Womanium Global Quantum + AI Project

Team Name:

Qoherence

Team Members:

Katayoun Emadzadeh; Mahla Moridi Farimani

Womanium Quantum+AI 2024

Content

- **Introduction to the Paper**
  - **Title:** Evidence for the Utility of Quantum Computing Before Fault Tolerance
- **Introduction to the Trotter Algorithm**
- **Toy Problem Overview**
  - Algorithm Building Blocks
- **Circuit Enlargement**
- **Optimization Techniques**
- **Acknowledgments**

Introduction to the Paper

Article

Evidence for the utility of quantum computing before fault tolerance

https://doi.org/10.1038/s41586-023-06096-3

Received: 24 February 2023

Accepted: 18 April 2023

Published online: 14 June 2023

Open access

Check for updates

Youngseok Kim[1,6] ✉, Andrew Eddins[2,6] ✉, Sajant Anand[3], Ken Xuan Wei[1], Ewout van den Berg[1], Sami Rosenblatt[1], Hasan Nayfeh[1], Yantao Wu[3,4], Michael Zaletel[3,5], Kristan Temme[1] & Abhinav Kandala[1] ✉

Quantum computing promises to offer substantial speed-ups over its classical counterpart for certain problems. However, the greatest impediment to realizing its full potential is noise that is inherent to these systems. The widely accepted solution to this challenge is the implementation of fault-tolerant quantum circuits, which is out of reach for current processors. Here we report experiments on a noisy 127-qubit processor and demonstrate the measurement of accurate expectation values for circuit volumes at a scale beyond brute-force classical computation. We argue that this represents evidence for the utility of quantum computing in a pre-fault-tolerant era. These experimental results are enabled by advances in the coherence and calibration of a superconducting processor at this scale and the ability to characterize[1] and controllably manipulate noise across such a large device. We establish the accuracy of the measured expectation values by comparing them with the output of exactly verifiable circuits. In the regime of strong entanglement, the quantum computer provides correct results for which leading classical approximations such as pure-state-based 1D (matrix product states, MPS) and 2D (isometric tensor network states, isoTNS) tensor network methods[2,3] break down. These experiments demonstrate a foundational tool for the realization of near-term quantum applications[4,5].

Introduction to the Paper

Model: Simulating Quantum Dynamics Using the 2D Transverse-Field Ising Model and Trotterization

Hamiltonian:

$$H = -\sum_{\langle i,j \rangle} J Z_i Z_j + h \sum_i X_i$$

Z and X: Pauli Operators
J: the coupling constant between nearest-neighbor spins
h: the global transverse field

Trotterization: The time evolution of the Hamiltonian is simulated using first-order Trotter decomposition, which breaks down the evolution into discrete time steps:

$$e^{-iH\delta t} \approx \prod_{\langle i,j \rangle} e^{iJ\delta t Z_i Z_j} \prod_i e^{ih\delta t X_i}$$

This process allows the simulation of quantum dynamics over time by alternating between the ZZ interactions and X rotations.

Quantum Processor: The experiments were conducted on IBM's 127-qubit Eagle processor. This processor features a heavy-hexagonal qubit connectivity and advanced coherence properties, allowing for deep quantum circuits involving thousands of CNOT gates.

Introduction to the Trotter Algorithm

What is Trotterization?

Basically, it a method to approximate the exponential of a sum of non-commuting operators by breaking it into a product of exponentials of the individual operators. It's based on the Trotter-Suzuki decomposition.

Global sensitivity analysis for optimization of the Trotter-Suzuki decomposition

Alexey N. Pyrkov,[1,*] Yurii Zotov,[2] Jiangyu Cui,[3] and Manhong Yung[4]

[1] *Institute of problems of chemical physics RAS, Acad. Semenov av. 1, Chernogolovka, Moscow region, Russia, 142432*
[2] *Huawei Russian Research Institute, Huawei Technologies Co. Ltd., Moscow, Russia, 121614*
[3] *Central Research Institute, Huawei Technologies, Shenzhen 518129, China*
[4] *Data Center Technology Laboratory, Huawei Technologies Co Ltd, 115371 Shenzhen, Guangdong, China*

The Trotter-Suzuki decomposition is one of the main approaches for realization of quantum simulations on digital quantum computers. Variance-based global sensitivity analysis (the Sobol method) is a wide used method which allows to decompose output variance of mathematical model into fractions allocated to different sources of uncertainty in inputs or sets of inputs of the model. Here we developed a method for application of the global sensitivity analysis to the optimization of Trotter-Suzuki decomposition. We show with a proof-of-concept example that this approach allows to reduce the number of exponentiations in the decomposition and provides a quantitative method for finding and truncation 'unimportant' terms in the system Hamiltonian.

https://arxiv.org/abs/2101.03349v1

Introduction to the Trotter Algorithm

The simplest first-order Trotter-Suzuki decomposition can be represented as

$$e^{x(A+B)} \approx e^{xA} e^{xB} + O(x^2), \qquad (1)$$

where x - small parameter and A, B - non-commuting operators $[A, B] \neq 0$.

In order to obtain the second–order expansion, it is possible to represent both sides of the equation (1) in the following form:

$$e^{x(A+B)} = I + x(A+B) + \frac{1}{2}x^2(A+B)^2 + O(x^3) =$$

$$I + x(A+B) + \frac{1}{2}x^2\left(A^2 + AB + BA + B^2\right) + O(x^3),$$

$$e^{xA}e^{xB} = \left(I + xA + \frac{1}{2}x^2A^2 + O(x^3)\right) \times$$

$$\left(I + xB + \frac{1}{2}x^2B^2 + O(x^3)\right) =$$

$$I + x(A+B) + \frac{1}{2}x^2\left(A^2 + 2AB + B^2\right) + O(x^3). \qquad (2)$$

https://arxiv.org/abs/2101.03349v1

Related papers:
https://arxiv.org/abs/2310.13296v2
https://arxiv.org/abs/2109.07987v1

Toy Problem Overview

- **Key Concepts in this Quantum Algorithm:**
  - **Quantum Circuit**
  - **Qubits**
  - **RZZ Gates:** Represent the interaction between two qubits with the term of ZZ in Hamiltonian.
  - **RX Gate:** Represents the rotation around X-axis in Bloch sphere.
  - **Trotter steps**

- **Algorithm Building Blocks:** In this section, we will implement a toy problem from the mentioned paper based on the Suzuki-Trotter algorithm. To do so, we will use the Classiq platform for quantum algorithms.
  - **Trotter Algorithm**
  - **Quantum Circuit Design**
  - **Simulation Execution**
  - **Analysis of Results**

Toy Problem Overview

Trotter Algorithm:

Here we use the Suzuki-Trotter function in **Classiq IDE platform** (QMod) for this Hamiltonian.

$$H = -\sum_{\langle i,j \rangle} J Z_i Z_j + h \sum_i X_i$$

The **Suzuki-Trotter** function in **Classiq platform**, includes these parts:

allocate(3, qba): to initialize the qubits (here 3 qubits named as qba)

PauliTerm: we can expand the Hamiltonian (contains the Pauli Gates)

Coefficients: for out Hamiltonian are J and h.

x, 4, 1, qba:

x = 0.1 :time step size

4 = Trotter steps*

1 = scaling factor (total evolution time)

qba: qubit register

* The number of trotter steps should be even which ensures the symmetric form of the Trotterization, reducing errors and improving accuracy.

```
qfunc main(a: real, x: real, output qba: qbit[]) {
  allocate(3, qba);
  suzuki_trotter([
    PauliTerm {
      pauli=[
        Pauli::X,
        Pauli::X,
        Pauli::Z
      ],
      coefficient=a
    },
    PauliTerm {
      pauli=[
        Pauli::Y,
        Pauli::X,
        Pauli::Z
      ],
      coefficient=0.5
    }
  ], x, 4, 1, qba);
}
```

Toy Problem Overview

```
qfunc main(output qba: qbit[]) {
  allocate(8, qba);


  suzuki_trotter([
    // ZZ interactions for 8 qubits
    PauliTerm {
      pauli=[
        Pauli::Z,
        Pauli::Z,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I
      ],
      coefficient=1
    },
  ...
      // X gates
      PauliTerm {
        pauli=[
          Pauli::X,
          Pauli::I,
          Pauli::I,
          Pauli::I,
          Pauli::I,
          Pauli::I,
          Pauli::I,
          Pauli::I
        ],
        coefficient=0.5
      },
      .
      .
      .
      ], 0.1, 4, 1, qba);
    }
```

First, we implemented **the Suzuki-Trotter algorithm in Classiq Platfprm (Qmod)** **for 8 qubits** and synthesized in in this configuration:

Synthesis Configuration

Constraints

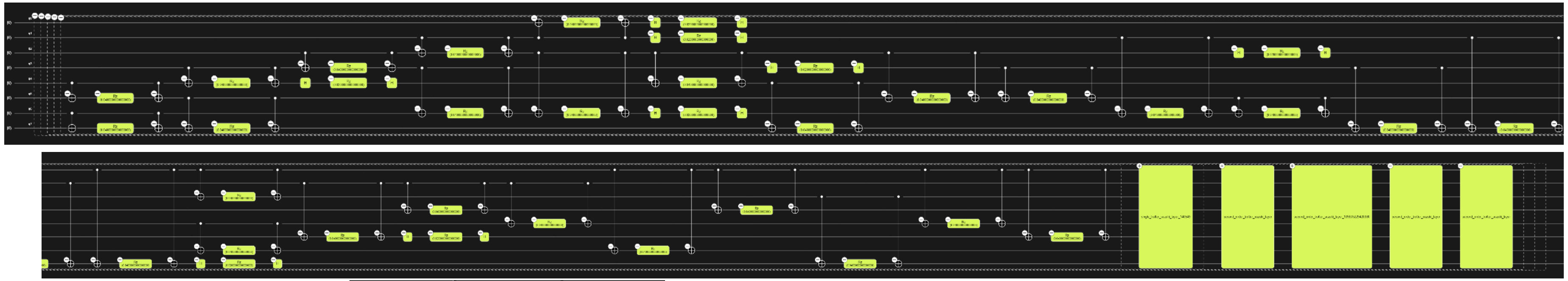
Max Width

Max Depth

Max Gate Count

Optimization Parameter

Here is the quantum circuit:



Transpiled Info | Program Info | Data

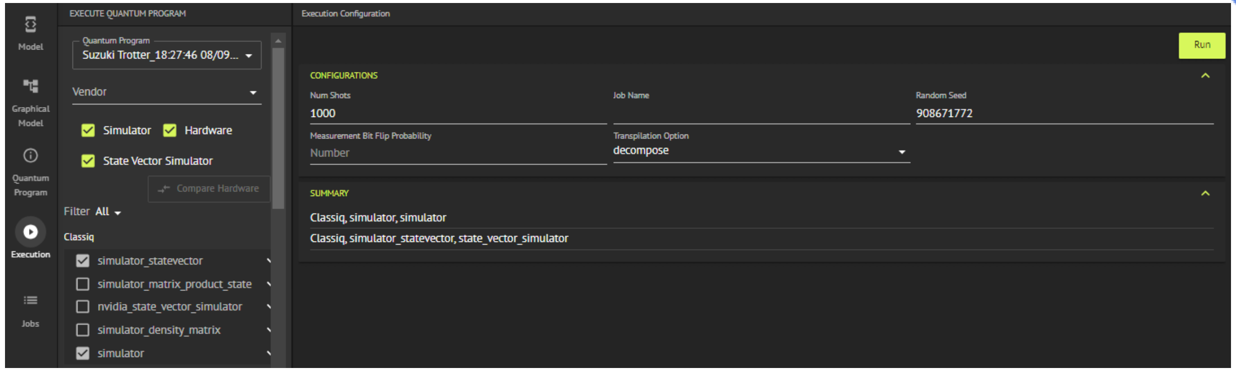
Backend name: Default
Depth: 453
Width: 8

Gate count
U : 292
CX : 524

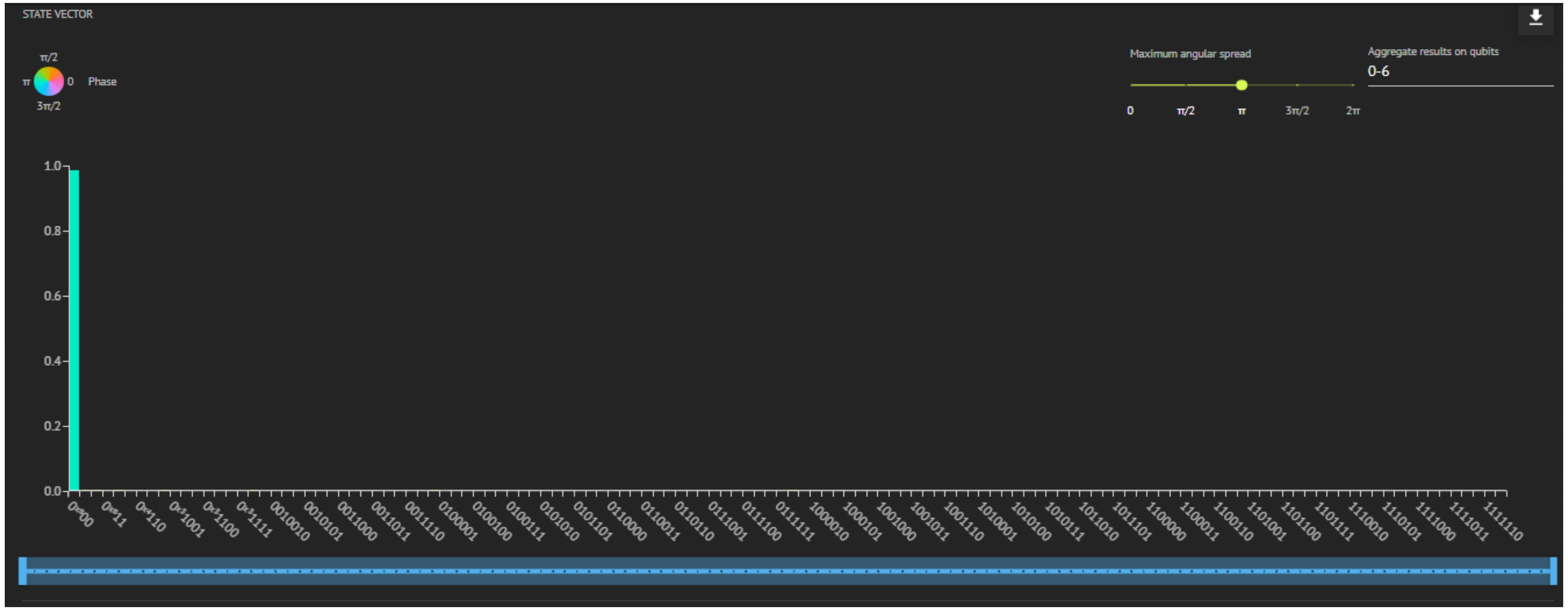
Hardware details
Basis Gates: cy, y, p, cx, cz, rx, ry, u2, t, r, sdg, sx, u1, z, tdg, id, u, x, s, h, sxdg, rz
Connectivity Map:
null
Symmetric Connectivity: True

Toy Problem Overview

Then, we execute it,
and run it on Classiq platform Simulator:



The result:



Toy Problem Overview

It also provides information of state vectors, magnitudes, and phases for result qubits. Here is a part of this information:

Result: the bitstring representation of the quantum state after measurement (e.g., "11111111").

State Vector: the complex amplitude of the quantum state, represented in the form (a + bi).

Magnitude: the absolute value of the complex amplitude, which indicates the probability amplitude.

Phase: the phase of the complex amplitude, expressed as a fraction of (pi).

Results

Rows per page: 100 1–100 of 256

Result	State Vector	Magnitude ↑	Phase
11111111	(-1.074e-11 + 3.476e-11j)	3.638e-11	19π/12
11111011	(-6.271e-10 - 3.706e-10j)	7.284e-10	π/6
11110111	(-6.304e-10 - 3.725e-10j)	7.322e-10	π/6
11011111	(-6.31e-10 - 3.719e-10j)	7.325e-10	π/6
10111111	(-6.319e-10 - 3.731e-10j)	7.338e-10	π/6
01111111	(-6.327e-10 - 3.73e-10j)	7.345e-10	π/6
11111101	(-6.342e-10 - 3.742e-10j)	7.364e-10	π/6
11101111	(-6.345e-10 - 3.746e-10j)	7.368e-10	π/6
11111110	(-6.373e-10 - 3.763e-10j)	7.401e-10	π/6
01111011	(8.539e-9 - 1.16e-8j)	1.44e-8	17π/24
11011011	(8.539e-9 - 1.16e-8j)	1.44e-8	17π/24
11110011	(8.557e-9 - 1.159e-8j)	1.441e-8	17π/24
11010111	(8.556e-9 - 1.161e-8j)	1.442e-8	17π/24
10111011	(8.565e-9 - 1.162e-8j)	1.443e-8	17π/24
11101011	(8.574e-9 - 1.161e-8j)	1.444e-8	17π/24
01110111	(8.58e-9 - 1.164e-8j)	1.446e-8	17π/24
10011111	(8.568e-9 - 1.165e-8j)	1.446e-8	17π/24
11111001	(8.594e-9 - 1.165e-8j)	1.448e-8	17π/24
11111010	(8.604e-9 - 1.164e-8j)	1.448e-8	17π/24
11110101	(8.605e-9 - 1.165e-8j)	1.448e-8	17π/24

Toy Problem Overview

- **The results:**

  Magnitude Distribution: The magnitude column shows probability amplitudes for each 8- qubit state, ranging from 0 to 2.68×10^{-7}, with an average of 1.63×10^{-7}.

  Phase Information: Phase is given in π, essential for understanding interference patterns in quantum states.

- **Noise Consideration:**

  Low Magnitudes: Some states have low or zero magnitudes, meaning they are unlikely to be observed, suggesting minimal noise impact.

  State Vector Complexity: The state vectors are complex numbers, indicating quantum interference. Noise could disrupt these interferences, altering the phase or magnitude.

  Uniformity and Spread: The results show some states dominate, implying noise hasn't caused significant decoherence, and the quantum state remains well-preserved.

Toy Problem Overview

- **Analysis the results:**

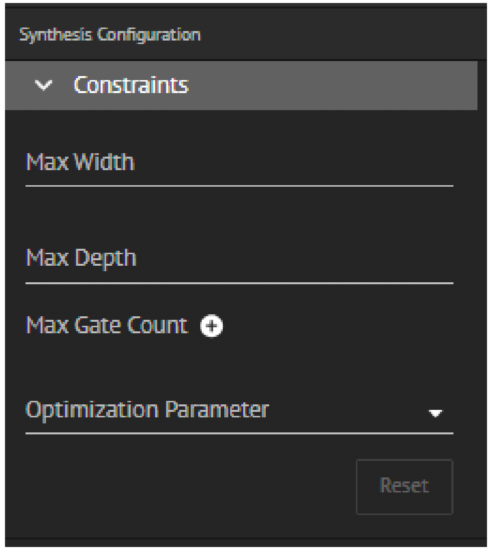
  The results suggest that the system is operating in a relatively low-noise environment, as indicated by the clear distinction in magnitude between different states and the presence of well-defined phases.

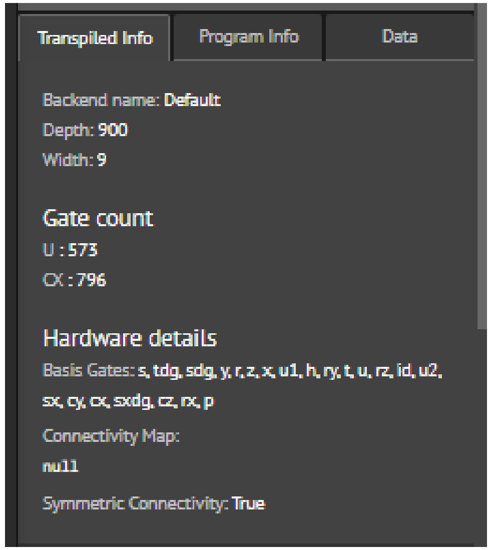
  In an actual noisy quantum device, we expect the magnitudes to be more uniform across different states, and phases might become randomized, leading to less constructive interference and more decoherence.

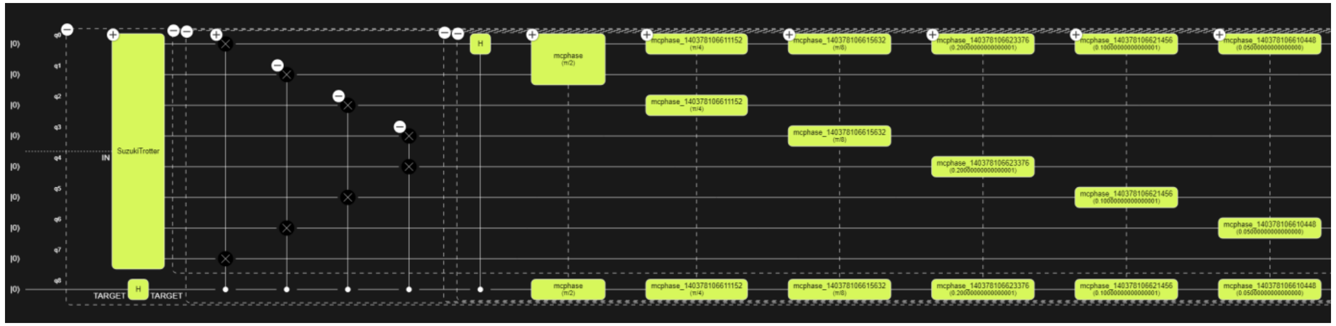
Toy Problem Overview

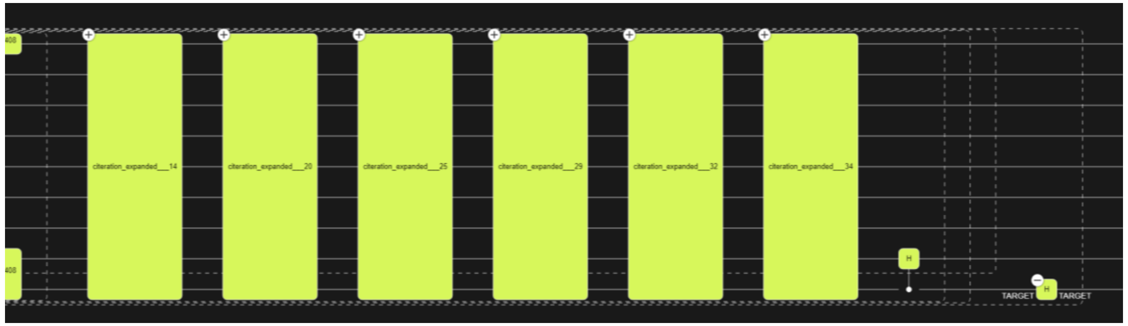
We also implemented the Suzuki-Trotter implementation for 8 qubits,
and considered the QFT (Quantum Fourier Transform) in our Qmod Code.
Since the QFT is applied conditionally,
the outcome might reflect specific properties of the evolved state.

```
        within{
          H(expectation_value);
        }
        apply{
         control(expectation_value){
              qft(qba);
         }
        }
      }
```







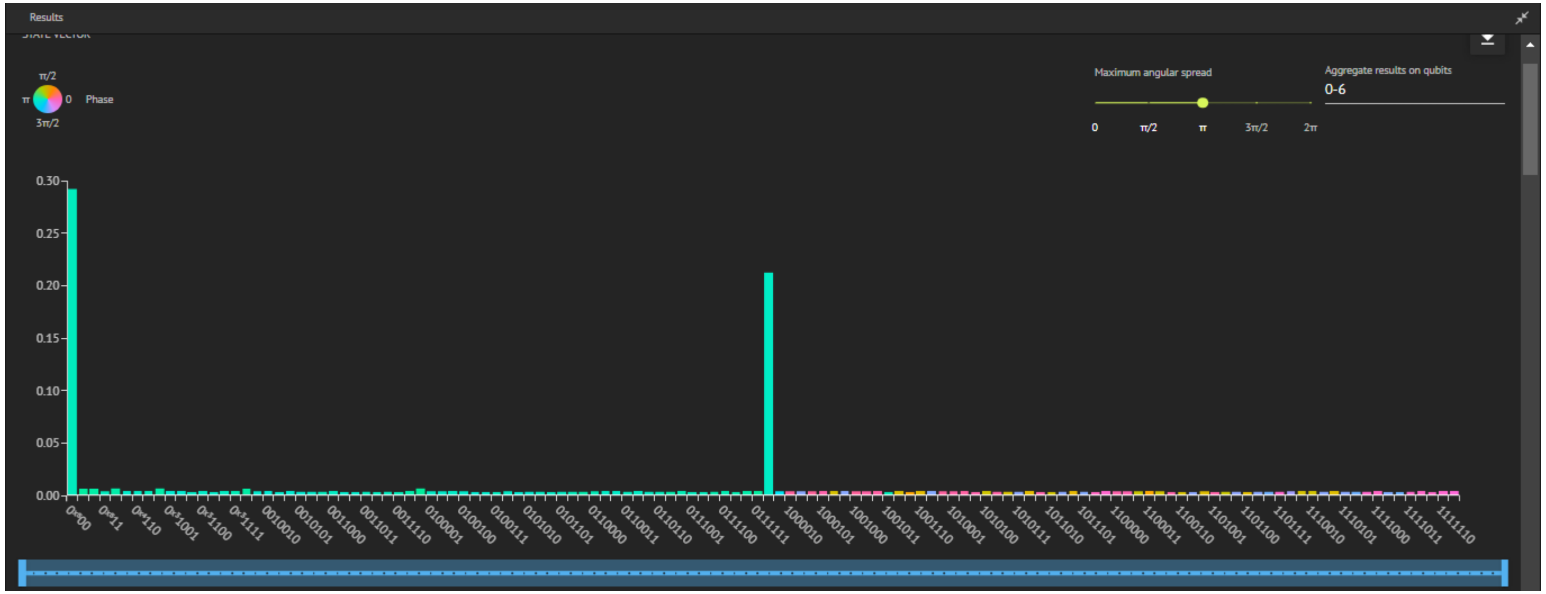


Synthesize Configuration

Program Info

Toy Problem Overview

The result:



Results

Rows per page: 100 1–100 of 512

Result	State Vector	Magnitude ↑	Phase
110000000	(-0.022 - 0.01j)	0.024	π/8
101000000	(-0.022 - 0.012j)	0.025	π/6
110110111	(-0.026 + 0.001j)	0.026	2π
010110111	(0.026 - 0.001j)	0.026	π
111011011	(-0.026 + 0.001j)	0.026	2π
011011011	(0.026 - 0.001j)	0.026	π
110101011	(-0.026 - 0.003j)	0.026	π/24
010101011	(0.026 + 0.003j)	0.026	25π/24
111010111	(-0.026 + 0.001j)	0.026	2π
011010111	(0.026 - 0.001j)	0.026	π
110111011	(-0.026 + 0.001j)	0.026	2π
010111011	(0.026 - 0.001j)	0.026	π
100100000	(-0.023 - 0.013j)	0.026	π/6
101101011	(-0.026 - 0.002j)	0.026	0
001101011	(0.026 + 0.002j)	0.026	π
101011011	(-0.026 - 0.001j)	0.026	0

Toy Problem Overview

- **Analysis the results:**

  The results show two major peaks in the probability distribution:

  one at "00000000" and another at "01111111," indicating these states are most likely after the quantum circuit execution.

  The QFT likely transformed the initial superposition into a pattern with constructive interference at these states.

  Minor peaks suggest partial interference effects from other states. The results are consistent with a Hamiltonian governed by significant ZZ interactions, decomposed through Suzuki-Trotter expansion, and further analyzed using QFT.

  These patterns suggest coherence and phase information influence the observed distribution.

Circuit Enlargement

Scalability of the Implementation:

The implementation is scalable.

The toy problem demonstrates a 8-qubit system, but the structure allows for extension to more qubits by:

- Adding more PauliTerm objects to the suzuki_trotter function.
- Increasing the size of the qubit array "qba" in the allocate function.

To extend to a more complicated scenario, we can adjust the number of qubits and add more interaction terms for larger systems.

As the implementation is scalable, we implemented it for a 16-qubit system, too.

Circuit Enlargement

Implementation of a 16-qubit system in susuki-trotter algorithm, using Classiq IDE:

```
qfunc main(output qba: qbit[]) {
  allocate(16, qba);


  suzuki_trotter([
    // ZZ interactions for 16 qubits
    PauliTerm {
      pauli=[
        Pauli::Z,
        Pauli::Z,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I
      ],
      coefficient=1
    },
```

...

```
    PauliTerm {
      //16
      pauli=[
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::I,
        Pauli::X
      ],
      coefficient=0.5
    }
  ], 0.1, 4, 1, qba);
}
```

Transpiled Info | Program Info | Data

Backend name: Default
Depth: 993
Width: 16

Gate count
U : 1169
CX : 2216

Hardware details
Basis Gates: ry, sxdg, u, sx, p, u1, rx, z, x, rz, id, h, r, u2, y, cz, sdg, cx, s, tdg, t, cy
Connectivity Map:
null
Symmetric Connectivity: True

Transpiled Info | Program Info | Data

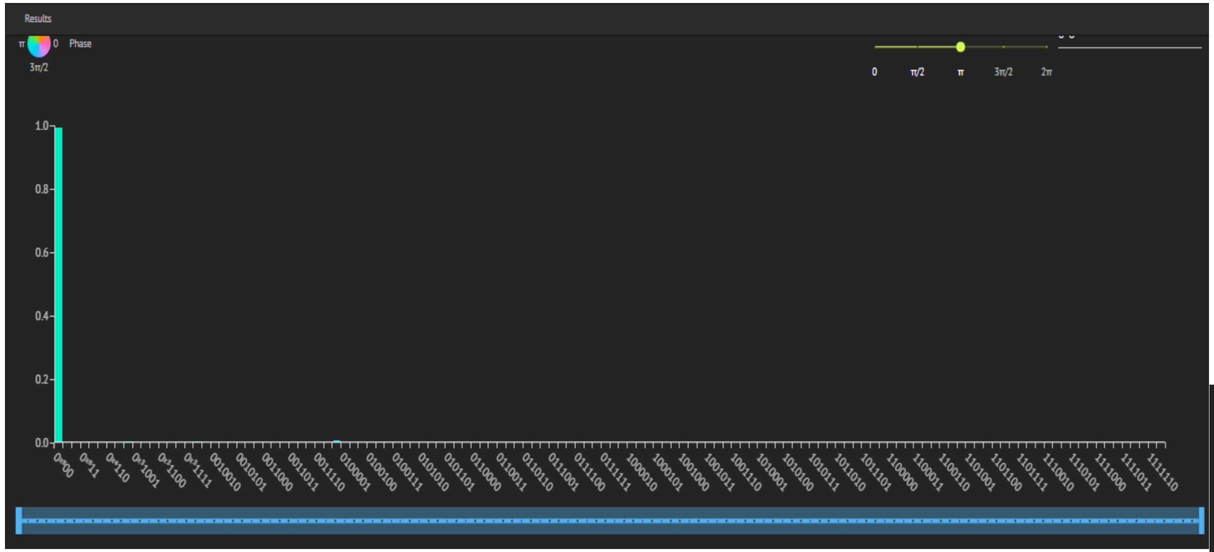
Depth: 1300
Width: 16

Gate count
CX : 2260
RZ : 1700
H : 320

Meta data
Circuit Display Name: program
ID: 75665e00-c0df-4980-ae68-6bd75ac53515

Circuit Enlargement

Results of a 16-qubit system in susuki-trotter algorithm, using Classiq IDE (without QFT)

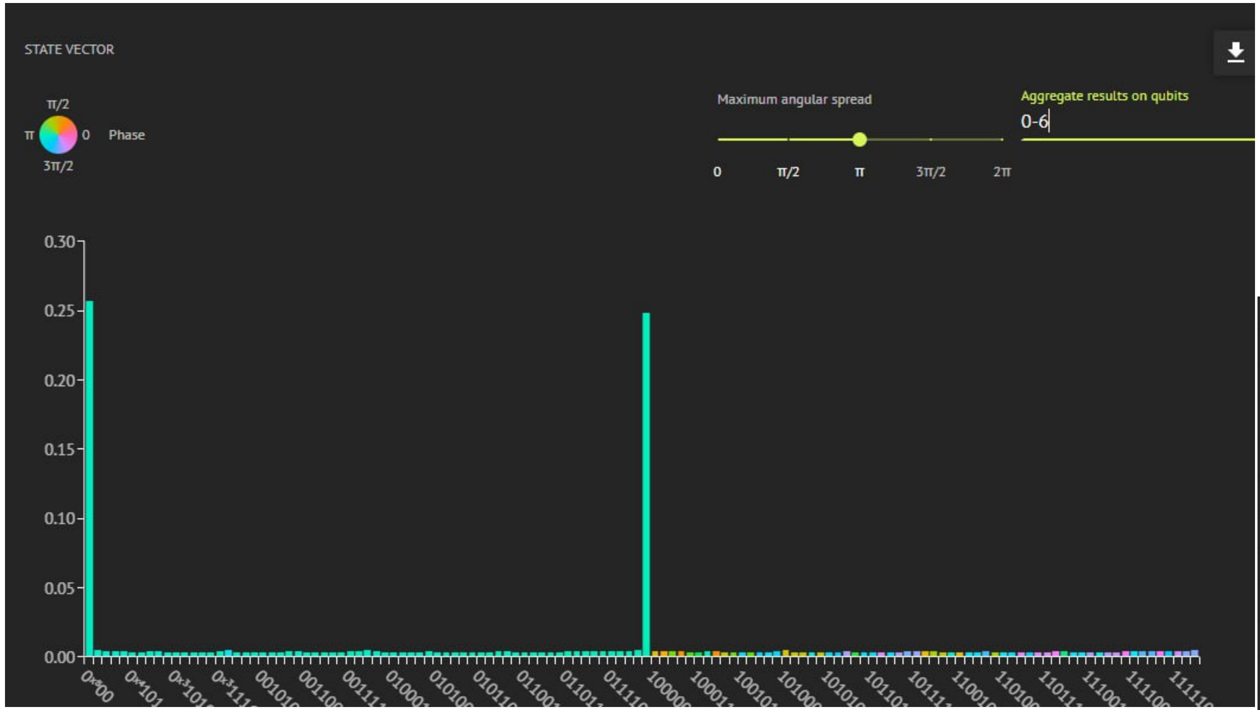


Rows per page: 100 1–100 of 65536

Result	State Vector	Magnitude ↑	Phase
1111111111111111	(-2.384e-22 - 4.149e-22j)	4.785e-22	π/3
1111111110111111	(1.008e-20 - 1.809e-22j)	1.008e-20	π
0111111111111111	(1.014e-20 - 3.059e-22j)	1.014e-20	π
1111101111111111	(1.016e-20 + 1.862e-22j)	1.016e-20	π
1111011111111111	(1.054e-20 - 2e-22j)	1.054e-20	π
1111110111111111	(1.055e-20 + 7.257e-22j)	1.057e-20	25π/24
1111111111110111	(1.062e-20 + 3.244e-22j)	1.062e-20	π
1110111111111111	(1.066e-20 - 2.535e-22j)	1.066e-20	π
1111111111101111	(1.067e-20 + 3.014e-22j)	1.068e-20	π
1101111111111111	(1.07e-20 - 4.038e-22j)	1.07e-20	π
1111111101111111	(1.071e-20 + 3.557e-22j)	1.071e-20	π
1111111111111011	(1.077e-20 + 3.044e-22j)	1.077e-20	π
1111111111011111	(1.082e-20 + 3.939e-22j)	1.083e-20	π
1111111111111110	(1.105e-20 + 2.834e-22j)	1.106e-20	π
1111111111111101	(1.12e-20 + 1.026e-21j)	1.125e-20	25π/24
1111111011111111	(1.123e-20 + 8.195e-22j)	1.126e-20	25π/24
1011111111111111	(3.649e-21 + 1.164e-20j)	1.22e-20	17π/12
1111101110111111	(-8.083e-20 + 1.756e-19j)	1.933e-19	13π/8
1111111110110111	(-8.792e-20 + 1.73e-19j)	1.941e-19	5π/3

Circuit Enlargement

Results of a 16-qubit system in susuki-trotter algorithm, using Classiq IDE (with QFT)



000000001000000000	(0.016 + 0.003j)	0.016	13π/12
000000010000000000	(0.017 + 0.002j)	0.017	25π/24
000000000000001000	(0.017 + 0.002j)	0.017	25π/24
000000000001000000	(0.017 + 0.002j)	0.017	25π/24
000000000000100000	(0.017 + 0.002j)	0.017	25π/24
000000000100000000	(0.017 + 0.004j)	0.017	13π/12
000000000000010000	(0.017 + 0.002j)	0.018	25π/24
000000000000000010	(0.017 + 0.003j)	0.018	25π/24
000000000000000100	(0.018 + 0.002j)	0.018	25π/24
000000000010000000	(0.018 + 0.002j)	0.018	25π/24
000000000000000001	(0.018 + 0.002j)	0.018	25π/24
000001000000000000	(0.019 + 0.001j)	0.019	π
010000000000000000	(0.019 - 0.001j)	0.019	π
000100000000000000	(0.019 - 0.001j)	0.019	π
000010000000000000	(0.02 - 0.001j)	0.02	π
101000000000000000	(0.015 + 0.027j)	0.031	4π/3
001000000000000000	(0.021 + 0.027j)	0.035	31π/24
100000000000000000	(0.492 - 0.007j)	0.492	π
000000000000000000	(0.499 - 0.007j)	0.499	π

Circuit Enlargement

Implementing for an Actual Problem

To implement this for an actual problem:

Model the Problem: Identify the specific Ising model parameters (e.g., values for J and h) and the system size (number of qubits).

Set Parameters: Assign appropriate values to J and h based on the actual problem.

Expand Hamiltonian Terms: Add more PauliTerm objects if the problem has more interactions.

Increase Qubit Array Size: Adjust the size of qba in allocate to match the number of qubits in the problem.

Optimization Techniques

Optimizing for Hardware

Optimizing the solution for specific hardware involves several considerations:

Gate Fidelity: We should use hardware with high gate fidelity for accurate results.

Qubit Connectivity: We should ensure that the hardware supports the required qubit interactions (e.g., nearest-neighbor interactions).

Error Mitigation: We can implement error correction techniques if supported by the hardware.

Resource Management: We can optimize the number of qubits and gates to fit within the hardware's capabilities.

Optimization Techniques

Optimizing for Hardware

Here, we suggest some tips that can be done for optimizing such system:

- **For the toy problem:**

  - We can simulate the circuit on a quantum simulator to verify correctness.

  - We can use a quantum processor with at least 4 qubits and support for the required gate operations.

- **For the actual problem:**

  - One can choose hardware with sufficient qubits to accommodate the larger system.

  - We can ensure the hardware has the necessary connectivity and gate operations for the Ising model interactions.

  - We can consider running multiple experiments and using statistical analysis to mitigate errors.

Acknowledgments

We would like to express our gratitude to **the Womanium Quantum+AI**, not only for spreading Quantum Science and Technology but also for fostering collaboration in scientific projects.

Our thanks also go to the **Classiq** support team for providing the platform and support that made this project possible.

We are especially grateful to the **Womanium** team, including **Dr. Marlou Slot**, **Mr. Vardaan Sahgal**, and **Jasmine**, as well as the **Classiq** support team, particularly **Dr. Eden Schirman** and **Mr. Bakhao Dioum**.

Katayoun, Mahla