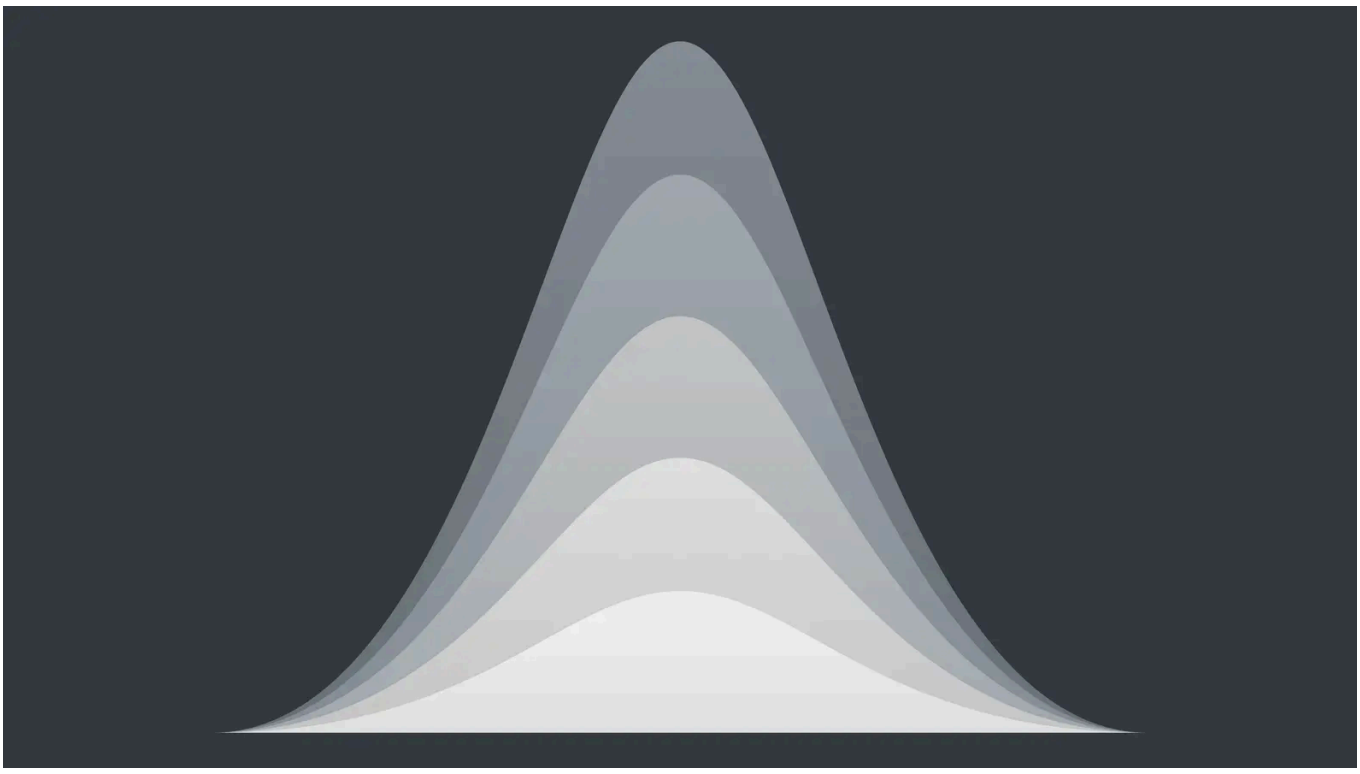
New fractional gates reduce circuit depth for utility-scale workloads

Fractional gates are a new type of quantum logic gate that can help to increase the efficiency of utility-scale experiments.



Date

7 Nov 2024

Authors

Daniella Garcia Almeida

Blake Johnson
Robert Davis

Topics

Enablement Software Updates

Share this blog

IBM® is introducing a new type of quantum logic gate to IBM Quantum Heron™ QPUs. They're called "fractional gates," and they are a powerful new tool for reducing the circuit depth of quantum workloads. Fractional gates introduce helpful new efficiencies for building utility-scale quantum experiments, and this is particularly true of quantum experiments designed to simulate nature. So, how do they work?

When running large experiments with 100+ qubits and thousands of gates, it's important to take advantage of any opportunity to reduce the depth of your quantum circuits. Shorter circuits allow us to do more with the noisy quantum hardware we have today. Fractional gates help to reduce circuit depths by adding two useful new ISA instructions to our quantum fleet.

These new ISA instructions give you the ability to execute a single- or two-qubit rotation in the form of either an $R_X(\theta)$ gate or an $R_{ZZ}(\theta)$ gate. You can use these gates to significantly reduce gate counts in just about any quantum circuits containing single- and two-qubit rotations — such as, for example, the quantum circuits we use to simulate the dynamics of quantum systems.

An important note on pulse-level control for IBM Quantum QPUs

Pulse-level control will be removed from the service on **February 3, 2025.** This decision is part of our ongoing efforts to focus more of our attention and resources on higher-level services aimed at supporting utility-scale experiments and the search for quantum advantage.

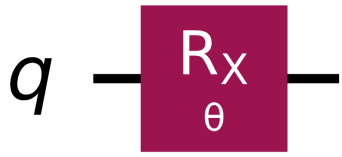
If you are someone who has been using IBM Quantum's pulse-level control features for optimal control studies, we encourage you to explore tools and resources like Qiskit Dynamics that enable such studies in simulation and may provide a replacement for your specific use case.

We mention this deprecation here because one of the most common use cases for pulse-level control is the direct implementation of the single- and two-qubit rotations described in the introduction to this article. As we'll see below, the new fractional gates provide a convenient replacement for that functionality.

Why are traditional $R_X(\theta)$ and $R_{ZZ}(\theta)$ gates less efficient?

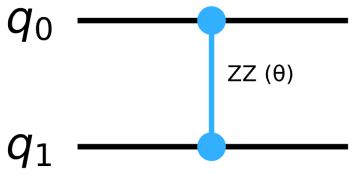
To give you an idea of the computational cost-savings fractional gates can bring to your final circuit depth, let's look at how the Qiskit transpiler typically decomposes an $R_X(\theta)$ and $R_{ZZ}(\theta)$ gate. We'll start by first creating two circuits, one with an $R_X(\theta)$ gate and one with an $R_{ZZ}(\theta)$ gate. Here's the $R_X(\theta)$ gate:

```
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
rx_qc = QuantumCircuit(1)
param = Parameter('θ')
rx_qc.rx(param, 0)
```



And here's the $R_{ZZ}(\theta)$ gate:

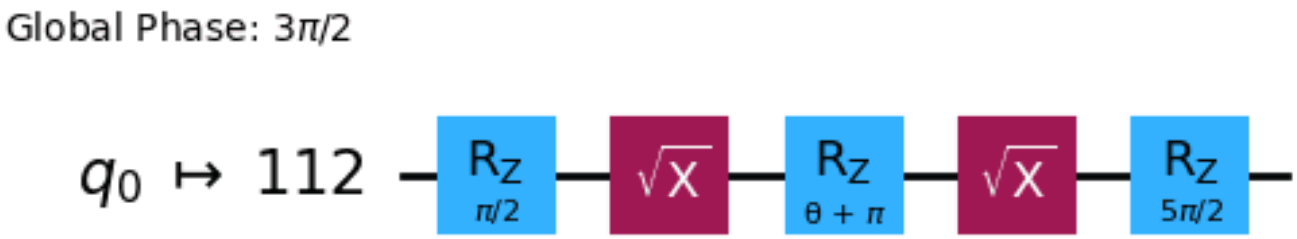
```
rzz_qc = QuantumCircuit(2)
rzz_qc.rzz(param, 0, 1)
rzz_qc.draw('mpl')
```



Then, when we transpile them, the $R_X(\theta)$ gate becomes:

```
from qiskit_ibm_runtime import QiskitRuntimeService
# Decomposition of an RX(θ) gate using the IBM Quantum QPU basis
service = QiskitRuntimeService()
backend = service.backend("ibm_torino", use_fractional_gates=True)
```

```
rx_transpiled_circuit.draw('mpl', idle_wires=False)
```

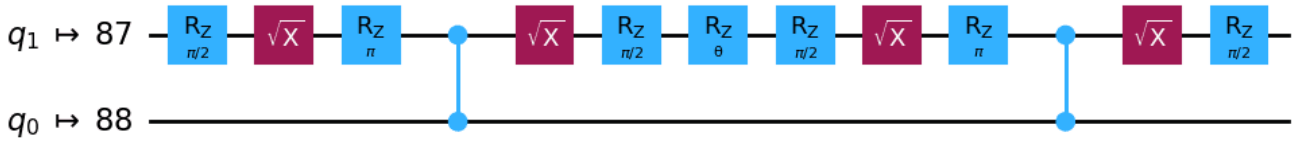


This is a *substantial* increase in gate depth. The circuit has quintupled in size from just a single quantum logic gate to five.

In this case, it's worth noting that only the $\sqrt{X}$ gates contribute to the gate error. This is due to the specific way we execute the RZ gate in the IBM Quantum fleet. Still, even if we only look at the $\sqrt{X}$ gates, that's double the original gate depth, and that doubling becomes significant when we extrapolate to a circuit comprising 100+ qubits.

Let's now take a look at how the $R_{ZZ}(\theta)$ gate is decomposed:

```
rzz_transpiled_circuit = pm.run(rzz_qc)
rzz_transpiled_circuit.draw('mpl', idle_wires=False)
```



That's an even larger increase! To execute an $R_{ZZ}(\theta)$ gate, we need to execute *two* CZ gates, which perform the action of a CNOT gate, along with several more single-qubit gates.

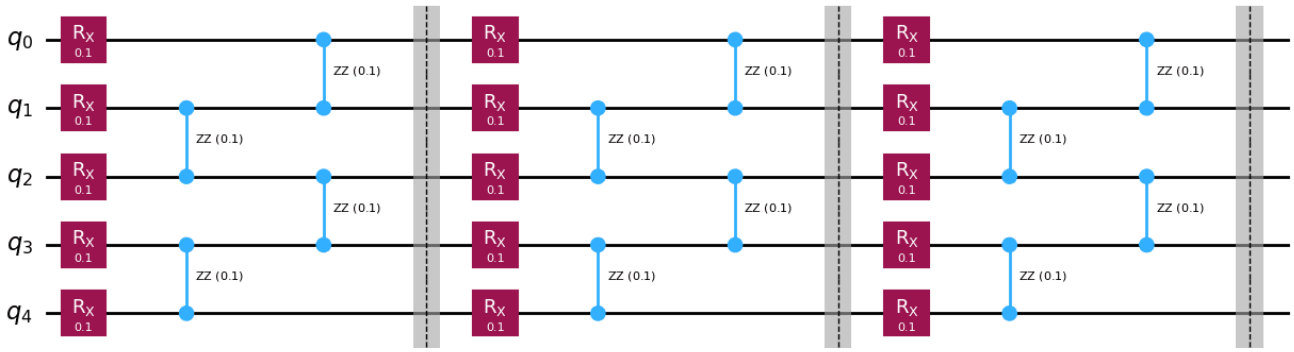
However, using the new fractional gates, we can execute these single and two-qubit rotations directly and avoid this issue of increasing circuit depth.

How to use fractional gates: A simple example based on the Ising model

To quickly demonstrate how to use the new fractional gate instructions, let's look at an example circuit that simulates the dynamics of a transverse field Ising model. This is a relatively simple quantum system to simulate and will serve as a useful case study demonstrating the power of fractional gates.

In this model, each qubit represents a single site in either a one-dimensional line or a two-dimensional lattice. For this example, let's simulate the time evolution of a one-dimensional chain of sites. We can construct the circuit in the following way:

```
num_qubits = 5
num_time_steps = 3
rx_angle = 0.1
rzz_angle = 0.1

ising_circuit = QuantumCircuit(num_qubits)
for i in range(num_time_steps):
    # rx layer
    for q in range(num_qubits):
        ising_circuit.rx(rx_angle, q)
    for q in range(1, num_qubits-1, 2):
        ising_circuit.rzz(rzz_angle, q, q+1)
    # 2nd rzz layer
    for q in range(0, num_qubits-1, 2):
        ising_circuit.rzz(rzz_angle, q, q+1)
    ising_circuit.barrier()
ising_circuit.draw('mpl')
```

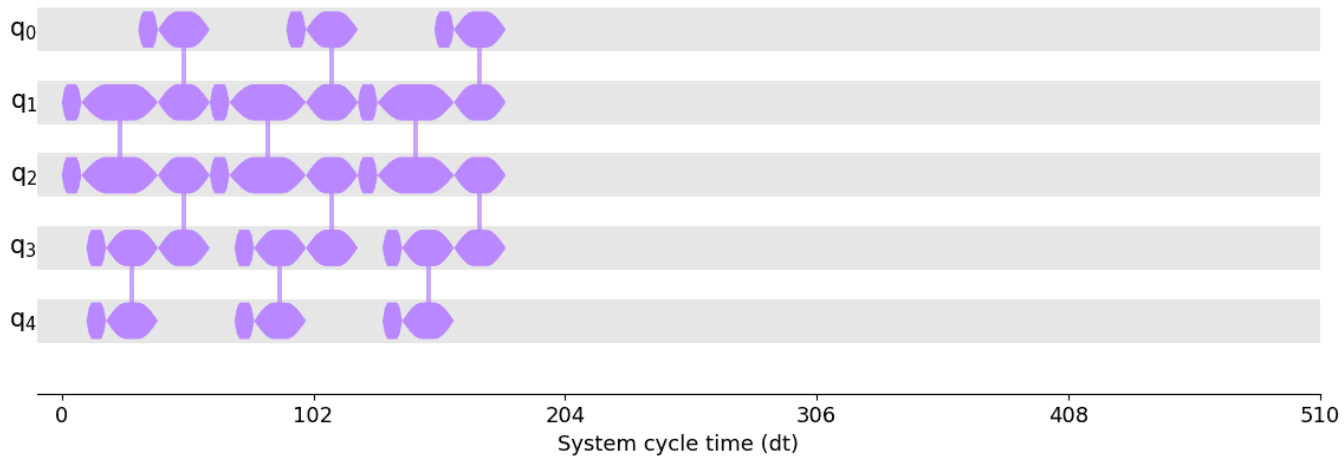


Now, to take advantage of fractional gates, all we have to do is specify a backend that supports them! The transpiler will know what to do from there:

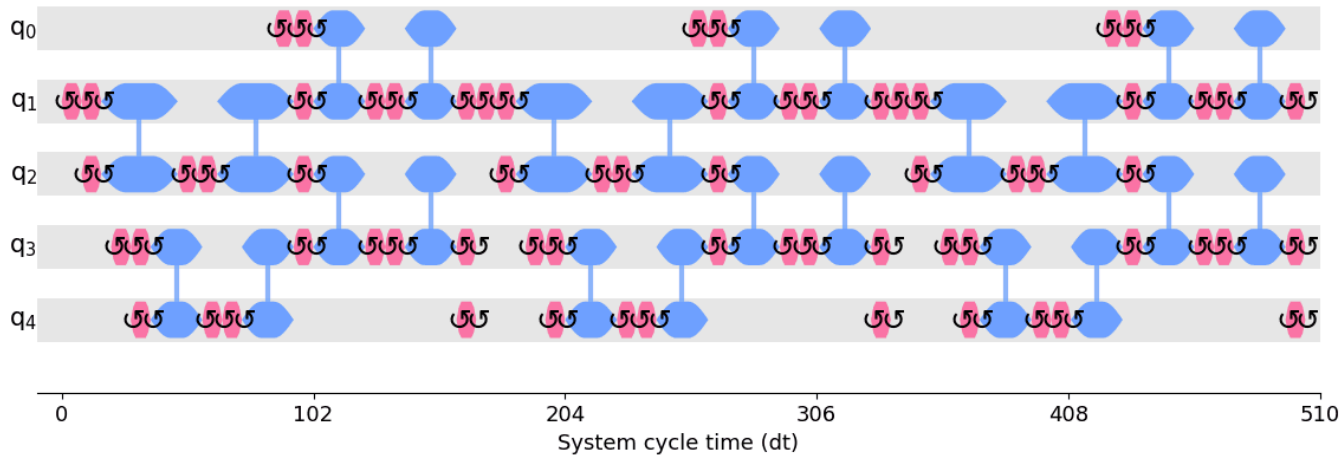
```
from qiskit_ibm_runtime import QiskitRuntimeService

service = QiskitRuntimeService()
backend_fractional = service.backend("ibm_torino", use_fractional_gates=True)
```

```
ising_circuit_transpiled = pm_fractional.run(ising_circuit)
ising_circuit_transpiled.draw('mpl', idle_wires=False)
```

To see the difference this makes, let's now compare the scheduled circuit time we get using a backend with fractional gates enabled vs. one that doesn't have fractional gates enabled. Here's what we get using fractional gates:



And here's what we get without fractional gates—as you can see, system cycle time is more than doubled:



Fractional gates limitations

Fractional gates are a powerful tool, but they do come with a few limitations that you should keep in mind. For one thing, you cannot use fractional gates with every backend. You can use the following code to query which backends support them:

```
fractional_gate_backends = []
for backend in service.backends():
    if 'rzz' in backend.target.operation_names:
        fractional_gate_backends.append(backend)
```

Alternatively, you can use something like the following to query which backends support fractional gates:

This second code example does the same thing as the first example, but keeps the details under the cover. The same `filters` can be passed to `least_busy`.

Another consideration is that, at least for now, you will not be able to use fractional gates in combination with dynamic circuits in the same job, and you cannot use them with primitive-based error mitigation methods like PEC, ZNE with PEA, or Pauli twirling. You *will*, however, be able to use fractional gates alongside dynamical decoupling and the T-REx method. In the future, we may be able to relax the current limitations.

You'll find a helpful chart outlining which error mitigations you can and cannot use with fractional gates in our documentation.

Getting started with fractional gates

We hope you'll experiment with the new fractional gates to see how they can help you reduce gate depth for your utility-scale workloads. For more guidance on how to use—and how not to use—fractional gates, be sure to check our documentation here.

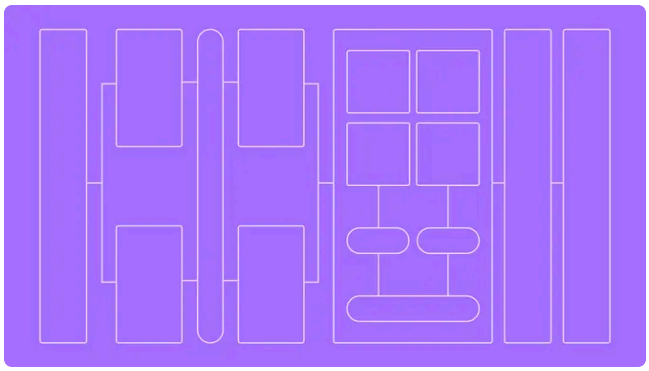
Sign in to Platform ↗

Start using our 100+ qubit systems

View pricing ↗

Keep exploring

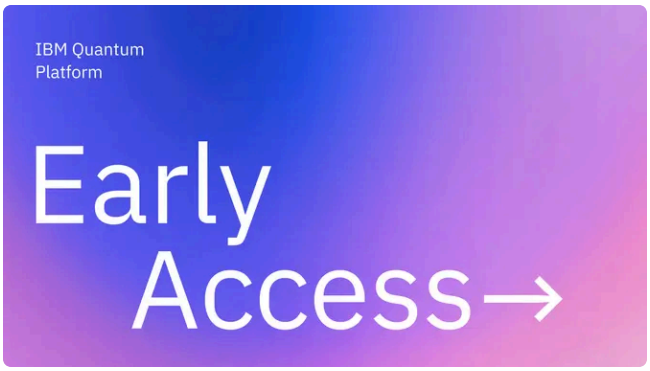
View all blogs →



How real researchers are using Qiskit circuit functions to accelerate their experiments

5 Mar 2025 • Suhare Nur, Sanket Panda, Francisco Martin, SheshaShayee Raghunathan, Robert Davis

Enablement | Network | Software



The next evolution of IBM Quantum Platform: How to prepare for the transition

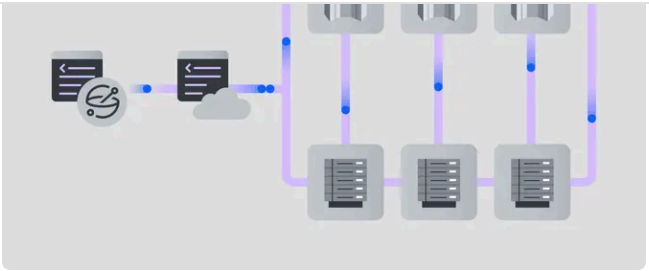
26 Feb 2025 • Julianna Roberts, Leron Gil, Kayla Lee, Fran Cabrera, Sean Dague, Robert Davis

Enablement | Software Updates | Qiskit

Course updates and new learning pathways on IBM Quantum Learning

23 Oct 2024 • Olivia Lanes, Christopher Porter, Jennifer Janechek, Robert Davis

Enablement



Qiskit Serverless sets the stage for Qiskit Functions in the cloud

4 Sep 2024 • Kaelyn Ferris, Sanket Panda, Robert Davis

Enablement Software Qiskit