Graphix: develoment and usage updates

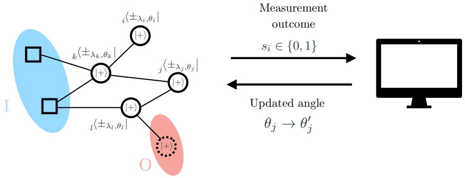
https://github.com/TeamGraphix/graphix/

22 September 2025

Maxime Garnier, Thierry Martinez, Mateo Uldemolins

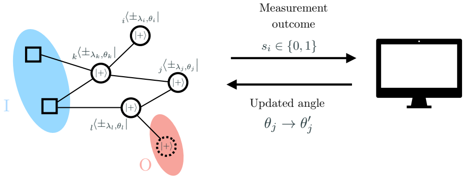
Measurement-based quantum computing

– Principle [RB01]



Measurement-based quantum computing

– Principle [RB01]



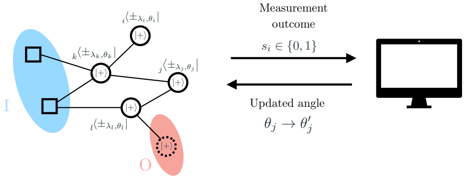
– Commands [DKP07]

- N_i: ancilla node preparation $\otimes \left|+\right\rangle_i$
- E_{ij}: entanglement CZ_{ij}
- $M_i^{\lambda,\alpha}$: measurement $\langle \pm_{\lambda,\alpha}|$
  $\lambda \in \{XY, YZ, XZ\}, \alpha \in [0, 2\pi[$
- X_i^s, Z_i^t: Pauli corrections $s, t \in \{0,1\}$

– Pattern: sequence of commands

Measurement-based quantum computing

– Principle [RB01]



– Commands [DKP07]

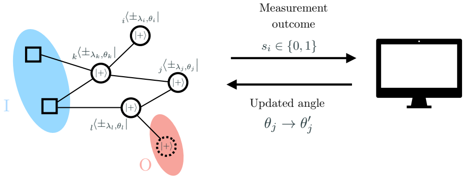
- N_i: ancilla node preparation $\otimes |+\rangle_i$
- E_{ij}: entanglement CZ_{ij}
- $M_i^{\lambda,\alpha}$: measurement $\langle \pm_{\lambda,\alpha}|$
  $\lambda \in \{XY, YZ, XZ\}, \alpha \in [0, 2\pi[$
- X_i^s, Z_i^t: Pauli corrections $s, t \in \{0,1\}$

– Pattern: sequence of commands

– Equivalence to circuit model and universality

Measurement-based quantum computing

– Principle [RB01]



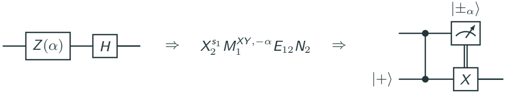
– Commands [DKP07]

- N_i: ancilla node preparation $\otimes |+\rangle_i$
- E_{ij}: entanglement CZ_{ij}
- $M_i^{\lambda,\alpha}$: measurement $\langle \pm_{\lambda,\alpha}|$
  $\lambda \in \{XY, YZ, XZ\}, \alpha \in [0, 2\pi[$
- X_i^s, Z_i^t: Pauli corrections $s, t \in \{0,1\}$

– Pattern: sequence of commands

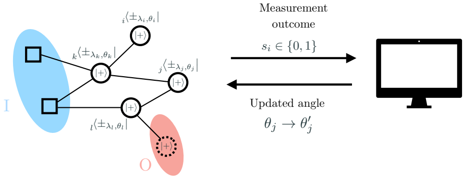
– Equivalence to circuit model and universality

- compilation to $\{J(\alpha) := HZ(\alpha), CNOT\}$ gate set [DKP07]



Measurement-based quantum computing

– Principle [RB01]



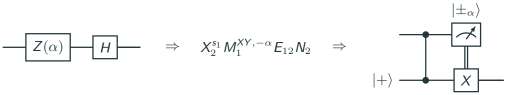
– Commands [DKP07]

- N_i: ancilla node preparation $\otimes\,|+\rangle_i$
- E_{ij}: entanglement CZ_{ij}
- $M_i^{\lambda,\alpha}$: measurement $\langle \pm_{\lambda,\alpha}|$
  $\lambda \in \{XY, YZ, XZ\}, \alpha \in [0, 2\pi[$
- X_i^s, Z_i^t: Pauli corrections $s, t \in \{0,1\}$

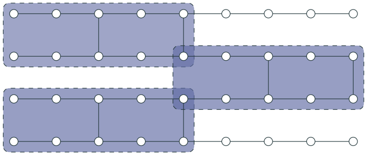
– Pattern: sequence of commands

– Equivalence to circuit model and universality

- compilation to $\{J(\alpha) := HZ(\alpha), CNOT\}$ gate set [DKP07]



- universal graphs: *e.g.* brickwork [BFK09]



Determinism and flow conditions

– Strong determinism: all computation branches are equal up to a global phase

Determinism and flow conditions

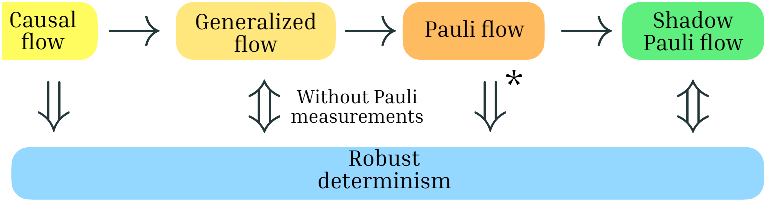
- Strong determinism: all computation branches are equal up to a global phase
- Strongly deterministic patterns implement isometries

Determinism and flow conditions

– Strong determinism: all computation branches are equal up to a global phase
– Strongly deterministic patterns implement isometries
– Robust determinism [PS17]: strong, uniform, stepwise

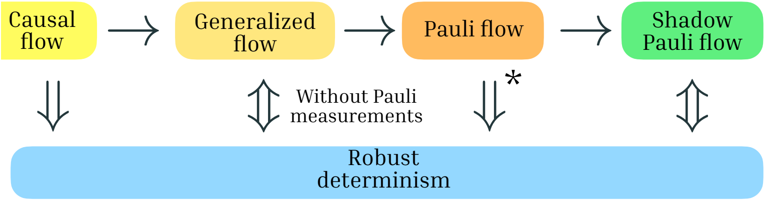
Determinism and flow conditions

- Strong determinism: all computation branches are equal up to a global phase
- Strongly deterministic patterns implement isometries
- Robust determinism [PS17]: strong, uniform, stepwise
- Flow conditions: graphical conditions on open graphs (G, I, O, λ) [DK06, BKMP07, BMBdF^{+}21, MPS22]



Determinism and flow conditions

- Strong determinism: all computation branches are equal up to a global phase
- Strongly deterministic patterns implement isometries
- Robust determinism [PS17]: strong, uniform, stepwise
- Flow conditions: graphical conditions on open graphs (G, I, O, λ) [DK06, BKMP07, BMBdF^{+}21, MPS22]
- Can be found in polynomial time [MP08, BMBdF^{+}21, Sim21, MPS22, MB24]



The Graphix project

– Why MBQC?
  - Interesting primitives (feedforward)
  - Intermediate language for circuit optimisation [BMBdF+21]
  - Applications (network scenarios, native algorithms)

The Graphix project

- Why MBQC?
  - Interesting primitives (feedforward)
  - Intermediate language for circuit optimisation [BMBdF+21]
  - Applications (network scenarios, native algorithms)
- Why an MBQC software?
  - Needed for our own research work
  - No fully satisfactory solution [JD22, EOSR23, BD25]

The Graphix project

- Why MBQC?
  - Interesting primitives (feedforward)
  - Intermediate language for circuit optimisation [BMBdF+21]
  - Applications (network scenarios, native algorithms)
- Why an MBQC software?
  - Needed for our own research work
  - No fully satisfactory solution [JD22, EOSR23, BD25]
- The team (research engineers, researchers, PhD students): Paris - Oxford - Tokyo

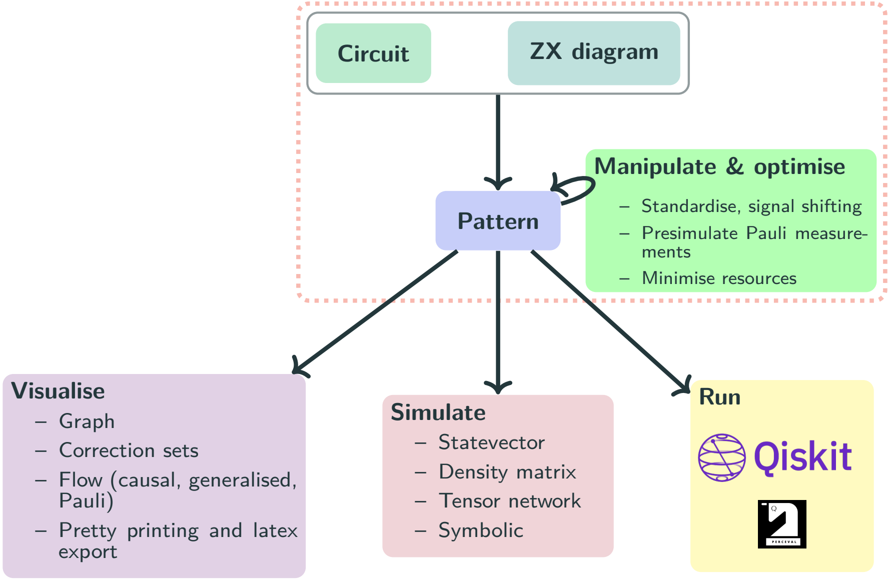
RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Inria

UNIVERSITY OF OXFORD

UTokyo

Structure of Graphix



Highlights

Unitary Foundation Continued support from the Unitary Foundation

– September 2025: Talk at the International Workshop on Quantum Compilation 2025 in Helsinki, Finland. Poster in 2024.
– More and more interest in the software: need to structure a developer and user community

This workshop

- MBQC/infrastructure
- Applications (what we have done with Graphix, what we think can be done)
-
  - MBQC-native variational algorithms and quantum machine learning (M. Leonetti, L. Mantilla)
  - Quantum error correction and fault-tolerance
  - pseudorandomness (D. Markham)
  - verification of QC and quantum cryptography (S. Abdul Sater, M. Inajetovic, L. Music)
  - ...
- Practicals and interaction

Graphix technical updates since last year’s workshop

- Flows
- New features
- Bug fixes
- Developer experience improvements
- Interfaces with other tools

Flows

New Pattern generation from Pauli Flow (#309)

```
og = graphix.opengraph.OpenGraph(graph, inputs,
outputs, ...)
og.to_pattern()
```

Causal flow ? — Yes →

$$\mathcal{P}_{f,G} = \prod_{i\in O^c}^{<} \left(X_{f(i)}^{s_i} Z_{N_G(f(i))\setminus\{i\}}^{s_i} M_i^{\alpha_i} \right) E_G N_{I^c}$$

No ↓

Gflow ? — Yes →

$$\mathcal{P}_{g,G} = \prod_{i\in O^c}^{<} \left(X_{g(i)\setminus\{i\}}^{s_i} Z_{Odd(g(i))}^{s_i} M_i^{\lambda(i)\alpha_i} \right) E_G N_{I^c}$$

No ↓

Pauli flow ? — Yes →

$$\mathcal{P}_{g,G} = \prod_{i\in O^c}^{<} \left(X_{g(i)\cap\{j|j>i\}}^{s_i} Z_{Odd(g(i))\cap\{j|j>i\}}^{s_i} M_i^{\lambda(i)\alpha_i} \right) E_G N_{I^c}$$

Browne et al 2007 New J. Phys. **9** 250

New feature!

New Implementation of $O(N^3)$ Pauli-Flow finding algorithm (#337, under review)

- Algebraic interpretation of Pauli flow (Piotr Mitosek's last year talk!)
  - **Open graph** expressed in terms of M, N ($\mathbb{F}_2$ matrices).
  - **Correction function** extracted from M^{-1}.
  - **Partial order** extracted from NM^{-1}, a DAG.
  - Better **complexity**:
    Gflow: $O(N^4) \to O(N^3)$.
    Pauli flow: $O(N^5) \to O(N^3)$.
  - Improved Graphix module for $\mathbb{F}_2$ linear algebra.

Now we can find **Pauli flow** of open graphs with $|\mathbf{N}| \sim \mathbf{10^3}$ in **seconds** instead of hours!

Mitosek and Backens, 2024, arXiv:2410.23439 ;

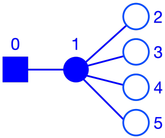
New Pattern and Open-graph composition (#320)

Pattern composition $\mathcal{P}_c = \mathcal{P}_1 \circ \mathcal{P}_2$

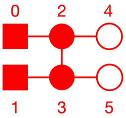
- Pattern concatenation $\circ$ renumbering.
- Directed: output P_1 $\to$ input P_2
- `Pc, mapping = P1.compose(P2, {0: 1})`
- $P_1 = X_1^0 Z_1^0 M_0 E_{0,1} N_0$, $P_2 = X_2^0 Z_2^0 M_0 E_{0,2} N_2$, $\mathcal{P}_c = X_2^1 Z_2^1 M_1 E_{1,2} N_2 X_1^0 Z_1^0 M_0 E_{0,1} N_0$
- *Very useful for MBQC-VQE* (M. Leonetti's talk after the break and this afternoon's discussions)

Open-graph composition

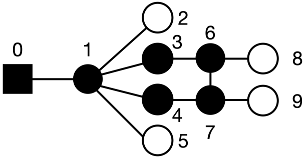
- Graph merging $\circ$ renumbering (more general than pattern composition !).
- `OGc, mapping = OG1.compose(OG2, {0: 3, 1: 4})`



I = [0]
O = [2, 3, 4, 5]



I = [0, 1]
O = [4, 5]



I = [0]
O = [2, 5, 8, 9]

New Parametric patterns and symbolic reasoning (#158): 1/2

Motivations

- Most pattern analyses are **uniform**: they don't care about actual angle values.
  E.g., Transpilation, flows, standardization, depth, space.

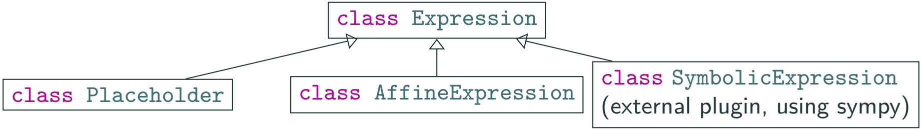
- Various applications where the pattern structure is fixed and where only angles change.
  E.g., Variational algorithms, brickwork universal graph-state.

New

Parametric patterns and symbolic reasoning (#158): 2/2

Rotations in circuits and measures in patterns are now parameterized by values of type `Angle`.

```
Angle: TypeAlias = float | Expression
```



`AffineExpression` encompass Clifford absorption in measurements: *e.g.*, $M_0^{XY,\alpha} \circ C_0^{H\circ S} = M_0^{YZ,\frac{\pi}{2}-\alpha}$.
Placeholders can be substituted with actual values with `subst` or `xreplace` (parallel substitution).

Application (see Marta's talk on VQE, today at 11.45am):

```
class Gadget:
    pattern: Pattern
    angles: tuple[Placeholder, ...]
```

New Transpilers (#142): 1/2 simpler transpiler

Theorem ([DKP07])

MBQC is universal.

Proof.

- $\mathfrak{J}(\alpha)[i \to o] = X_o^{\{i\}} \circ M_i^{XY,-\alpha} \circ E_{i,o} \circ N_o$ *(one fresh ancilla node per* $\mathfrak{J}(\alpha)$*)*
- $\wedge Z(n_1, n_2) = E_{n_1,n_2}$

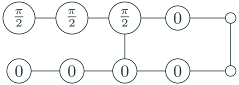
$\square$

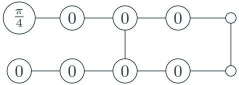
Constructive proof: transpiler from circuits to patterns.
`https://github.com/qat-inria/graphix-jcz-transpiler`

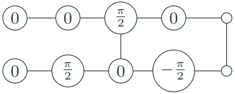
New

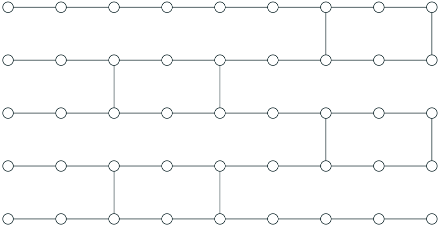
Transpilers (#142): 2/2 brickwork graph-state transpiler

Universal brickwork graph-state transpiler [BFK09]: `https://github.com/qat-inria/gospel`

– $H =$ 

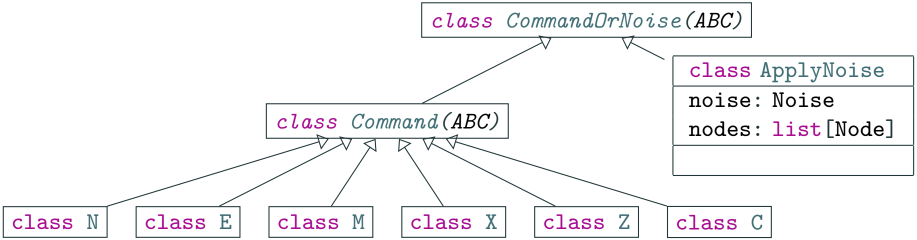
– $T =$ 

– $\wedge X =$ 



Application: Implementation of VBQC with Veriphix (see Sami's talk, tomorrow at 11.45am).

New API for noise models (#271): 1/3 patterns with noise commands

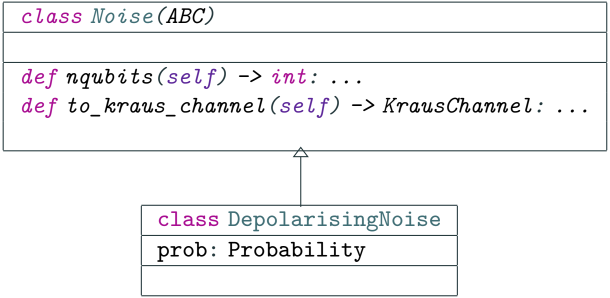


$$\cdots \mathcal{A}_2(\mathcal{N}) \circ M_1 \circ \mathcal{A}_1(\mathcal{N}) \circ \cdots \circ \mathcal{A}_{1,2}(\mathcal{N}) \circ E_{1,2} \circ \cdots \circ \mathcal{A}_1(\mathcal{N}) \circ N_1 \cdots$$

Noise can appear before or after the ideal command and can be applied to other nodes (*e.g.*, neighbors).

New

API for noise models (#271): 2/3 describing noise symbolically



The density-matrix backend applies the noise using the Kraus channel.

Other backends can implement noises using the higher-level noise description (*e.g.*, Stim backend).

New API for noise models (#271): 3/3 noise models as transpilers

```python
class NoiseModel(ABC)

def input_nodes(self, nodes, rng=None) -> list[CommandOrNoise]: ...
def command(self, cmd: CommandOrNoise, rng=None) -> list[CommandOrNoise]: ...
def confuse_result(self, cmd: M, result: Outcome, rng=None) -> Outcome: ...

def transpile(self, sequence, rng=None) -> list[CommandOrNoise]
```

```python
class DepolarisingNoiseModel
```

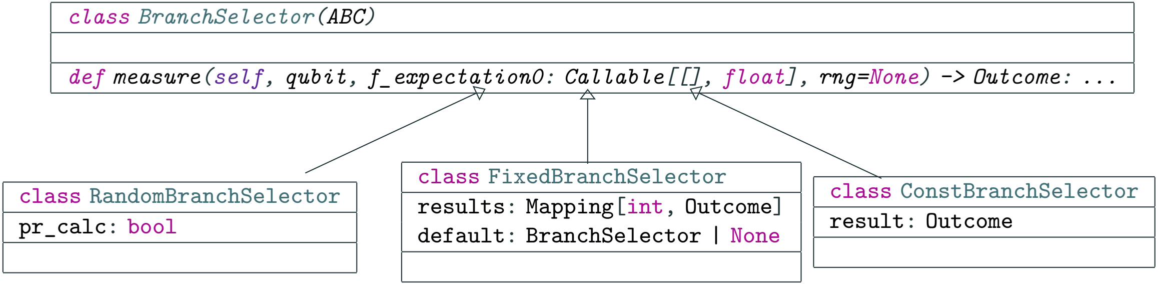
```python
class ComposeNoiseModel
```

```python
class NVCenterNoiseModel
```
Gospel project, afternoons' discussion

Similar to circuit-centered packages:
noise models = transpilers from ideal patterns to noisy patterns.

New API for branch selection (#300)

Branch selector: $\underbrace{\mathbb{N}}_{\text{measured qubit}} \times \underbrace{\mathbb{P}}_{\text{expectation for 0}} \times \underbrace{\mathbb{P}}_{\text{source of randomness}} \to \{0, 1\}.$



Applications:

- Exploring the same branch multiple times (*e.g.*, to compare different backends).
- Complete branch space exploration (*e.g.*, for testing edge cases).

Warning: division-by-zero when exploring 0-probability branches.

Fix Pattern generation from graph-state (#283)

$\mathcal{G} \to$ open graph $\to$ pattern

```
graph: nx.Graph = ...
og = OpenGraph(inside=graph, inputs=[], outputs=graph.nodes, measurements={})
pattern = og.to_pattern()
```

Pattern generation used to fail when measurements $= \emptyset$ (no flow found).

Application: Graph-state preparation (see Marta's talk on VQE, today at 11.45am).

Fix Standardization and local Clifford (#322)

Theorem ([DKP07])

Every MBQC pattern can be rewritten in an equivalent pattern in standard form:

$$\underbrace{Z^* \circ X^*}_{\textit{byproduct corrections}} \circ \underbrace{M^*}_{\textit{one-qubit measurements}} \circ \underbrace{E^* \circ N^*}_{\textit{graph-state preparation}}$$

*(" * " is the Kleene star: 0 or more)*

Some Clifford gates do not commute with $\wedge Z$.
E.g., there is no *ancilla-free* pattern of the form $P \circ E_{1,2}$ that is equivalent to $E_{1,2} \circ C_1^H$.
The theorem does not hold in general in **the MBQC+LC fragment**, *i.e.*, MBQC with the additionnal C_i^c command, where c is an arbitrary one-qubit Clifford gate.

Now, standardization procedure correctly rewrites the pattern in standard form if it is possible (even in presence of some Clifford gates), and **fails if there is no standard form**.

Consequence: Since Pauli presimulation necessarily introduces local Clifford, composing Pauli-presimulated patterns with other patterns may lead to patterns that cannot be standardized.
E.g., $\mathfrak{J}(\alpha)[0 \to 1] = X_1^0 \circ M_0 \circ E_{0,1} \circ N_1 = X_1^0 \circ C_1^H \circ N_1$.

New Pretty-printing (#277)

```
>>> circuit = Circuit(1)
>>> circuit.h(0)
>>> pattern = circuit.transpile().pattern
>>> print(str(pattern))
X(1,{0}) M(0) E(0,1) N(1)                # compact and human readable
>>> print(repr(pattern))
Pattern(                                 # copy-pastable Python code
    input_nodes=[0], cmds=[N(1), E((0, 1)), M(0), X(1, {0})],
    output_nodes=[1]
)
>>> print(pattern.to_unicode())
X₁⁰ M₀ E₀₋₁ N₁                           # even more compact and human readable
>>> print(pattern.to_latex())
\(X_{1}^{0}\,M_{0}\,E_{0,1}\,N_{1}\) # copy-pastable in a LaTeX document
```

$X_1^0\,M_0\,E_{0,1}\,N_1$

Fix Well-typed code base (#288, #302, #308, #312)

Most modules are now well-typed (mypy and pyright compliant).

Programs that use graphix can be type-checked.

Some internal modules, tests and examples are still untyped. Could be a easy and useful first contribution!

Fix pyzx interface (#273)

The following diagram now commutes:

$$
\begin{tikzcd}
\text{circuit} \arrow[r, "\text{embedding}"] \arrow[d, "\text{transpilation}"'] \arrow[drr, "\text{simulation}"] & \text{ZX-diagram} \arrow[r, "\text{rewriting}"] & \text{pattern} \arrow[d, "\text{simulation}"] \\
\text{pattern} \arrow[rr, "\text{simulation}"'] & & \text{state}
\end{tikzcd}
$$

New Symbolic plugin (#158)

Parameterized circuits and patterns can use sympy expressions as angles with the symbolic plugin:
`https://github.com/TeamGraphix/graphix-symbolic`
Symbolic patterns can even be simulated!

```
def test_parameter_circuit_simulation(fx_rng: Generator) -> None:
    alpha = SympyParameter("alpha")
    circuit = Circuit(1)
    circuit.rz(0, alpha)
    result_subs_then_simulate = (
        circuit.subs(alpha, 0.5).simulate_statevector().statevec)
    assert result_subs_then_simulate.psi.dtype == np.complex128
    result_simulate_then_subs = (
        circuit.simulate_statevector().statevec.subs(alpha, 0.5))
    assert np.allclose(
        result_subs_then_simulate.psi, result_simulate_then_subs.psi)
```

The following diagram commutes:

$$\begin{tikzcd} \text{parameterized pattern} \arrow[r, "\text{substitution}"] \arrow[d, "\text{simulation}"'] & \text{pattern} \arrow[d, "\text{simulation}"] \\ \text{symbolic state} \arrow[r, "\text{substitution}"'] & \text{state} \end{tikzcd}$$

New Stim back-end (#150)

https://github.com/thierry-martinez/graphix-stim-backend

Efficient backend for patterns with Pauli measurements only.

Support for many ($\sim$ millions) shots. Support noise models (depolarising noise and Pauli noise).

Can be used to presimulate patterns:

```python
def presimulate_pauli(pattern: Pattern) -> Pattern:
    pattern.move_pauli_measurements_to_the_front()
    pauli_pattern, non_pauli_pattern = cut_pattern(pattern)
    backend = StimBackend()
    measure_method = DefaultMeasureMethod()
    pauli_pattern.simulate_pattern(backend, measure_method=measure_method)
    output_node_set = set(pauli_pattern.output_nodes)
    input_nodes = [node for node in pattern.input_nodes if node in output_node_set]
    result_pattern = backend.to_pattern(input_nodes, non_pauli_pattern.input_nodes)
    result_pattern.results = measure_method.results
    result_pattern.extend(non_pauli_pattern)
    return result_pattern
```

Application: Fast sampling simulation of Clifford computation

New QASM parser (#70)

https://github.com/TeamGraphix/graphix-qasm-parser

```
from graphix_qasm_parser import OpenQASMParser
s = """
    include "qelib1.inc";
    qubit q;
    rz(5*pi/4) q;
"""
parser = OpenQASMParser()
circuit = parser.parse_str(s)
pattern = circuit.transpile().pattern
print(pattern)
```

The parser itself is automatically extracted from the OpenQASM specification (antlr4).

Applications: Benchmarking, comparisons with other frameworks.

Device interfaces (#261)

Qiskit backend: `https://github.com/TeamGraphix/graphix-ibmq`

Perceval backend: see Luka's talk tomorrow at 10.45.

Conclusion & practicalities

– Graphix

– Practicalities

  - Coffee breaks are on the 3rd floor
  - Lunche are on the 3rd floor (menu will be passed around)
  - Dinner at 7:30 pm across the street

Acknowledgements

Thank you!

Contributors
Sami Abdul Sater (Paris), William Cashman (Oxford), Masato Fukushima (Tokyo), Maxime Garnier (Paris), Benjamin Guichard, Thierry Martinez (Paris), Rajarsi Pal (Paris), Sora Shiratani (Tokyo), Shinichi Sunami (Oxford), Mateo Uldemolins (Paris)

Unitary Foundation

QuanTiP

FRANCE HYBRID HPC QUANTUM INITIATIVE

References I

- Greg Bowen and Simon Devitt. tūQ: a design and modelling tool for cluster-state algorithms, 2025.
- Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Universal blind quantum computation. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, page 517–526. IEEE, October 2009.
- Daniel E. Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250, aug 2007.
- Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. There and back again: A circuit extraction tale. *Quantum*, 5:421, March 2021.
- Vincent Danos and Elham Kashefi. Determinism in the one-way model. *Phys. Rev. A*, 74:052310, Nov 2006.
- Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus, 2007.
- Aidan Evans, Seun Omonije, Robert Soulé, and Robert Rand. MCBeth: A Measurement-based Quantum Programming Language. In *2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE)*, pages 1–8, 2023.

References II

- Bus Joost and Radoica Draškić. Measurement-based quantum computation, 2022. https://pennylane.ai/qml/demos/tutorial_mbqc, accessed 10 July 2025.
- Piotr Mitosek and Miriam Backens. An algebraic interpretation of Pauli flow, leading to faster flow-finding algorithms, 2024.
- Mehdi Mhalla and Simon Perdrix. Finding optimal flows efficiently. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 857–868, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- Mehdi Mhalla, Simon Perdrix, and Luc Sanselme. Shadow Pauli Flow: Characterising Determinism in MBQCs involving Pauli Measurements. *arXiv:2207.09368*, 2022.
- Simon Perdrix and Luc Sanselme. Determinism and computational power of real measurement-based quantum computation. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory*, pages 395–408, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
- Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001.

References III

Will Simmons. Relating Measurement Patterns to Circuits via Pauli Flow. In Chris Heunen and Miriam Backens, editors, Proceedings 18th International Conference on *Quantum Physics and Logic*, Gdansk, Poland, and online, 7-11 June 2021, volume 343 of *Electronic Proceedings in Theoretical Computer Science*, pages 50–101. Open Publishing Association, 2021.