جامعة الملك فهد للبترول والمعادن
King Fahd University of Petroleum & Minerals

Comparison of Quantum Circuit Simulation Methods

ICS Project

Ibraheem AlYousef
Supervisor: Dr. Alawi AlSaqqaf
King Fahd University of Petroleum and Minerals

Outline

Grover's Search Algorithm

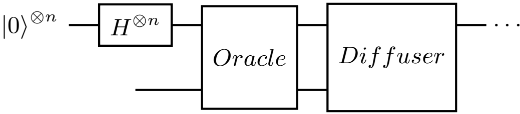
Goal: Find marked item in unsorted database of $N = 2^n$ items

- Classical: $O(N)$ queries
- Quantum: $O(\sqrt{N})$ queries — **quadratic speedup**

Success probability after k iterations:

$$P_{\text{target}} = \sin^2((2k+1)\theta), \quad \theta = \arcsin(1/\sqrt{N})$$

Optimal: $k = \lfloor \pi\sqrt{N}/4 \rfloor$



Repeat Oracle+Diffuser $\lfloor \pi\sqrt{N}/4 \rfloor$ times

Overview of Simulation Methods

State-Vector	Density Matrix	MPS	Unitary	Pulse-Level
$O(2^n)$	$O(4^n)$	$O(n\chi^2)$	$O(4^n)$	$O(4^n)$

Method	Representation	Best For
State-Vector	Complex amplitude vector	Pure states, $n < 30$
Density Matrix	$2^n \times 2^n$ matrix	Noise, decoherence
MPS (Tensor)	Chain of tensors	Low entanglement
Unitary	Full transformation U	Gate verification
Pulse-Level	Hamiltonian $H(t)$	Hardware dynamics

State-Vector Simulation

Type: **Exact**

Representation:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \quad \sum_i |\alpha_i|^2 = 1$$

Properties:

- Memory: $O(2^n)$ — most efficient
- Practical up to $\sim$30 qubits

Choose when: Developing/debugging algorithms on pure states without noise

```
def run_statevector(circuit):
    from qiskit_aer import AerSimulator

    sim = AerSimulator(method='statevector')
    circuit_copy = circuit.copy()
    circuit_copy.save_statevector()

    result = sim.run(circuit_copy).result()
    sv = result.get_statevector()
    return np.abs(sv.data)**2
```

Density Matrix Simulation

Type: Exact

Representation:

$$\rho = \sum_k p_k |\psi_k\rangle \langle\psi_k|$$

Properties:

- Memory: $O(4^n)$ — $2^n \times 2^n$ matrix
- Practical limit: $\sim$14 qubits

Choose when: Simulating noise, decoherence, or mixed quantum states (open systems)

```python
def run_density_matrix(circuit, n_qubits):
    if n_qubits > 14:  # Memory guard
        return None

    from qiskit_aer import AerSimulator

    sim = AerSimulator(method='density_matrix')
    circuit_copy = circuit.copy()
    circuit_copy.save_density_matrix()

    result = sim.run(circuit_copy).result()
    dm = result.data()['density_matrix']
    return np.diag(dm.data).real
```

Matrix Product State (MPS) Simulation

Type: **Approximate** (tunable via χ)

Tensor Network Representation:

$$|\psi\rangle = \sum_{i_1,\ldots,i_n} A^{[1]}_{i_1} A^{[2]}_{i_2} \cdots A^{[n]}_{i_n} |i_1 \cdots i_n\rangle$$

Properties:

- Memory: $O(n \cdot \chi^2)$ — linear in qubits!
- Bond dimension χ controls accuracy
- Truncates high-entanglement states

Choose when: Simulating many qubits (> 30) with limited entanglement (e.g., 1D systems, shallow circuits)

```
def run_mps(circuit):
    from qiskit_aer import AerSimulator

    sim = AerSimulator(
        method='matrix_product_state'
    )
    circuit_copy = circuit.copy()
    circuit_copy.save_statevector()

    result = sim.run(circuit_copy).result()
    sv = result.get_statevector()
    return np.abs(sv.data)**2
```

Unitary & Pulse-Level Simulation

Unitary Simulation: Exact

- Computes full U: $|\psi'\rangle = U|\psi\rangle$
- Memory: $O(4^n)$, limit ∼12 qubits

Choose when: Verifying gate implementations, debugging circuits, reusing U for multiple inputs

Pulse-Level: Exact (numerical)

$$i\hbar \frac{d|\psi\rangle}{dt} = H(t)|\psi\rangle$$

- Solves Schrödinger equation directly
- Models real hardware dynamics

Choose when: Designing control pulses, analyzing hardware timing, studying decoherence effects

```python
def run_qutip_pulse(n_qubits, target, iters):
    from qutip_qip.circuit import QubitCircuit
    from qutip_qip.device import LinearSpinChain
    from qutip import basis, tensor

    qc = QubitCircuit(n_qubits)
    # Build Grover circuit with native gates
    for i in range(n_qubits):
        qc.add_gate("SNOT", targets=i)
    # ... oracle and diffuser

    processor = LinearSpinChain(n_qubits)
    processor.load_circuit(qc)
    init = tensor([basis(2,0) for _ in range(n)])
    result = processor.run_state(init_state=init)
    return result.states[-1].full().flatten()
```

Oracle Construction

Goal: Flip phase of target state: $|t\rangle \to -|t\rangle$

Key Idea: Multi-controlled Z (MCZ) flips phase only when *all* qubits are $|1\rangle$. To mark arbitrary target $|t\rangle$:

1. Apply X gates to qubits where target bit is 0
   (transforms $|t\rangle \to |11...1\rangle$)
2. Apply MCZ gate (flips phase of $|11...1\rangle$)
3. Undo X gates (restores basis states)

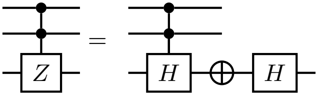
Example: Target $|01\rangle$ (decimal 1)

- Apply X to qubit 1: $|01\rangle \to |11\rangle$
- MCZ: $|11\rangle \to -|11\rangle$
- Undo X: $-|11\rangle \to -|01\rangle$

MCZ via H-MCX-H:

MCZ not native $\to$ decompose using identity:

$$\text{CZ} = (I \otimes H) \cdot \text{CNOT} \cdot (I \otimes H)$$



H converts Z basis to X basis,
MCX (Toffoli) flips target,

H converts back

Verification Results

Test: 8-qubit Grover's search, target = $|00101010\rangle$ (42), 13 iterations

All Methods Achieve Theory

Method	Target Prob	Status
State-vector	0.9862	PASS
Density Matrix	0.9862	PASS
MPS	0.9862	PASS
Unitary	0.9862	PASS

Theoretical: $\sin^2(27\theta) \approx 0.9862$

Performance Comparison

Method	Time	Memory	Scaling
State-vec	538ms	4KB	$O(2^n)$
Density	2570ms	1MB	$O(4^n)$
MPS	675ms	4KB	$O(n\chi^2)$
Unitary	66ms	1MB	$O(4^n)$

Pulse-level (2 qubits, target $|11\rangle$): Achieves 100% success, matching state-vector.

Why is Unitary Faster than Density Matrix?

Both have $O(4^n)$ memory, but unitary is ∼40$\times$ faster. Why?

Unitary Simulation:

- Gate application: $U' = G \cdot U$
- One matrix multiplication per gate
- Final step: $|\psi'\rangle = U|0\rangle$ (matrix-vector)
- Optimized for transformation computation

Density Matrix Simulation:

- Gate application: $\rho' = G\rho G^\dagger$
- **Two** matrix multiplications per gate
- Must track full $2^n \times 2^n$ operator
- Designed for noise/decoherence modeling

$\Rightarrow$ Density matrix does $2\times$ operations per gate + overhead for mixed state tracking

Use Cases Summary

Method	Best Use Cases	Limitations
State-vector	Algorithm development, pure state simulation	No noise modeling
Density Matrix	Noise characterization, decoherence, open systems	$n < 15$ due to memory
MPS	Large qubit counts, low-entanglement circuits	Accuracy depends on χ
Unitary	Gate verification, circuit debugging	Same limits as density matrix
Pulse-Level	Hardware timing analysis, control pulse design	Complex gate decomposition

Key Takeaways

1. **State-vector** is optimal for pure-state algorithm development with $O(2^n)$ memory scaling
2. **Density matrix** enables noise modeling but scales as $O(4^n)$, limiting to $\sim$14 qubits
3. **MPS** offers memory efficiency with tensor decomposition—best for low-entanglement
4. **Unitary** simulation is specialized for gate verification with fastest execution
5. **Pulse-level** via QuTiP provides hardware-level insight into Hamiltonian dynamics
6. All methods achieved the expected 98.6% success probability, validating Grover's algorithm

Questions?