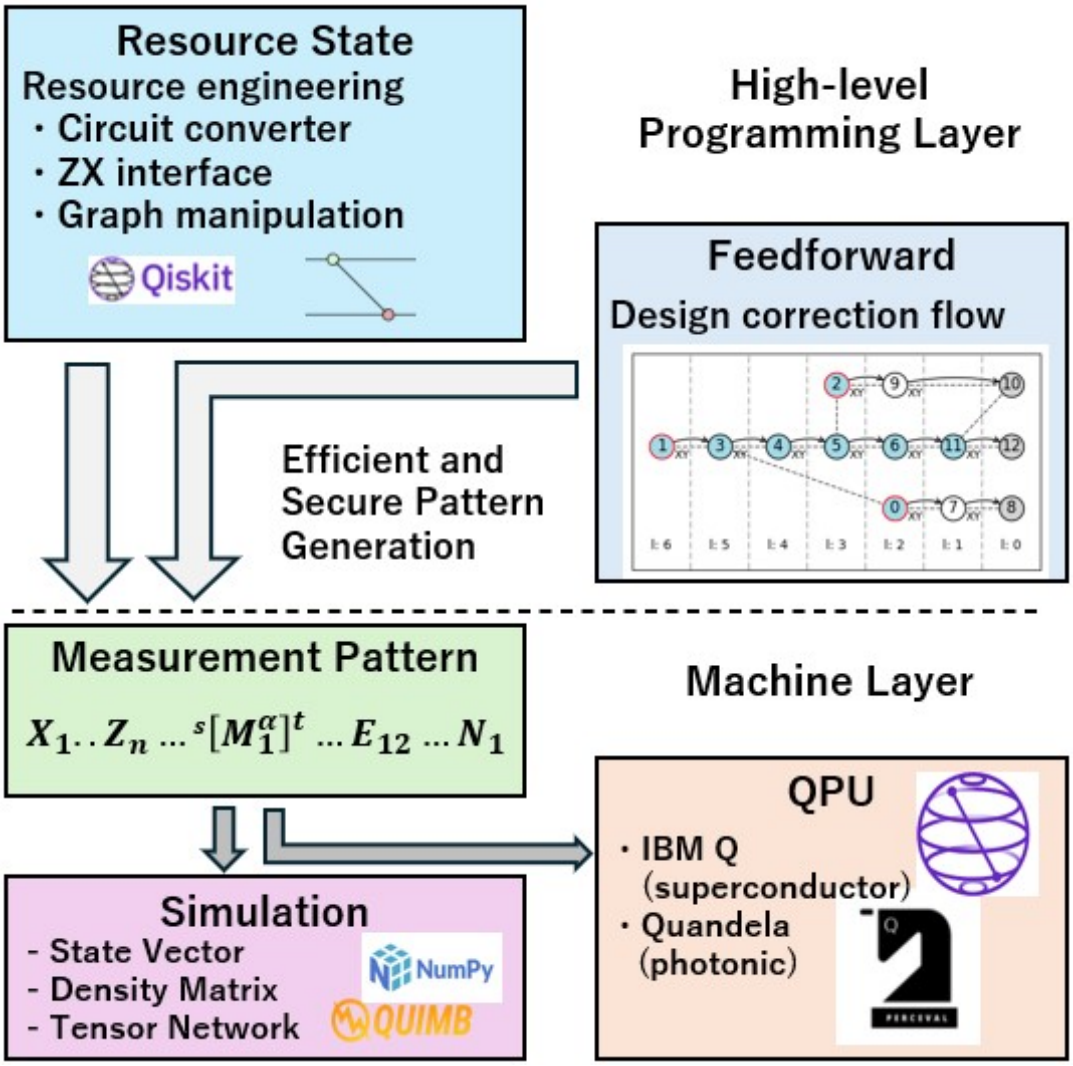


Fig. 2. Overview of the Developed Software. The “High-level Programming Layer” shown here corresponds to the newly developed software. At this upper layer, users program the computational graph and feedforward operations, generate measurement patterns optimized for execution, and support both simulation and real-device implementations.

Results

The developed software is implemented based on the theoretical framework of Backens et al. [4], which represents computational graphs using ZX diagrams and enables optimization processes on them (Fig. 2). ZX diagrams are tensor network-like structures that graphically represent quantum states and operations using nodes and edges, and they can be transformed and simplified based on the theoretical framework of ZX Calculus [3].

In this study, we implemented an optimization process that removes unnecessary Pauli measurement nodes from the computational graphs (Fig. 3, Tab. 1). Pauli measurement nodes represent measurements in the special Pauli bases. It is known that the effects of Pauli measurements satisfying certain conditions can be absorbed as influences on the measurement operations of adjacent nodes, allowing them to be removed from the ZX diagram. This enables the simplification of measurement patterns while maintaining the computational capabilities realized by the original computational graphs.

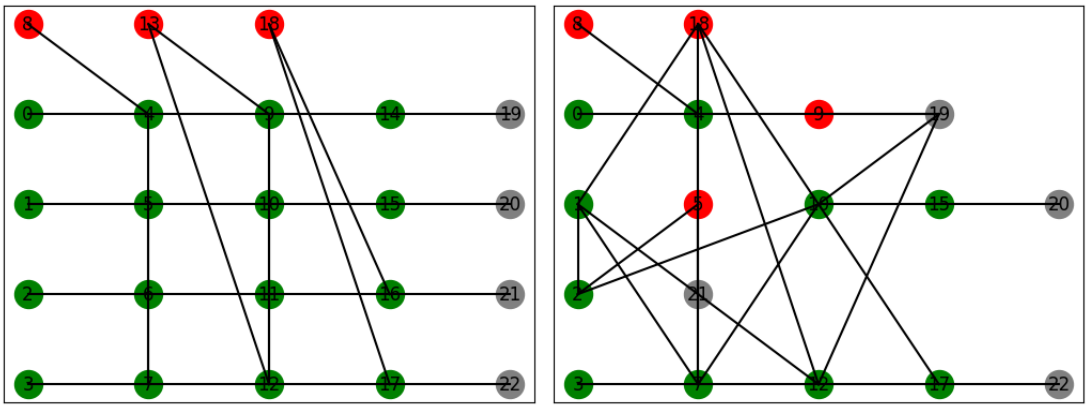


Fig. 3. Initial State (Left) and the Result (Right) After Removing Unnecessary Pauli Measurement Nodes. Green nodes represent measurements in the XY plane, red nodes represent measurements in the YZ plane, and black nodes denote output nodes. Refer to Tab. 1 for the measurement angles.

Tab. 1. Measurement Bases in the Initial State (a) and After Optimization (b). The node numbers with colors are Pauli measurement nodes.

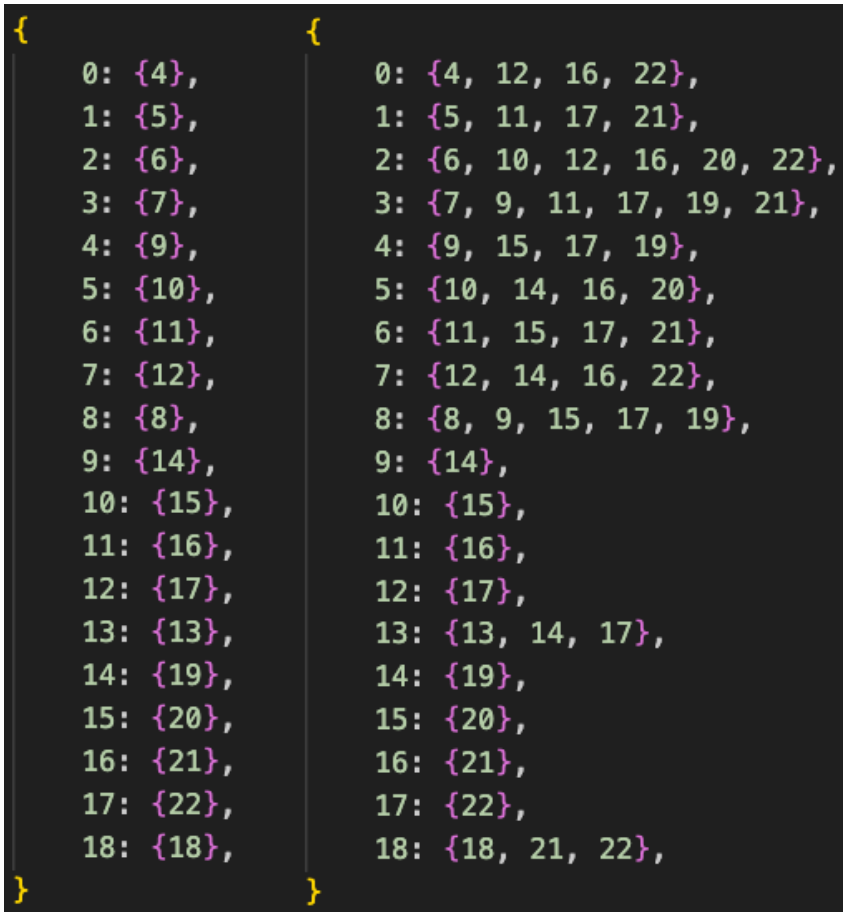
(a)

node	plane	angle $[/\pi]$
0	XY	0
1	XY	$-2/3$
2	XY	$-2/3$
3	XY	-1
4	XY	$-1/3$
5	XY	$-1/3$
6	XY	0
7	XY	$-2/3$
8	YZ	$2/3$
9	XY	$-2/3$
10	XY	$-1/3$
11	XY	0
12	XY	$-1/3$
13	YZ	0
14	XY	0
15	XY	$-1/3$
16	XY	0
17	XY	$-1/3$
18	YZ	$2/3$

(b)

node	plane	angle $[/\pi]$
0	XY	0
1	XY	$4/3$
2	XY	$4/3$
3	XY	-1
4	XY	$5/3$
5	YZ	$5/3$
7	XY	$4/3$
8	YZ	$2/3$
9	YZ	$4/3$
10	XY	$5/3$
12	XY	$5/3$
15	XY	$-1/3$
17	XY	$-1/3$
18	YZ	$2/3$

Furthermore, we restructured the optimization of feedforward operations—which had previously been performed only indirectly at the measurement-pattern level—so that it can now be directly executed at the layer above the measurement pattern (Fig. 2). Concretely, we represent feedforward operations using “Flows” (Causal Flow, Generalized Flow, Pauli Flow), each ensuring determinism and realizing feedforward optimization by transforming these Flows (Fig. 4). While optimizing feedforward operations, we leverage stabilizer-operator identities on the graph state to reduce the computational depth. The results are consistent with those obtained via the “Signal Shifting” method described in [4], indicating that feedforward operations are also amenable to optimization under our framework.



```
{
    0: {4},
    1: {5},
    2: {6},
    3: {7},
    4: {9},
    5: {10},
    6: {11},
    7: {12},
    8: {8},
    9: {14},
    10: {15},
    11: {16},
    12: {17},
    13: {13},
    14: {19},
    15: {20},
    16: {21},
    17: {22},
    18: {18},
}
```

```
{
    0: {4, 12, 16, 22},
    1: {5, 11, 17, 21},
    2: {6, 10, 12, 16, 20, 22},
    3: {7, 9, 11, 17, 19, 21},
    4: {9, 15, 17, 19},
    5: {10, 14, 16, 20},
    6: {11, 15, 17, 21},
    7: {12, 14, 16, 22},
    8: {8, 9, 15, 17, 19},
    9: {14},
    10: {15},
    11: {16},
    12: {17},
    13: {13, 14, 17},
    14: {19},
    15: {20},
    16: {21},
    17: {22},
    18: {18, 21, 22},
}
```

Fig. 4. Results of Feedforward-Operation Optimization. The left figure shows the Flow before optimization (corresponding to the graph state in Fig. 3 left / Tab. 1(a)), and the right figure shows the Flow after optimization.

Moreover, we confirmed that, based on the optimized fundamental graph state and feedforward operations, the system can be transformed (compiled) into an executable measurement pattern (Tabs. 2 and 3). Tab. 2 shows the result of compiling into a measurement pattern without optimization, whereas Tab. 3 shows the result after applying optimization. Without optimization, the measurement pattern consists of 74 commands; this number is reduced to 62 following optimizations. We generated state vectors from both measurement patterns and evaluated the expectation values of random 2×2 Hermitian operators. As shown in Fig. 6, these expectation values remain identical even after optimization, suggesting that the optimization procedure was carried out correctly.

Tab. 2. Measurement Pattern Obtained by Compiling the Initial State Without Feedforward-Operation Optimization. Owing to space constraints, the details are omitted here, but this pattern contains a total of 74 commands.

Command	Contents
Nodes	$4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22$
Edges	$(9, 10), (10, 11), (0, 4), (9, 13), (11, 16), (1, 5), (4, 9), (3, 7),$ $(12, 13), (6, 7), (5, 10), (14, 19), (7, 12), (16, 21), (9, 14),$ $(16, 18), (17, 18), (4, 7), (10, 15), (12, 17), (6, 11), (9, 12),$ $(15, 20), (11, 12), (17, 22), (2, 6), (5, 6), (4, 8)$
Measurements	Node 0: Plane $=$ XY, Angle $= 0$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 1: Plane $=$ XY, Angle $= -2.094$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 2: Plane $=$ XY, Angle $= -2.094$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 3: Plane $=$ XY, Angle $= -3.142$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 4: Plane $=$ XY, Angle $= -1.047$, s-domain $= \{0\}$, t-domain $= \{8, 3\}$
	$\vdots$
	Node 17: Plane $=$ XY, Angle $= -1.047$, s-domain $= \{12\}$, t-domain $= \{18, 7\}$
	Node 18: Plane $=$ YZ, Angle $= 2.094$, s-domain $= \{11, 12\}$, t-domain $= \{\}$
X Operations	Node 19: Domain $= \{14\}$
	Node 20: Domain $= \{15\}$
	Node 21: Domain $= \{16\}$
	Node 22: Domain $= \{17\}$
Z Operations	Node 19: Domain $= \{9\}$
	Node 20: Domain $= \{10\}$
	Node 21: Domain $= \{11\}$
	Node 22: Domain $= \{12\}$

Tab. 3. Measurement Pattern Obtained by Compiling After Feedforward-Operation Optimization of the Initial State. This pattern contains a total of 62 commands.

Command	Contents
Nodes	$4, 5, 7, 8, 9, 10, 12, 15, 17, 18, 19, 20, 21, 22$
Edges	$(5, 21), (1, 2), (0, 4), (9, 19), (3, 7), (2, 10), (12, 19),$ $(5, 7), (7, 12), (17, 18), (2, 5), (4, 7), (10, 15), (1, 18),$ $(12, 17), (1, 21), (4, 19), (7, 10), (15, 20), (17, 22),$ $(1, 7), (4, 8), (10, 19), (12, 21), (12, 18), (5, 18)$
Measurements	Node 0: Plane $=$ XY, Angle $= 0$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 1: Plane $=$ XY, Angle $= 4.189$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 2: Plane $=$ XY, Angle $= 4.189$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 3: Plane $=$ XY, Angle $= -3.142$, s-domain $= \{\}$, t-domain $= \{\}$
	Node 4: Plane $=$ XY, Angle $= 5.236$, s-domain $= \{0\}$, t-domain $= \{\}$
	$\vdots$
	Node 18: Plane $=$ YZ, Angle $= 2.094$, s-domain $= \{0, 1, 2, 3, 4, 5, 7, 8, 12\}$, t-domain $= \{\}$
X Operations	Node 19: Domain $= \{8, 3, 4\}$
	Node 20: Domain $= \{2, 5, 15\}$
	Node 21: Domain $= \{1, 18, 3\}$
	Node 22: Domain $= \{0, 17, 18, 2, 7\}$
Z Operations	Node 19: Domain $= \{9, 5, 7\}$
	Node 20: Domain $= \{8, 10, 4\}$
	Node 21: Domain $= \{0, 2, 5, 7\}$
	Node 22: Domain $= \{1, 3, 4, 8, 12\}$

```
Expectation values for rand_mat
===============================
rand_mat:
 [[1.74138237+0.j         0.86083945-0.12750481j]
 [0.86083945+0.12750481j 1.15586911+0.j         ]]
Before optimization:  1.4486257417605
After optimization:  1.4486257417605004
np.isclose:  True
```

Fig. 5. Verification of Equivalence Between Measurement Patterns Pre- and Post-Optimization. To confirm that both measurement patterns embed the same computation, we compared the expectation values of random 2×2 Hermitian operators for the output states generated by each pattern. The expectation values matched within computational precision, demonstrating the correctness of the optimization process.

Outlook

At the current stage, removing Pauli measurement nodes has enabled the elimination of computations equivalent to Clifford gates. Recently, however, methods have also been proposed for removing certain non-Clifford gates [8]. Incorporating such methods into our implementation may enable handling the broad range of computational models and resource states indicated by MBQC theory, thereby facilitating further optimization and the advanced utilization of computational resources. Additionally, by maintaining compatibility with the measurement patterns of Graphix—a precursor to this project—direct compilations for execution on IBM Quantum and Quandela devices, as well as on Aer and Perceval simulators, are possible via the graphix-ibmq and graphix-perceval submodules.

Looking ahead, one important direction is the development of algorithms that can embed a computational graph into a hardware-feasible graph structure. Since MBQC allows the use of a wider variety of graph geometries than circuit-based models, combining considerations of graph construction costs on actual devices with computational-graph optimization may allow us to propose an execution strategy that exploits MBQC's resource efficiency. Furthermore, in the software constructed in this study, extending feedforward operations would make it possible to implement a compilation approach for measurement patterns that process fault tolerance.

Acknowledgements

We are sincerely grateful for the support and cooperation of many individuals and organizations in conducting this research.

First, we would like to thank the members of Fixstars Amplify, particularly CEO Takuji Hiraoka, for providing the initial opportunity that led to this research, as well as for their substantial support and collaboration. We are also deeply indebted to Dr. Shinichi Sunami (Postdoctoral Researcher at the University of Oxford) and Daichi Sasaki (Kusaka Laboratory, Graduate School of Science, Faculty of Science) for engaging in regular discussions regarding our research objectives and implementation strategies. We further express our appreciation to Sora Shiratani (Todo Laboratory, Graduate School of Science, Faculty of Science) for implementing the fastflow module and offering valuable advice on efficient and high-quality programming.

We are also grateful to the staff at the National Institute of Information and Communications Technology (NICT). This research was selected for the NICT Quantum Camp (探索型人材育成コース) for Fiscal Year 2024, and the advisors provided valuable guidance and financial support, which we deeply appreciate.

Lastly, we extend our thanks to the International Graduate Program for Excellence in Material and Information Science (MERIT-WINGS) at The University of Tokyo for the financial assistance provided to support our research activities.

[1] R. Raussendorf et al., Phys. Rev. Lett. **86**, 22 (2001).
[2] A. Broadbent et al., FOCS (2009).
[3] M. Backens et al., Quantum **5**, 421 (2021).
[4] V. Danos, et al., J. ACM, **54**, 2 (2007).
[5] A. Evans et al., arxiv:2204.10784 (2022).
[6] A. Kissinger et al., EPTCS 318 (2020).
[7] D. E. Browne et al., New J. Phys. **9**, 8 (2007).
[8] A. Kissinger et al., Phys. Rev. A, **102-2**, 102:022406, 8 (2020).