# ECE 382V: Introduction to Quantum Computing Systems from a Software and Architecture Perspective
# Homework 1

Sayam Sethi

September 2023

## Contents



## 1 Question 1

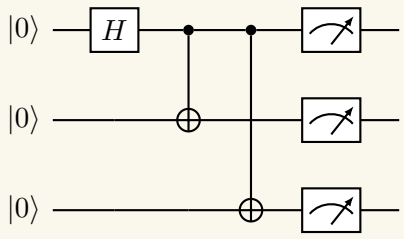
**Error Model**

The impact of errors can be analytically modeled by using various parameters accounting for the failure probabilities corresponding to different types of errors. Let us estimate the probability of successfully executing a 3-qubit bell-pair circuit.

*Note: All the values used in the following calculations are taken from the IBM Seattle machine.*

**Question.** *Estimate the error-rate assuming only gate and measurement errors occurs.*

**Answer.** The circuit for a 3-qubit bell-pair is given by,




Let the probability of H gate be $p_H$, the probability of error of a CNOT gate be $p_{CNOT}$ and the probability of error of a measurement be $p_M$. Then, the probability of successfully executing a 3-qubit bell-pair circuit is given by,

$$P_0 = (1 - p_H) \cdot (1 - p_{CNOT})^2 \cdot (1 - p_M)^3 \tag{1}$$

Plugging in values for $p_H = 6.338 \times 10^{-4}$, $p_{CNOT} = 2.253 \times 10^{-2}$ and $p_M = 5.32 \times 10^{-2}$, we get $P_0 = 0.810$.

**Question.** *Now append the error-model to include decoherence. How does the probability of successful circuit execution change?*

**Answer.** Let the decoherence time be $T_2$. Also assume that the time it takes for the Hadamard gate is $T_H$, the time it takes for the CNOT gate is $T_{CNOT}$ and the time it takes to measure the qubit is $T_M$. Then, the probability of successfully executing a 3-qubit bell-pair circuit is given by,

$$\begin{aligned} P_1 =& P_0 \cdot P_{decoherence}^3 \\ =& P_0 \cdot \left( \exp\left( -\frac{T_H + 2 \cdot T_{CNOT} + T_M}{T_2} \right) \right)^3 \end{aligned} \tag{2}$$

Plugging in values for $T_H = 35.56\,\text{ns}$, $T_{CNOT} = 660\,\text{ns}$, $T_M = 2000\,\text{ns}$, $T_2 = 59.94\,\mu\text{s}$, we get $P_1 = 0.685$.

**Question.** *How can you further improve the accuracy of this error model? Estimate the probability of success by including at least one additional parameter.*

**Answer.** Some simplifying assumptions made in the above model are:

1. The errors are assumed to be additive. However, in reality, the errors from different gates and decoherences might cancel out and hence we might have a higher probability of success.
2. We have only considered the error because of phase decoherence. Incorporating the error because of amplitude decoherence will give a lower chance of success.
3. The decoherence time for all the qubits was assumed to be the same. This will definitely not be the case in an actual system and the different decoherence times will affect the probability of success.
4. We have assumed that the probability of error of both CNOT gates is the same. However, in reality, the probability of error of the CNOT gate might be different for different pairs of qubits.
5. We have also assumed that the total time it takes to measure each qubit and the time it takes to execute each CNOT gate is the same. However, in reality, the times will be different and hence the decoherence probabilities will be different.

For this part, we will get rid of the simplifying assumption 2. The resultant error model becomes,

$$\begin{aligned} P_2 =& P_1 \cdot P_{decoherence(amplitude)} \\ =& P_1 \cdot \left( \exp\left( -\frac{T_H + 2 \cdot T_{CNOT} + T_M}{T_1} \right) \right)^3 \end{aligned} \tag{3}$$

Plugging in value for $T_1 = 91.12\,\mu\text{s}$, we get $P_2 = 0.613$.

**Question.** *What happens to the complexity of the error-model when you increase the circuit size beyond three qubits?*

**Answer.** As we increase the circuit size, the number of gates and measurements increase as $O(\mathsf{poly}(n))$, where $n$ is the number of qubits. We can also have exponentially many combinations of multi-qubit gates and measurements, therefore the error model can have $O(\mathsf{poly}(n))$ different terms. Thus, the probability of success will be negligible since it will be $\leq (\max p)^{O(\mathsf{poly}(n))}$.

## 2 Question 2

System Calibrations

Quantum systems are frequently calibrated to enable high fidelity quantum gate and measurement operations.

**Question.** *Why is system calibration non-trivial?*

**Answer.** System calibration is a non-trivial problem due to a lot of reasons:

1. The quantum systems are very sensitive to the environment and hence the calibration needs to be done frequently.
2. Each qubit is subjected to different operations and measurements and is surrounded by different neighbours. As a result, even if we want to apply the same gate to two different qubits, we will have to calibrate the gate differently for both the qubits.
3. For multi-qubit gates, we need to take into account the parameters for all the qubits involves in the gate. Therefore, the problem of finding the optimal gate parameters becomes exponentially hard as the number of qubits involve increase (since each qubit has multiple parameters involved, such as neighbours, idle gate frequencies, etc.).
4. When we calibrate the system, the qubits involved in the calibration go down and hence this stalls the quantum operations and slows down the computation. This not only affects the qubits that are being calibrated but also the qubits that have multi-qubit gates with these qubits under calibration.

**Question.** *What are the trade-offs involved in too frequent system calibrations and infrequent system calibrations?*

**Answer.**
**Too frequent system calibrations:** If we calibrate the system too frequently, then we will not be able to perform any useful computation on the system. This is because the system will be in the calibration mode for most of the time and hence we will not be able to perform any useful computation on the system. This can also potentially increase the decoherence errors of the system.
**Infrequent system calibrations:** If we do not calibrate the system frequently, then the system errors will continue to accumulate and compound thus rendering the results highly inaccurate. This would make the computation useless.

**Question.** *Most device providers opt for localized recalibrations as opposed to full-system calibrations. Why? How does this impact the "performance" of the system?*

**Answer.** A full-system recalibration will be computationally much heavier than a localized recalibration. Additionally, we will have to perform a full-system recalibration at the required frequency of the most error-prone regions. This would lead to useless computation for those qubits that do not require calibrations that frequently. Performing a calibration taking all qubits (and their neighbours) into account would also lead to a more complex calibration process since it would consider a larger set of parameters. Instead, a localized calibration process will be computationally more

efficient even if we consider the computation done per qubit (since we assume that the parameters of the qubits outside the neighbourhood are fixed). Localized calibration also allows for computation to take place parallelly elsewhere in the system and this leads to reduced system time and by extension, reduced decoherence errors.

**Question.** *In the class, we discussed the snake optimizer routine for performing large-scale system calibrations. What are the potential drawbacks of this technique? How do you think this can be improved?*

**Answer.** The snake optimizer routine is largely sequential in execution with respect to interdependent parameters and constraints. The optimizer is also inefficient with the model computation after every calibration. The vanilla algorithm does not consider local re-calibrations and performs system-wide calibrations which is very slow. Traversal for each thread has scope for optimization since heuristic based approach is not optimal for many situations and usually works the best for a small set of programs. The snake optimizer is also inefficient for architectures with many qubits and connections.
The vanilla implementation can be improvised by extending the implementation to support local re-calibration. This can easily be done by *deleting* the information about the region that has to be locally re-calibrated from $P^*$ and executing the snake optimizer algorithm. If we want to improve the runtime by compromising on the calibration accuracy, we can also get rid of some constraints from $R_P^{d_R}$ when performing calibration of a single element. Additionally, we can also optimize the runtime of the optimizer by caching information that can be reused with little to none modifications, such as $P_g^{d_P}, R_P^{d_R}, \mathsf{model}$.

# 3 Question 3

Compilation: Qubit Mapping and Routing

Compilers translate a sequence of high-level instructions (program) into a functionally equivalent sequence of low-level native gates (assembly). Qubit mapping and routing are two fundamental steps involved in this process.

**Question.** *How does the quality of initial qubit mapping or allocation impact the performance of the quantum computer for a given application?*

**Answer.** The initial qubit mapping determines the relative positions of the qubits. This is crucial to the routing decisions since qubits have limited connectivity, usually in a grid-like layout. Thus, if a pair of qubits that have a lot of 2-qubit gates between them are located far away, we would require a larger number of swap operations than if they were located close by. This would increase the execution time of the program and hence increase the decoherence error. Additionally, the number of gates would also increase the accumulated error and hence the program accuracy would also decrease.

**Question.** *How does the routing policy impact the quality of the solutions for a given application?*

**Answer.** A bad routing policy would require more swap operations for each two-qubit (or multi-qubit) gate to place the associated qubits next to each other. On the other hand, a good routing

policy would reduce the number of swap operations required significantly thus improving the quantum program run time. This would also reduce the decoherence error since a reduced execution time between measurements would reduce the effects of decoherence. Additionally, lesser gates also imply lesser accumulated errors thus increasing the program accuracy.
Different routing policies are implemented for different use cases, some might be implemented for machines that have a lower decoherence time or some that might have higher gate error rates. Similarly, the routing policy might also favour some applications over the others. For example, a routing policy might optimize for the number of swap operations required for a given program while another might optimize for the execution time of the program. Thus, the choice of the routing policy will affect the performance of the quantum computer for a given application.

**Question.** *There exists numerous routing policies in the compilation space. Why? Which would you like to opt for in order to run any given program of your choice?*

**Answer.** Different routing policies operate at different levels of space and time versus the quality of the routing. The solver based routing policies take exponential space and time however they give the best routing possible for any program. On the other hand routing policies like SABRE take very little space and time, but they do not guarantee the best routing possible and there are bound to be programs that will perform very poorly in execution time and/or accuracy when compiled with SABRE's routing policy. A different routing policy that maintains a working set of solutions and constantly prunes it (like ForeSight), takes more space and time in comparison to SABRE but guarantees better routing.
Thus, the choice of the routing policy can be determined by the required goals. If we want very low runtimes with higher accuracies, then we would want to choose the solver based approach. However, if we want to reduce the space and time complexity of the routing algorithm, then we would want to choose a routing policy like SABRE. If we want to have a balance between the two, then we would want to choose a routing policy like ForeSight.

**Question.** *What are the trade-offs involved in routing overheads and device topology?*

**Answer.** If the device has a simpler topology with limited connectivity, then the routing algorithms will have reduced complexity and hence the routing problem will be much simpler. On the other hand, if the device has larger connectivity, there are multiple paths to perform swaps between two pairs of qubits and hence the routing problem requires larger computation. Even if the connectivity remains the same but the number of qubits in the device are increased, the routing becomes more complex and will lead to larger overheads.

# 4 Question 4

Pulse Compilation

Most quantum algorithms can be described with circuit operations alone. When we need more control over the low-level implementation of our program, we can use pulse gates. Pulse gates remove the constraint of executing circuits with basis gates only, and also allow you to override the default implementation of any basis gate. Programmers can leverage this property to directly compile their programs into a series of pulses. Often these pulses are more compact compared to the pulses generated using only the basis gate set. For example, a SWAP pulse is typically shorter

than the pulse generated by concatenating the CNOT pulse thrice.

**Question.** *What are the potential advantages of this approach?*

**Answer.** The advantage to this approach is that we can reduce the program execution time significantly which will also lead to reduced errors. This also increases the scope of optimization since we have a larger set of pulses to choose from than we would have if we just used the basis set.

**Question.** *What are the key limitations in this approach?*

**Answer.** The drawback of this approach is that the compiler code will become more complex, and it will take more time to execute. Additionally, if we have a larger set of pulses to implement, this can lead to difficulties in the calibration of the system since we will have to choose more pulses from the same frequency band and thus the problem of choosing distant enough pulses becomes harder.

**Question.** *How do you think a programmer can best exploit this trade-off to their advantage?*

**Answer.** A programmer can inherit the flexibility of the pulses while not increasing the complexity of the compiler and the calibration algorithm by choosing different sets of pulses for different qubits. Not all qubits need to be implemented using pulse compilation and thus the programmer can exploit this fact to their advantage by only increasing the complexity of compilation for the hotspots, i.e., the qubits that are very active. For example, the programmer can only choose the qubits that have a lot of swap operations to be implemented using pulse compilation and the other qubits can be implemented using the basis set. This will reduce the complexity of the compiler and the calibration algorithm while still giving the programmer the flexibility of the pulses.

**Question.** *Can you transfer the pulse schedule from one machine to another?*

**Answer.** No, we cannot transfer the pulse schedule from one machine to another since the pulse schedule is dependent on the machine and the calibration of the qubits. Different machines will have different ways of implementing the same gates and hence the same sequence of pulses will not implement the same set of gates. Even if we have the same architecture of both machines, we still cannot transfer the schedule since the qubits for both the machines will have different calibration parameters and the same pulse schedule won't work for both the machines. Additionally, a system that has been compiled at the device level requires device level information and hence exporting them after optimization is not possible at all.