Simulation of digital quantum circuits

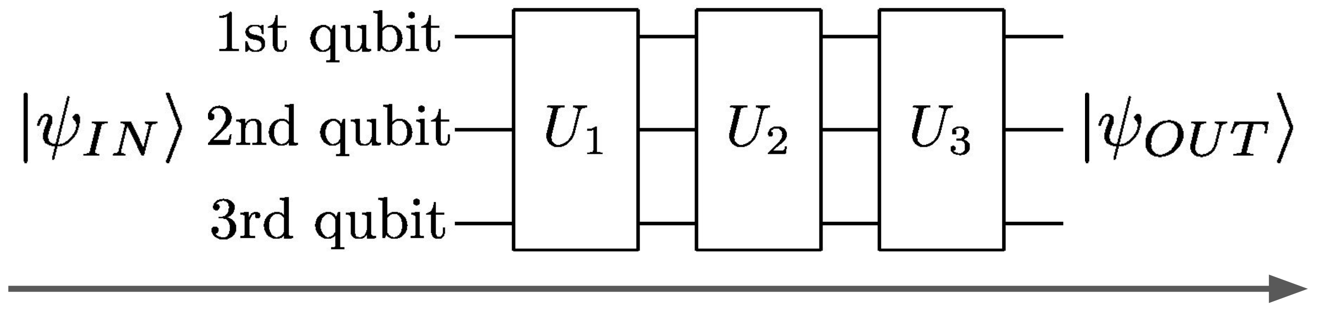
Dhyllan Skiba, Lukas Rosario, Hiroki Kawai

What the simulator can do

- It can simulate the process of the digital quantum computation on a classical computer (i.e. our laptops etc.).
  - One can add any 1-qubit quantum gates and CNOT to the circuit.
  - The simulator computes the unitary representation of the overall circuit, and apply it to an input qubit state by vector-matrix multiplication.
- We take advantage of the sparseness of the matrix representation of the quantum gates to speed up the simulation.
  - We use the library called Eigen (https://eigen.tuxfamily.org/) for the sparse matrices and their multiplication, etc.

Quick Recap of Quantum Circuits

- We usually depict the process of QC as:



$$\wr\wr$$

$$|\psi_{OUT}\rangle = U_3 U_2 U_1 |\psi_{IN}\rangle$$

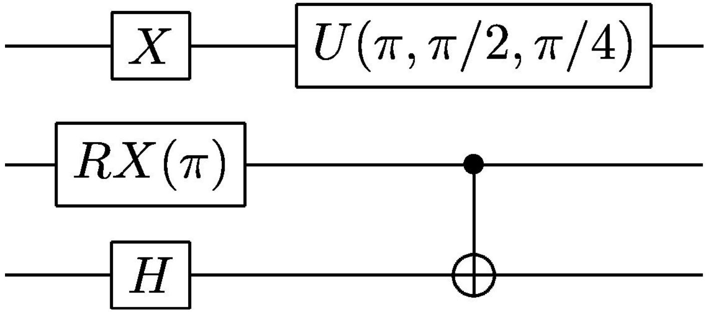
vector matrices vector

Adding gates to the circuit

Code example:

```
QCircuit circ(3); // Initialize a 3-qubit circuit
circ.addX(0); // Add X gate acting on qubit-0
circ.addH(2); // Add H gate acting on qubit-2
circ.addRX(1, PI); // Add rotation X gate on qubit-1 with angle PI
circ.addU1(0, PI, PI/2.0, PI/4.0); // Add general SU(2) rotation
circ.addCNOT(1, 2); // Add CNOT with qubit-1 as control and qubit-2 as target
```

Equivalent to:



Computing matrix representation

- Add(gate name) member functions of the QCircuit class do not compute the matrix representation yet, but the circuit object stores the list of the gates, qubits they are acting on, and the rotation angles if needed.
- To compute the matrix representation, one needs to run QCircuit.compMatrix() once.
- For 1-qubit gates, compMatrix() computes the 1-qubit gate representation (2 x 2-D matrix), and compute the whole 2ˆN x 2ˆN-D representation by taking the tensor product with identity matrices.
  - E.g. for X gate acting on i-th qubit: $X_{whole} = I_{2^i} \otimes X \otimes I_{2^{N-i-1}}$
- The 2ˆN x 2ˆN-D matrix rep of the CNOT gate is directly computed.
- The entire unitary matrix for the circuit is computed by matrix multiplications of those matrices.
- All matrices are stored in the sparse CSR format.
- And applyGates(Complex* state) applies the computed matrix on the input state.

Compressed Sparse Row Matrix Format (CSR)

- Only stores nonzero values in the ‘val’ array
- Stores the index of start of each row in the ‘row’ array
  - Row 1 starts at val[0]
  - Row 2 starts at val[2]
  - Row 3 starts at val[3]
  - Row 4 starts at val[4]
  - Row length = row[i+1] - row[i]
- Stores the column corresponding to each value in the ‘col’ array

$$\begin{bmatrix} 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathsf{val}[\mathsf{numVals}] = \begin{bmatrix} 1 & 2 & 1 & 3 & 2 & 1 \end{bmatrix}$$

$$\mathsf{row}[\mathsf{numRows}+1] = \begin{bmatrix} 0 & 2 & 3 & 5 & 6 \end{bmatrix} \longleftarrow \text{numVals}$$

$$\mathsf{col}[\mathsf{numVals}] = \begin{bmatrix} 1 & 3 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Tensor Products

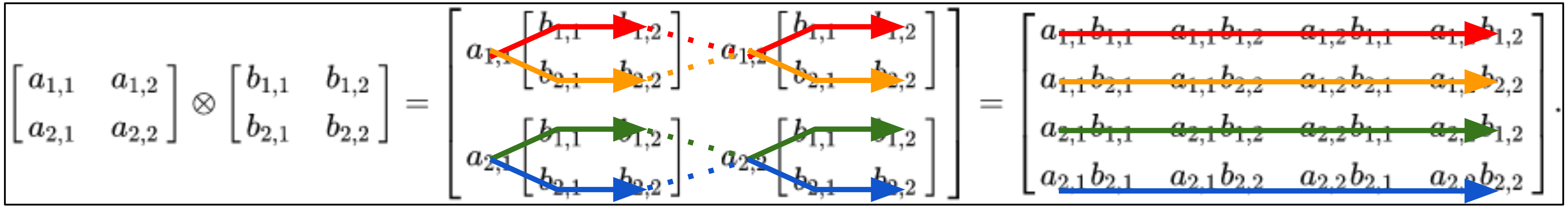
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

Tensor Product - Wikipedia

- “Tensor products are used to describe systems consisting of multiple subsystems.” - Quantiki
- For example, if we have a two qubit system, the whole system can be represented by:

$$|0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle$$

Tensor Products with CSR



$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

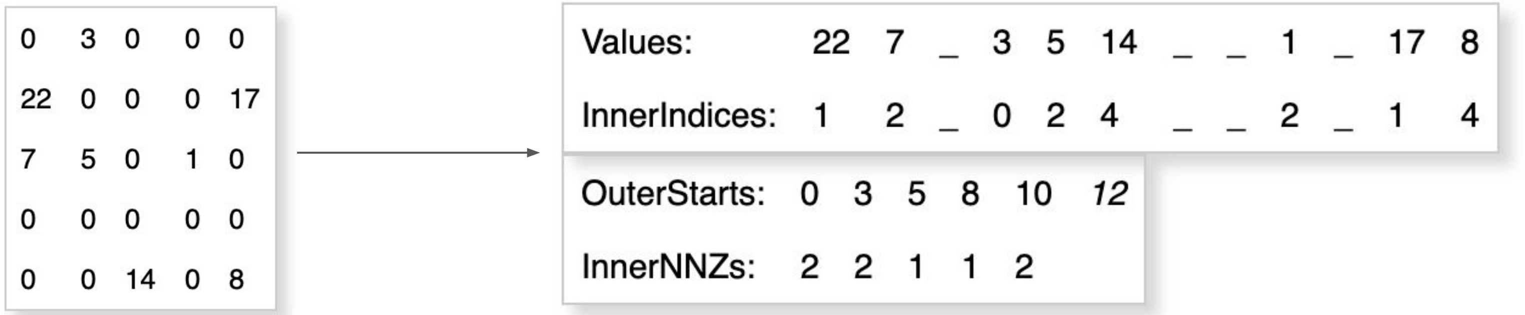
- Due to the CSR format, needed to fill in each row of the tensor product sequentially
- This likely isn't the most efficient algorithm since each value in the second matrix is accessed 4 times.
- With this algorithm, only the scalar multiplications can be executed in parallel
  - The varying row lengths make it difficult to predict loop boundaries, and thus parallelizing is difficult using OpenACC.

Implementation with Eigen

- Linear algebra operations implemented using Eigen
- Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.
- Offers simplicity and flexibility over initial implementations from scratch
- Out of the box support for matrix multiplication and parallelization using OpenMP

Sparse Matrices in Eigen

- Uses a variant of CSR / CSC that stores sparse matrix as 4 arrays
  - Values: stores the coefficient values of the non-zeros.
  - InnerIndices: stores the row (resp. column) indices of the non-zeros.
  - OuterStarts: stores for each column (resp. row) the index of the first non-zero in the previous two arrays.
  - InnerNNZs: stores the number of non-zeros of each column (resp. row). The word inner refers to an inner vector that is a column for a column-major matrix, or a row for a row-major matrix. The word outer refers to the other direction.



0	3	0	0	0
22	0	0	0	17
7	5	0	1	0
0	0	0	0	0
0	0	14	0	8

Values:	22	7	_	3	5	14	_	_	1	_	17	8
InnerIndices:	1	2	_	0	2	4	_	_	2	_	1	4

OuterStarts:	0	3	5	8	10	*12*
InnerNNZs:	2	2	1	1	2	

Summary & Further things to do

- Digital quantum circuit simulator on a classical computer.
- The user can add any 1-qubit gate or CNOT gate to the circuit.
- It computes the matrix representation of the circuit as sparse matrix and can apply it to the input state vector.

- To do:
    - Comparisons with other simulators, e.g. qiskit by IBM.
    - Benchmarking the parallelization.