KTH VETENSKAP OCH KONST

Design and Implementation of a Quantum Computing Simulator in Python

Duarte Rebelo de São José, Jens Verherstraeten

Problem & Goal

- **Problem:** A Quantum Computer Simulator
- **Goal:** Implementing a correctly working quantum simulator inside the programming language Python. We want to test the performance for varying key design choices and compare them to an existing quantum simulating framework (Qiskit). We want to observe their influence on custom circuits, such as the Quantum Fourier Transform.

Background

- Our implementation:
  - Model quantum gates via Unitary Operators
  - But utilize different state representations

- **Problems:**
  - The memory requirement scales as $O(n^2)$
    - 30 Qubits $\approx$ 16GB of memory
  - Modelling gates as matrices still allocates a lot of memory
    - Matrix dimensions are $n^2 \times n^2 \rightarrow O(2^{2n})$ FLOP for each multiplication

- How do we prevent/minimize these problems?

Methodology

- We use multiple techniques for storing state representations
  - e.g. for $|0\rangle$:

1. Dense Cartesian → $[1 + 0j, 0 + 0j, \dots] \in \mathbb{C}^{2^n}$
2. Dense Polar → $mag = [1,0, \dots] \in \mathbb{R}^{2^n}, phase = [0,0, \dots] \in \mathbb{R}^{2^n}$
3. Sparse Cartesian → $\{0\text{: } 1.0 + 0\text{j}\}$
4. Vectorized → $[1 + 0\text{j}, 0 + 0\text{j}, \dots] \in \mathbb{C}^{2^n}$(Uses NumPy's vectorized operations)

- There are implementations for the following gates:
  - X, Y, Z, H, S, T, RX/RY/RZ, CX/CZ/CP, SWAP, measure

Implementation

- **Dense Cartesian Simulator**
  - Stores full complex vector state of size 2^n
  - *apply_single_qubit():*
    - Loops over every amplitude pair differing by one bit; multiplies by the 2×2 gate
  - *apply_two_qubit():*
    - *L*oops over quadruples of amplitudes; applies 4×4 gate

  - Simple, direct, but slow for many qubits (explicit Python loops)

  - Takeaway: clear reference model for correctness

Implementation

- **Dense Polar Simulator**
  - Stores magnitude + phase arrays separately (r, θ)
  - Converts to complex → applies gate (same math) → converts back

  - Functionally identical to Cartesian version; slower due to conversions
  - Useful conceptually to separate amplitude and phase, but not efficient here
    - Could be used to make Phase modifying gates more efficient

  - Takeaway: demonstrates alternative representations of the same quantum state

Implementation

- **Sparse Dictionary Simulator**
  - Stores only non-zero amplitudes in a Python dictionary
    - *E.g.   {basis_index: amp}*

  - Great when state is sparse (few non-zeros)
  - Each gate only touches indices that could be affected

  - After updates:
    - _prune() removes tiny amplitudes

  - Scales linearly with number of active basis states, not 2^n
  - Takeaway: trades memory for flexibility; efficient for sparse superpositions

Implementation

- **Vectorized Simulator**
  - Same dense complex vector as Cartesian, but fully NumPy-vectorized
  - Reshape state → treat as tensor of shape [2]*n
  - Move target qubit axes to front, apply gate via matrix multiply, reshape back
  - No Python loops; all done in C-level NumPy code
  - Fastest dense simulator here
  - Takeaway: same math, different implementation for performance

Experiments & Results

- Verifying correctness:
  - Test suite using *pytest*

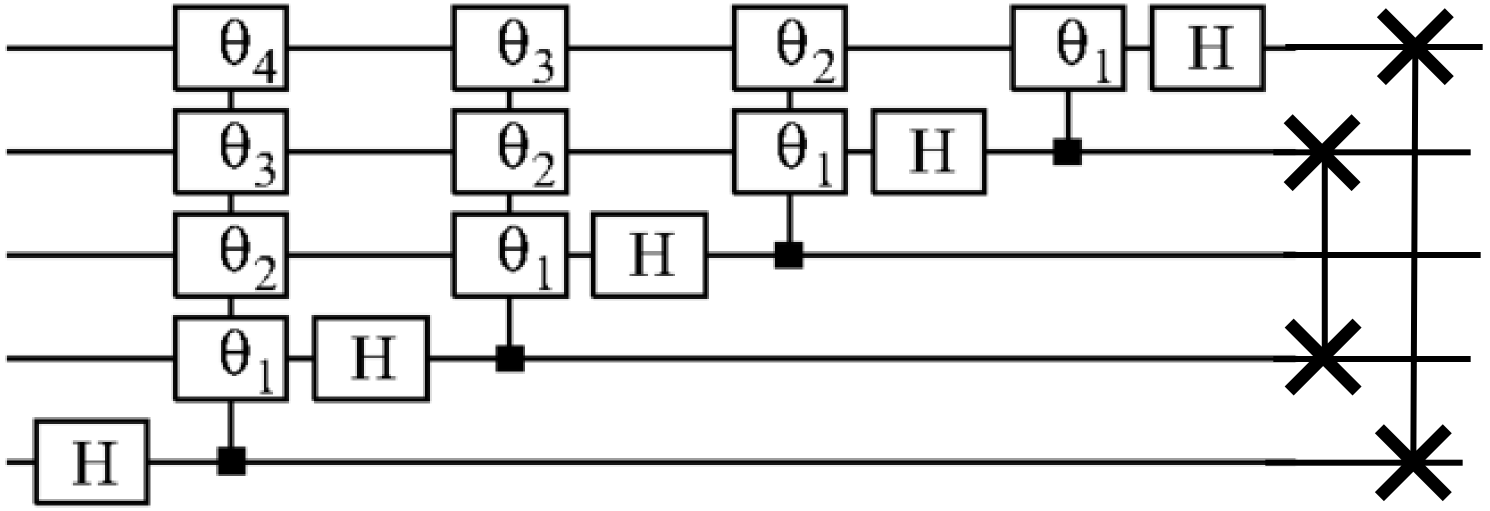
- Validate quantum gates on Single- and Multi-qubit systems
  - For different arbitrary situations

- Compare with Qiskit's implementation (assumed to be correct)

Case Study: The QFT circuit

From Qiskit's Circuit Library:

QFTGate().inverse() =



Source: "Analytical formulas for the performance scaling of quantum processors with a large number of defective gates", Yunseong N. (2015)

Theoretical Memory Model

- We are ignoring the overhead of the circuit building and backend libraries
- Theoretical bound (Dense Cartesian):
  - State representation *(np.complex):* $16 \cdot 2^{n}$ Bytes
  - QFT:
    - n Hadamard gates
    - $n(n-1)/2$ CPHASE gates
    - $\lfloor n/2 \rfloor$ SWAP gates
    - $\approx O(n^{2})$ gates (stored in compact matrices)

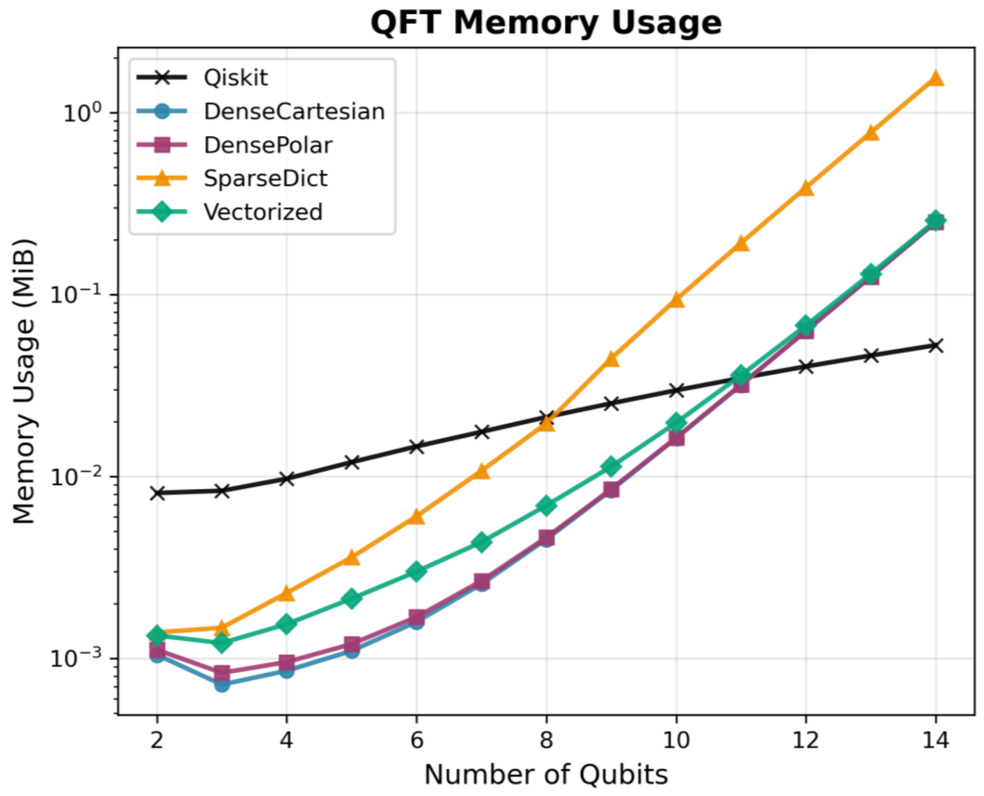
- Total Amount of allocated Bytes: $\approx 16 \cdot 2^{n} + O(n^{2}) = O(2^{n})$

Practical Memory Allocations

- Qiskit performs worse then all implementations for $n < 8$

- When the system grows, Qiskit scales better then our implementation

- Clear difference between the Sparse representation and the others



Theoretical Execution Model

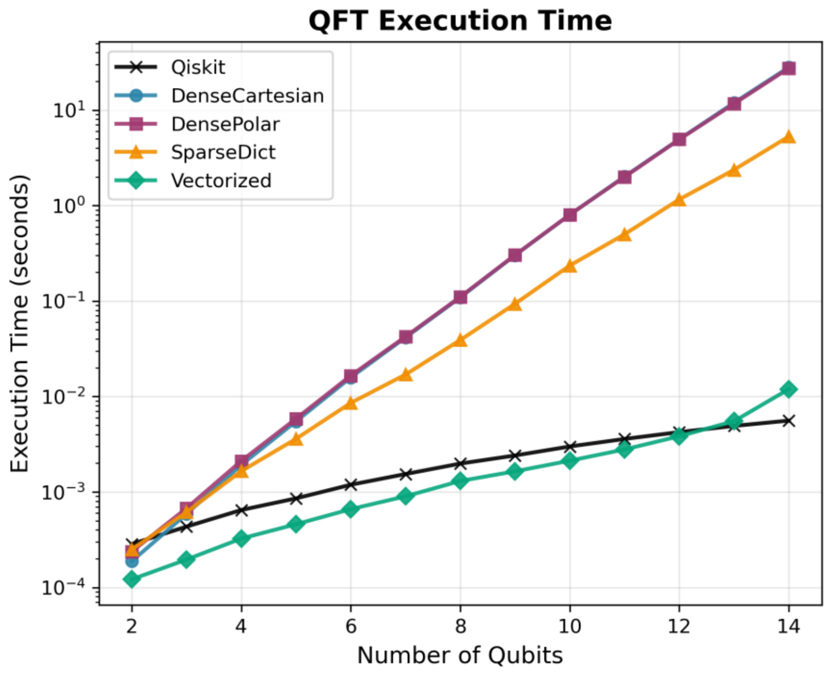
- Number of FLOP: $F(n) = (\mathrm{TotalGates}) \times (\mathrm{FLOP\ per\ Gate})$
- For the Dense Cartesian circuit:
  - QFT has $\approx \frac{1}{2}\mathrm{n}^2$ gates
  - 1-Qubit Gate requires $\mathrm{O}(2^{\mathrm{n}})$ FLOP
  - 2-Qubit Gate also requires $\mathrm{O}(2^{\mathrm{n}})$ FLOP

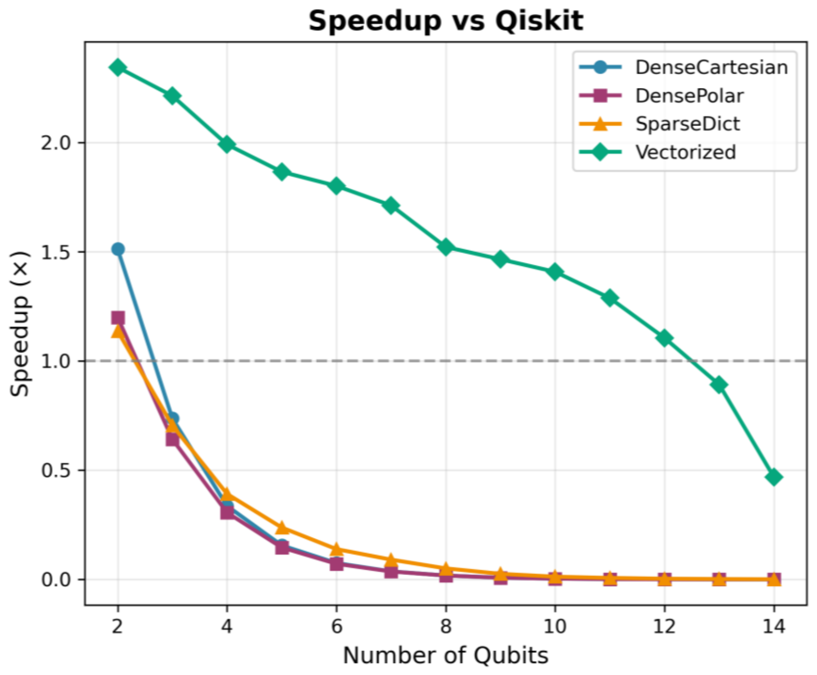
- Total amount of FLOP: $\mathrm{F(n)} \approx (\frac{1}{2}\mathrm{n}^2) \times (\mathrm{C} \cdot 2^{\mathrm{n}}) = \mathrm{O}(\mathrm{n}^2 2^{\mathrm{n}})$
- This also demonstrates the lower performance on classical hardware:
  - On QPU: QFT = $\mathrm{O}(\mathrm{n}^2)$

Execution Time

- We can only outperform Qiskit for small systems

- Clear difference in performance between Vectorized and classic Dense/Sparse implementations





Discussion

- We constructed a framework that implements a basic quantum computing simulator which can measure and compare different state representation methods for custom circuits

- Still room for improvement for gate specific optimizations for certain representation methods
  - E.g.: Different PHASE gate for polar representation

- Theoretical convergence for performance and memory allocation is in line with the observations

- We achieved a fairly great performance for simulating circuits using the vectorized approach
  - Scales much better compared to the other implementations for the QFT circuit

Contributions & Expected Grade

- Framework and Implementation: Duarte Rebelo de São José, Jens Verherstraeten
- Testing and Experimenting: Jens Verherstraeten, Duarte Rebelo de São José

- We expect to receive a grade of **“A”** as we developed a quantum computing simulator featuring multiple state representations and optimizations, and conducted extensive experiments to evaluate design choices

References

IBM, “Qiskit Documentation,” *IBM Quantum*, Accessed: Oct. 19, 2025. [Online]. Available: https://docs.quantum.ibm.com/

J. M. Romero, E. Montoya-González, G. Cruz, and R. C. Romero, “A novel quantum circuit for the quantum Fourier transform,” *arXiv preprint arXiv:2507.08699*, 2025. [Online]. Available: https://arxiv.org/abs/2507.08699

Github repo:

https://github.com/DuarteSJ/Quantum_Computing_KTH/tree/main/quantum-simulator