Qiskit Summer School 2024 Notes

Taylor Kimmel

April 9, 2025

Contents

First Steps

Youtube Playlist of the Summer School:
https://www.youtube.com/playlist?list=PLOFEBzvs-Vvr-GzDWlZpAcDpki5jUqYJu

Get Qiskit Installed:
https://docs.quantum.ibm.com/guides/install-qiskit
Get IBM Setup:
https://docs.quantum.ibm.com/guides/setup-channel

TestSetup.py - Makes sure that you can connect to IBM and get a result

Optimize for Hardware

Required - *transpile*[Transpilation]: abstracts the quantum circuit into a circuit that can run on target hardware
Optional - *verify*[Verification] circuit with simulation

Real quantum device constraints:
Basis Gate Set
Only a limited set of gates can be executed directly on the hardware
Other gates must be rewritten in terms of these basis gates

Qubit connectivity
Only certain pairs of qubits can be directly interacted with each other

Errors
Each operation has a chance of error, so circuit optimizations can greatly affect performance

Challenge is running abstract circuit on a specific quantum device.
The solution is **transpilation** - convert abstract circuit into ISA (instruction set architecture) circuit

Transpilation

Transpilation Terms:

Term	Definition	Orchestra Analogy
Pass	A standalone circuit or metadata transformation	An instrument
Pass Manager	A list of transpiler passes grouped into a logical unit	An instrument section
Staged Pass Manager	A list of pass managers, with each one representing a discrete stage of atranspilation pipeline	The conductor

Transpile a Circuit with Qiskit SDK:

1. Choose which device backend you want to target
2. Create a preset staged pass manager with your desired optimization level
3. Run the staged pass manager on the circuit

Python Code:

```
service = QiskitRuntimeService()
backend = service.backend("ibm_brisbane")
pass_manager = generate_preset_pass_manager(optimization_level=3,
                                            backend=backend)
isa_circuit = pass_manager.run(circuit)
```

Transpiler Stages:

1. Initialization
   The circuit is prepared for transpilation, e.g., multi-qubit gates are decomposed into two-qubit gates
2. Layout
   The abstract qubits of the circuit are mapped to the physical qubits on the device

3. Routing
   Swap gates are inserted to enable interactions between qubits that are not physically connected

4. Translation
   The gates of the circuit are translated to the basis gate set of the device

5. Optimization
   The circuit is rewritten to minimize its depth (# of operations) to decrease the effect of errors

6. Scheduling
   Delay instructions are added to align the circuit with the hardware's timing

Verification

Simulation Tools

Qiskit SDK reference primitives. Exact simulation, but small circuits only and no noise simulation.

Qiskit Runtime local testing. Provides "fake" backends to model each quantum machine.

Qiskit Aer. Ecosystem project for simulation, including

- larger circuits
- stabilizer circuits
- noise models

Problem: the runtime cost of simulating quantum circuits scales exponentially with the number of qubits.
$\sim$ 50+ qubits cannot be simulated

Techniques for large circuits:

1. Test smaller versions of circuit

2. Modify circuit so that it becomes classically simulatable: stabilizer circuit aka Clifford circuit

Execution

Primitives encapsulate the output of a quantum circuit

Sampler Primitive:
Output is mapping of bitstrings to counts, e.g., {'0': 12, '1': 9}

Estimator Primitive:
Output is the *expectation value* of an *observable*, e.g., the net spin of a system. Circuit should not include measurements.

To run a circuit on quantum hardware:

1. Initialize the Qiskit Runtime service
2. Choose a hardware backend
3. Initialize a Qiskit Runtime primitive with your chosen backend
4. Invoke the primitive with your circuit

Example Sampler Code:

```
import numpy as np
from qiskit.circuit.library import IQP
from qiskit.quantum_info import random_hermitian
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as
                                  Sampler

service = QiskitRuntimeService()

backend = service.least_busy(operational=True, simulator=False,
                                  min_num_qubits=127)

n_qubits = 127

mat = np.real(random_hermitian(n_qubits, seed=1234)
circuit = IQP(mat)
circuit.measure_all()

pm = generate_preset_pass_manager(backend=backend,
                                  optimization_level=1)
isa_circuit = pm.run(circuit)

sampler = Sampler(backend)
job = sampler.run([isa_circuit])
result = job.result()
```

The input to the Estimator primitive is a list of *primitive unified blocs (PUBs)*. Each PUB consists of:

- A single circuit without measurements
- One or more observables
- (Optional) One or more parameter values

Example Estimator Code:

```
import numpy as np
from qiskit.circuit.library import IQP
from qiskit.transpiler.preset_passmanagers import
                                  generate_preset_pass_manager
```

```
from qiskit.quantum_info import SparsePauliOp, random_hermitian
from qiskit_ibm_runtime import QiskitRuntimeService, EstimatorV2 as
                                   Estimator

service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False,
                                  min_num_qubits=127)
estimator = Estimator(backend)

n_qubits = 127

mat = np.real(random_hermitian(n_qubits, seed=1234)
circuit = IQP(mat)
observable = SparsePauliOp("Z" * n_qubits)

pm = generate_preset_pass_manager(backend=backend,
                                  optimization_level=1)
isa_circuit = pm.run(circuit)
isa_observable = observable.apply_layout(isa_circuit.layout)

job = estimator.run([isa_circuit, isa_observable])
result = job.result()

print(f" > Expectation value: {result[0].data.evs}")
print(f" > Metadata: {result[0].metadata}")
```

You can customize Qiskit Runtime behaviour

Shots
The number of measurements.
More shots reduce statistical error but increase running time.

Error Suppression
Use dynamical decoupling to reduce decoherence during execution.
Caveat: requires delays between gate executions, so it's not always possible.

Error Mitigation
Reduce the effects of device noise after execution.

- Twirled Readout Error eXtinction (TREX) measurement twirling
- Zero Noise Extrapolation (ZNE)

Downside: computational overhead

Execution Modes:

- Single job
- Batch: multiple concurrent jobs
- Session: iterative workload

Benefits of batch mode versus a single job with multiple circuits:

- Better concurrency of classical processing
- Get individual results sooner

Batch Mode:

```
from qiskit_ibm_runtime import SamplerV2 as Sampler, Batch

max_circuits = 100
all_partitioned_circuits = []
for i in range(0, len(circuits), max_circuits):
    all_partitioned_circuits.append(circuits[i : i + max_circuits])
jobs = []
start_idx = 0

with Batch(backend = backend):
    sampler = Sampler()
        for partitioned_circuits in all_partitioned_circuits:
            job = sampler.run(partitioned_circuits)
            jobs.append(job)
```

Sessions allow iterative workloads, like VQE
Run a job, based on result of the job send a new job

Sessions:

```
from qiskit_ibm_runtime import EstimatorV2 as Estimator
from qiskit_ibm_runtime import QiskitRuntimeService, Session

# Initialize Qiskit Runtime service
service = QiskitRuntimeService()
backend = service.backend("ibm_brisbane")

with Session(backend=backend):
    estimator = Estimator()
    # invoke the Estimator as usual
```

Postprocessing

Postprocess technique: visualize results
With Sampler output, use **plot_histogram()** from **qiskit.visualizations**

Postprocess technique: post-selection
Discard outputs known to be incorrect based on prior knowledge, e.g., if you know outputs must match a certain pattern

Postprocess technique: circuit knitting

1. During the optimize for hardware step, decompose the problem into smaller circuits, i.e., "circuit cutting"
2. Execute smaller circuits
3. "Knit" the results into a reconstruction of the original circuit's outcome