README.txt

Sparse Quantum States are given to us as a dictionary, of the form:

{'000': 0.25, '001': 0.5, '111': 0.25}

which translates to a three-qubit system in which only states 0, 1, and 7 have nonzero probabilities 0.25, 0.5, and 0.25 respectively. Sparsity, in this context, refers to the fact that a large number of basis states have zero probability. As state preparation is a common routine in quantum computing, it is crucial to come up with an efficient algorithm. Standard, generic algorithms for state preparation have been shown to be inefficient for the preparation of sparse states and an efficient algorithm that is specifically dedicated to the preparation of sparse states is needed. Such an algorithm is described in “An Efficient Algorithm for Sparse Quantum State Preparation” by Gleinig and Hoefler (2021). The purpose of this challenge was to implement the described algorithm and benchmark its performance.

The raw data translates to a more readable format as a 3-dimensional array:

Dictionary: {'000': 0.25, '001': 0.5, '111': 0.25}

3D-Array: [[1, [000, 0.25], [2, [001, 0.5]], [3, [111, 0.25]]

This is done by the function ***dict_to_3d_array(sparse_states)***, in which the parameter sparse_states is the dictionary given. Now formatted as a 3D array, we can calculate the difference between the quantum states digit-by-digit. Notably, the function ***unequal_sets(t, n)*** outputs the “best qubit” that maximizes its difference from the other qubits, and also outputs two nonempty subsets of the original set of qubits. The algorithm determines the set of NOT, CNOT and controlled rotation gates that are needed to transform the given state to the |0> basis state. This is done in a recursive fashion. After each iteration the number of basis states with nonzero probability decreases, until the state eventually collapses. The main part of the challenge was to implement the determined quantum circuit in classiq, but applied in reverse, as the algorithm is supposed to be used for state preparation, starting from |0>.

Once all necessary parameters are established, such as the indices where gates are applied, we have created a more efficient circuit. As we used different test cases, new blindspots were exposed such as the method of splitting registers and binding them together in the appropriate order. The most important factor is the position of the control qubit, in which we use different methods for each possible position within of the target qubit relative to the control qubit: leftmost, rightmost, left of the control qubit, and right of the control qubit. In this way we could avoid introducing any ancilla qubits that would increase the gate depth.

unitary_control(qubit: QArray[QBit], contrl: QArray[QBit], target: QParam[int])

- Parameters
    - qubit: A QArray of QBits, which is formatted to be read by the control(...) function imported from Classiq.
    - contrl: Holds that value of the qubit that has the target.
    - target: The index of the target qubit.
- Output
    - control(lambda: X(qubit[target]), contrl) takes in each parameter of unitary_control(...) as its own parameters.

y_rotation(theta: QParam[float], reg: QArray[QBit], target: QParam[int])

- Parameters
    - theta: The angle at which we are applying a rotation in the y-direction.
    - reg: Shorthand for register, holds the value of the qubit that has the target.
    - target: The index of the target qubit.
- Output
    - RY(theta, reg[target]) takes in each parameter of y_rotation(...) as its own parameters.

LeftControl(psi: QArray[QBit], theta: QParam[float], target: QParam[int], NUM_QUBITS: QParam[int])
RightControl(psi: QArray[QBit], theta: QParam[float], target: QParam[int], NUM_QUBITS: QParam[int])

LLControl(psi: QArray[QBit], theta: QParam[float], target: QParam[int], NUM_QUBITS: QParam[int], dq: QParam[int])
RRControl(psi: QArray[QBit], theta: QParam[float], target: QParam[int], NUM_QUBITS: QParam[int], dq: QParam[int])

- Parameters
    - psi: the QArray of QBits representing the quantum state.
    - theta: The angle at which we are applying a rotation in the y-direction. This is a parameter of my_controller_unitary(...).
    - target: The index of the target qubit.
    - NUM_QUBITS: The number of qubits.
    - dq: An individual value of dif_qubits.

dict_to_3d_array(sparse_states)

- Parameters
    - sparse_states: Raw data in the form of a dictionary, with a string of binary values as the key and its probability as its value.
- Output
    - A 3-dimensional array with its indices holding values of an enumeration of the states, a key, and its value.

algorithm_1(s,n, ops1, ops2, ops3, ops4, ops5, n9)

- Parameters
    - s: From main(), this is the parameter that holds the value of sparse_states.
    - n: From main(), this is the parameter that holds the value of num_qubits.
    - ops1: Initialized as an empty array and is appended:
        - Indices of qubits that are altered by another operation.
    - ops2: Initialized as an empty array and is appended:
        - Indices of qubits that have a NOT gate applied to them.
    - ops3: Initialized as an empty array and is appended 2-arrays with:
        - Index of the qubit that has the CNOT gate applied.
        - Index of the target qubit of the CNOT gate.

- ops4: Initialized as an empty array and is appended 2-arrays with:
    - A 4-array holding values of:
        - Alpha, the probability coefficient of the 0-state.
        - Beta, the probability coefficient of the 1-state.
        - The index of the qubit that has the G gate applied.
    - An n-array holding values of:
        - A new set of probabilities, which should be less than n.
        - The new number of states, which should be less than n.
- ops5: Initialized as an empty array and is appended the "best qubit"
- n9: The maximum difference between the "best qubit" and other states.

- Output
    - Appends sets of indices to the appropriate array of operations.

process_subsets(t, n, dif_qubits, dif_values)

- Parameters
    - t: the parameter holding the value of sparse_states.
    - n: Holds the value of num_qubits.
    - dif_qubits: stack of bits used to control "merging"
    - dif_values:
- Output
    - dif_qubits: stack of bits used to control "merging"
    - dif_values:
    - t: set T, lesser of T_0 and T_1

toggle_operations(index, n, x_x, ops1, ops2, s)

- Parameters
    - index:
    - n: value of num_qubits
    - x_x: single state element from the decomposition of T into unequal sets
    - ops1: array of operation enum values (1=NOT, 2=CNOT, etc.)
    - ops2: array of indexes for NOT gates

    - s: original state 3d array
- Output
    - Appends/modifies existing ops1 and ops2

conditional_toggle(ops1, ops2, n, dif, b, s)

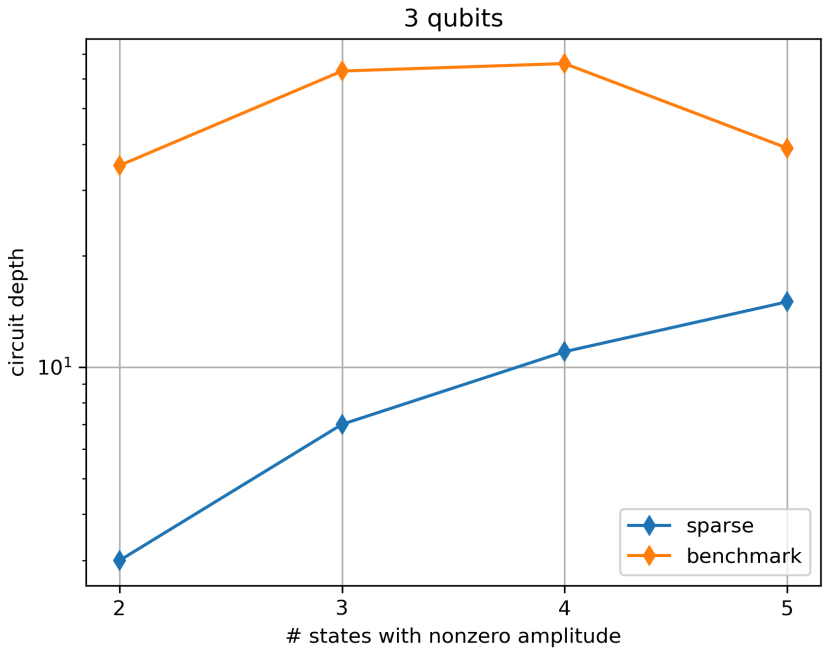
- Parameters
    - ops1: array of operation enum values (1=NOT, 2=CNOT, etc.)
    - ops2: passed ops3, array for CNOT qubit indexes [control, target]
    - n: value of num_qubits
    - dif: index of controlling qubit for “merging”
    - b: current loop index of states
    - s: original state 3d array
- Output
    - Appends/modifies existing arguments

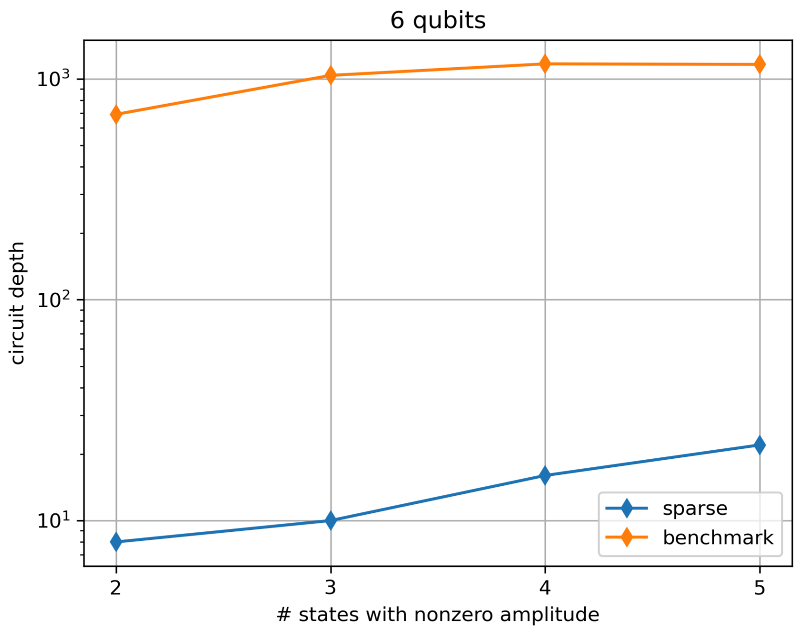
calc_alpha_beta(x_1,x_2)

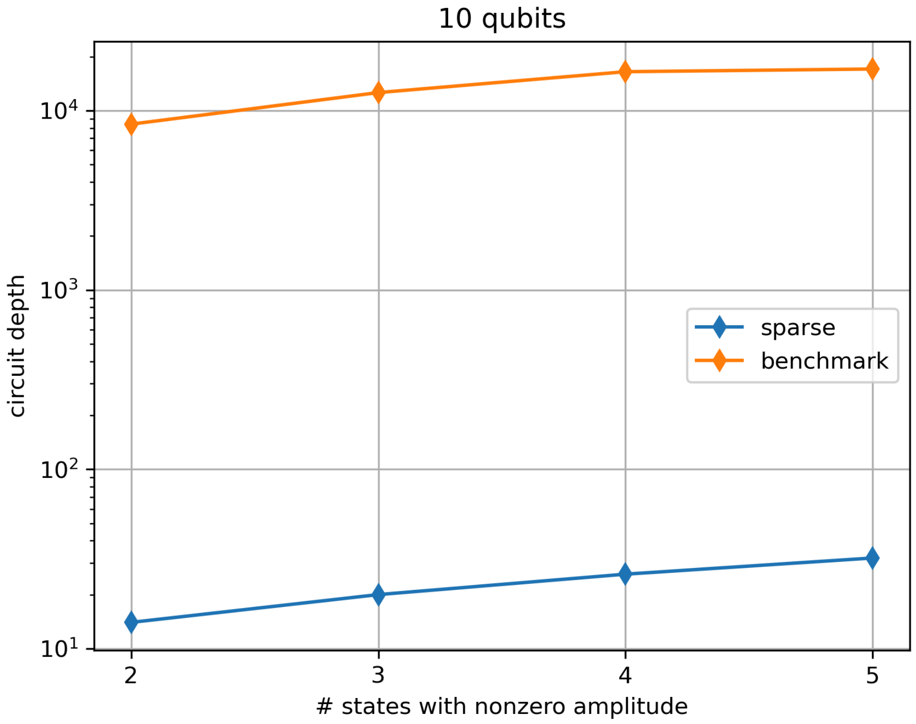
- Parameters
    - x_1: single element in T
    - x_2: single element in T’
- Output
    - alpha: probability amplitude associated with x_1
    - beta: probability amplitude associated with x_2

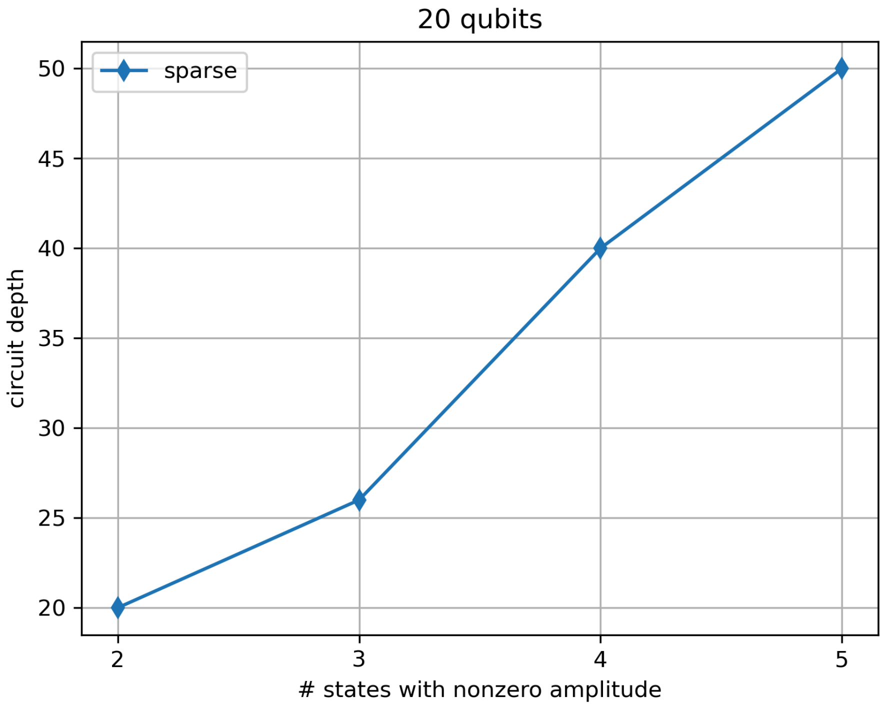
Results:

The designed algorithm successfully prepares the sparse quantum states. In order to benchmark the resulting circuit depth we compared the results to the circuit depth of the built-in state_preparation method in classiq. We have varied both the number of qubits in the register (between 3, 6, 10 and 20) as well as the number of states with nonzero probability (between 2, 3, 4, 5). In the tested scenarios, the sparse state preparation is about 1-3 orders of magnitude more efficient than the benchmark. Especially for systems with a large number of qubits this difference is crucial.


3 qubits
circuit depth
10^1
2
3
4
5
# states with nonzero amplitude
sparse
benchmark


6 qubits
circuit depth
10^3
10^2
10^1
2
3
4
5
# states with nonzero amplitude
sparse
benchmark


10 qubits
circuit depth
# states with nonzero amplitude
sparse
benchmark
10^1
10^2
10^3
10^4
2
3
4
5


20 qubits
circuit depth
# states with nonzero amplitude
sparse
20
25
30
35
40
45
50
2
3
4
5