# Computing width and depth of a quantum program in tracer

This document defines the terms and describes our approach to computing depth and width of a quantum program (circuit) in the tracer component of the quantum execution environment. It also describes the limitations and a possible way to overcome them.

## Terms

### Tracer

Tracer is a component of a quantum system that allows “execution” of a quantum program without a quantum device and without complete simulation of a quantum device on classical hardware.

The word execution here should not be taken literally because measurements will not have actual results that would match those of a quantum system, instead they are mocked in certain way to allow quantum program to proceed. With this approach tracer can execute quantum programs much faster than the complete honest simulator would (full-state simulator) and collect statistical data about it. The collection of such statistical data rather than obtaining correct result is the main purpose of the tracer.

### Qubit Manager

Qubit manager is a component of a quantum system that maps program demand for qubits to available qubits in the system.

Quantum program may allocate and deallocate qubits just like classical program allocates and deallocates classical memory. It is up to the qubit manager to decide how to serve such allocation requests and whether to reuse previously released qubits or use fresh ones.

### Circuit Width

Width of a quantum circuit (or quantum program), roughly defined, is the number of qubits needed to execute it.

There’re several uncertainties and details in this definition that need to be clarified. First, we cannot statically analyze all branches of a program and make conclusions about program’s width and depth in all possible cases. So, we measure width of one specific execution of a quantum program. Second, we count qubits that the program uses while executing on a Tracer. Execution on a quantum device may require other layers between the actual quantum device and the program such as a fault-tolerant encoding, which could increase the number of needed qubits significantly. This increase is out of scope of the Tracer.

### Circuit Depth

Depth of a quantum circuit (or quantum program), roughly defined, is the time it takes to execute it measured in appropriate units.

Only quantum part is measured. We don’t account for classical computation even if it takes considerable time. Often the time is quantized, and the unit of measurement is one such time step (often denoted as 1d – one depth unit). Typically, many quantum gates can be executed in parallel. In this case we don’t add up all times of all gates to get the depth. For example, if five gates each taking 2 depth units can all be done in parallel, the depth of such execution is still two depth units.

## Objective

The objective is to collect two pairs of numbers. First pair – minimum width and corresponding depth. To compute this pair we try to execute a program in such a way that it would take least amount of qubits and then measure the number of time steps required to execute it.

Second pair – minimum depth and corresponding width. To compute this pair we try to execute a program in such a way that it would take least amount of time steps and then count the number of qubits needed to execute it in such way.

These two pairs, minimum depth + corresponding width and minimum width + corresponding depth are the two extremes out of many choices in between. Someday it would be nice to find other pairs in the range as they may correspond to a better overall strategy (see sample chart below). However, this work is out of the scope of the current project. There’s a planned feature to compute minimum depth given certain width.

As mentioned before, we cannot statically analyze an arbitrary program and deduce its behavior in all possible cases (assuming no restrictions are in place). We also do not automatically write a different but equivalent program which would have the optimal depth or width. We can only measure one execution of a program. That said we do make equivalent transformations to the specific execution of a program to obtain desirable statistics. For example, we may decide to execute multiple gates in parallel if it doesn’t change the result. Such parallel execution will result in lower depth. Specific approach will be described later.

## Role of Qubit Manager

Qubit manager plays crucial role in width and depth calculation via qubit reuse. Consider the following example (a and b are different qubits):

Allocate(q1)  
Gate1(a, q1)  
Deallocate(q1)  
Allocate(q2)  
Gate2(b, q2)  
Deallocate(q2)

Gates Gate1 and Gate2 operate on different qubits so they can be executed simultaneously. Assuming it takes one depth unit to execute Gate1 and Gate2, executing them simultaneously also takes one depth unit. That is if we also assume qubits q1 and q2 are different. However, this program deallocates qubit q1 before allocating qubit q2. Qubit manager may decide to reuse the same underlying qubit for both names. This will result in the following transformation:

Gate1(a, c)  
Cleanup(c)  
Gate2(b, c)

These two gates operate on the same qubit c and therefore Gate2 can only be executed after Gate1. They no longer can be executed at the same time. Executing Gate1 and Gate2 one after another takes two depth units. So, we see that reusing qubits may increase the depth of the quantum circuit.

Since different qubit manager strategies affect reuse of qubits, we need two different modes of qubit manager to obtain two different extremes – minimum depth and the minimum width. The strategy choice is controlled via EncourageReuse parameter, which in turn is controlled via OptimizeDepth parameter in tracer configuration.

## Minimum Depth

To obtain minimum depth we set OptimizeDepth to true. In this case qubit manager is requested not to reuse qubits. When qubits are not reused, no additional dependencies are introduced (beyond ones that are present in the quantum program) and more gates can be executed in parallel.

Then we employ greedy linear algorithm to reorder gates and execute them as early as possible. For that we introduce availability time for each qubit as described here: <https://ms-quantum.visualstudio.com/CircuitTools/_git/ResearchTools?path=%2FDocs%2Fdistributed-depth-calculation.md>

The resulting approach gives us minimum depth of the circuit. To obtain corresponding width we count the number of qubits used. This will result in a pair of numbers – minimal depth and a width, compatible with the depth. Calculated width is an upper bound on minimum width, given the minimum depth constraint.

Although simple, this is not the best width **given the minimum depth** of the circuit. In other words, we may reorder some gates, reuse some of the qubits, and still get the minimum depth with lower width. Most of these approaches are computationally intensive and may be out of scope of the first implementation.

One approach that can be used in future is the greedy coloring of an interval graph described here: <https://en.wikipedia.org/wiki/Greedy_coloring>. This approach would not perform any reordering to minimize width given the depth, which is computationally hard. Instead, the ordering would be the same as in minimal depth calculation, but it would compute minimum width assuming this ordering (and hence depth).

## Minimum Width

Algorithms that compute minimal width assuming gate reordering is possible are computationally intensive and are out of scope of the first implementation. Therefore, a greedy approach is employed.

First, we request qubit manager to reuse qubits by setting OptimizeDepth to false in tracer configuration. Such simple greedy reuse of qubits will result in a minimum width when we do not allow reordering (execute gates in the order in which they are executed by the quantum program). As a result, obtained width is a reasonable upper bound on the minimum width.

Once the width is obtained, we employ availability time approach to reorder gates and compute minimum depth corresponding to the obtained width.

This will result in a pair of numbers – reasonable upper bound on minimal width and a minimum depth, compatible with this width.

## Allocation/Deallocation optimization

One further optimization that can be employed is allocation/deallocation time optimization. Such optimization will ignore specific times of user allocation and deallocation of qubits, instead such times will be derived from first and last use of the qubits. For example, at some point in time t1 user may be done with qubit q1. Then several gates are executed on other qubits and then qubit q1 is deallocated at time t2 > t1. In this case user qubit q1 can be deallocated at time t1 rather than t2 and underlying qubit can be reused earlier.

We do not use such optimizations in our first approach.

## Parameters for width and depth calculations

Besides quantum program itself, the following parameters influence measured width and depth.

### Single gate time

It may take different time to execute different gates. While the simplest case is when every gate takes one depth unit, such approach isn’t always sufficient to evaluate a circuit. Typically, a floating-point number representing gate’s execution time should be provided for each supported gate. This can be done by setting TraceGateTimes in Tracer configuration.

### Gate decomposition

While quantum program may include any gates, underlying quantum device or a target fault-tolerant encoding will not support arbitrary gates. Gates coming from user program need to be decomposed to obtain the sequence of gates that is compatible with the underlying model. Width and depth measurements should be done on this modified sequence of gates. Such decompositions should also be provided.

### Measurements

Measurements directly affect depth calculations and may also affect width indirectly. Consider the following fragment:

if ( Measure( q1 ) ) {  
 X( q2 )  
}

Further suppose that both Measure and X gates take one depth unit. Without knowing how the outcome of a measurement affects execution path it is not possible to obtain proper depth. This if statement may look like the following sequence to the depth algorithm:

M(q1)  
X(q2)

As these gates operate on different qubits they can be executed in parallel and hence the depth of this circuit is one depth unit. However the X gate cannot be executed at the same time as Measurement in the original program because the outcome of measurement is not known and the depth of the original program is actually two depth units.

To make this information known to the depth algorithm classical controls may be introduced. This is also outside of the scope of the first implementation. As a result, the depth calculated for a program that uses measurements may be lower than its actual depth.

## Per-scope width and depth

Please note that computation of a compatible width and depth pair is implemented in the DepthCounter metric collector. WidthTracker is a different metric collector and implements a different approach to circuit width.

The DepthCounter metric collector computes compatible width and depth for the entire quantum program as described above. It also computes width and depth for each scope (callable) in the program. This brings additional challenges – there may be multiple definitions of width and depth for a scope within a program (rather than taken separately). We assume the following definitions:

### Width

Width of a scope within a program is the number of additional qubits needed to execute this scope (operation). It may be very different than the width of the same operation taken separately. For example, suppose that ten qubits were needed to execute operation A and are now released. If operation B demands five qubits and this demand is satisfied by reusing five of the ten qubits previously released, the width of the scope B within the program is 0 as no additional qubits were needed.

### Depth

As implemented, depth of a scope within a program is the difference between maximum available time of all input qubits at the beginning of operation and maximum available time of all qubits used by operation at the end of operation.