

Proposals to improve the previous Quantum Simulator

1. Computations on CPU are slow when it comes to significant Linear transformations:
 - a. The earlier version used NumPy for the linear transformations; however, to my knowledge, NumPy is built to run on CPU, and I don't know any NumPy version that can run on GPU. To be more specific, NumPy arrays are not vectorized.
 - b. We can move our code depending on NumPy to torch to leverage the GPU capabilities. The torch.tensors can pretty much do all the operations we did use NumPy, and we can perform all the tasks on GPU as well(if available)
2. Another Optimization we could do is add a transpiler that can take a complex Quantum circuit as input and internally convert it into a simpler circuit that can be computed much faster. Ex: Adding two Hadamard gates one after the other (with no gates in between) is the same as not applying any Hadamard gate at all. Since Hadamard is a Hermitian operator and it is its inverse.
3. *'With Great Power comes Great Responsibility.'* Moving the code to PyTorch isn't enough. With the great GPU computation power available at hand, our responsibility is to make the best out of it and use it as efficiently as possible. Using techniques like **batching** must be implemented to leverage most of the potential that the hardware can offer for faster computations—more on how I used batching to calculate expectations in one go at the end of this document.
4. Measurement: When implementing a simulator, we don't need the measuring done using the number of shots. I think it is fair to directly return probabilities of the states or Expectation values of the qubits in the circuit.

Expectations of the qubits:

I'm only computing Pauli Z's expectations of all qubits in the circuit. As the eigenvalues of Z are [1,-1], it is expected that each qubit takes a value between -1 and +1.

The expectation of a qubit in the single-qubit circuit can be calculated as:

Expectation = $\langle \psi | Z | \psi \rangle$ Where $|\psi\rangle$ is the state vector.

For a multiple qubit system:

Expectation of 1st qubit = $\langle \psi | Z \otimes I \otimes \dots \otimes I | \psi \rangle = c1$

Expectation of 2nd qubit = $\langle \psi | I \otimes Z \otimes \dots \otimes I | \psi \rangle = c2$

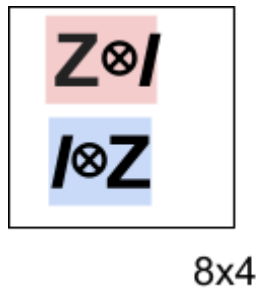
.

.

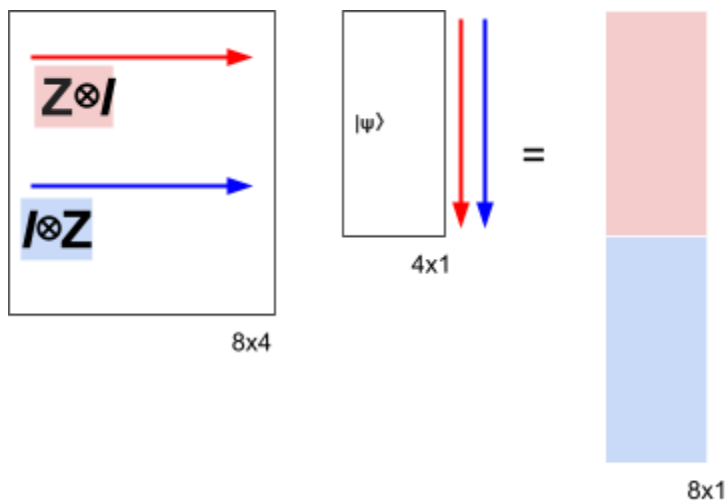
.

Expectation of nth qubit = $\langle \psi | I \otimes \dots \otimes Z | \psi \rangle = c3$

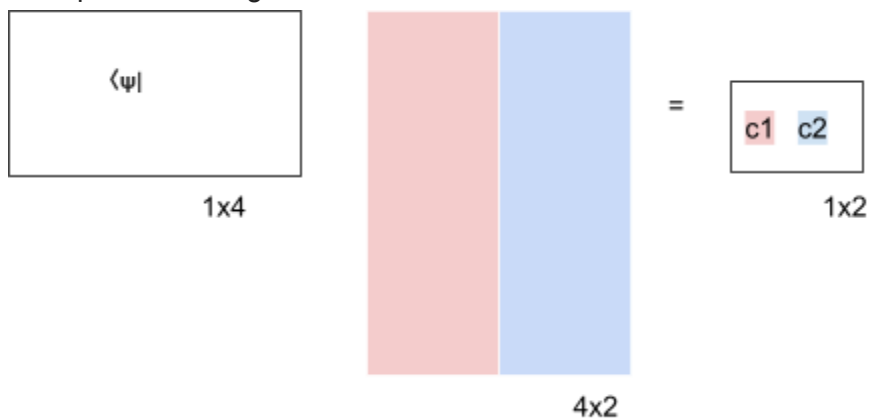
Let's say we have a 2 qubit system :
 We can concatenate the Transformations



Let's first multiply it with $|\psi\rangle$ that is 4x1 matrix



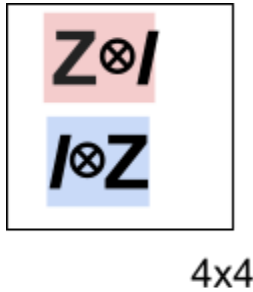
We get a result of an 8x1 matrix. And we reshape it to 4x2. And then, we multiply $\langle\psi|$ with the reshaped result to get the c1 and c2 values in a 1x2 matrix.



c1 and c2 here represent the expected values with respect to the 1st and 2nd qubits in the circuit.

In this way, we can calculate expectation values of a circuit consisting of n qubits in one go. **In the end, we get a result of a $1 \times n$ matrix that has expectations of all qubits.**

Note that the first concatenated matrix I mentioned, i.e.,


$$\begin{matrix} Z \otimes I \\ I \otimes Z \end{matrix}$$

4x4

I could not figure out how to concatenate this in one go. It takes $O(n)$ time to concatenate all the transformations. Once concatenation is done, all the expectation values are computed in one shot. I.e., $O(1)$.

So the total time complexity for this is $O(n)$