OAK RIDGE National Laboratory

June 8, 2025, 1st International Workshop on Software Frameworks for Integrating Quantum and HPC Ecosystems, ICS 2025, Salt Lake City

Q-IRIS: The Evolution of the IRIS Task-Based Runtime to Enable Classical-Quantum Workflows

PRESENTED BY ELAINE WONG

Collaborators:

Narasinga Rao Miniskar, Vicente Leyton-Ortega, Mohammand Alaul Haque Monil, Seth R. Johnson

Advisors and PIs:

Jeffrey S. Vetter, and Travis S. Humble

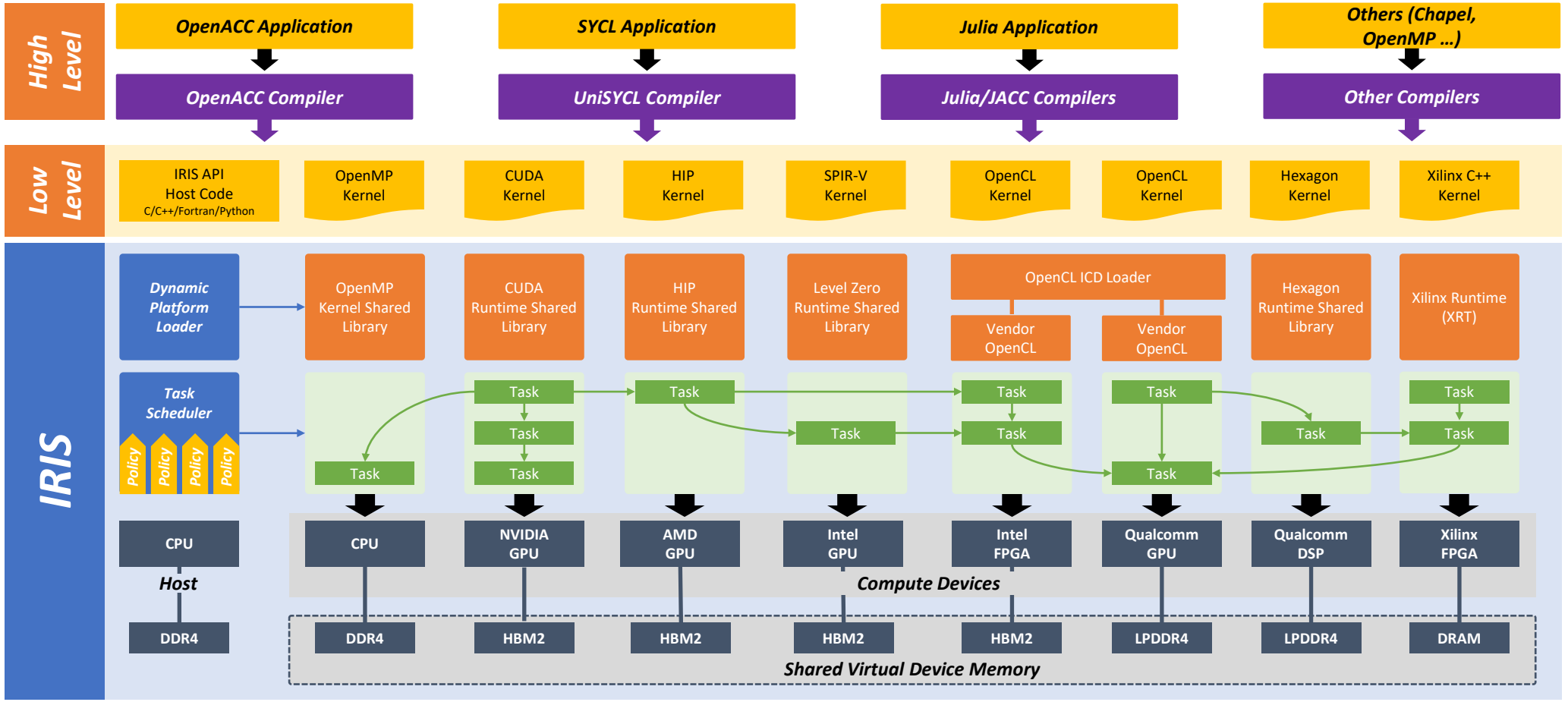
U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

Initial Objectives for Q-IRIS

- **Design** a portable **hybrid programming paradigm** based on extending the **IRIS heterogeneous runtime system** to accommodate **parallel, asynchronous quantum computation**.
- **Utilize** the execution engine for the Quantum Intermediate Representation (QIR-EE) and IBM's qiskit programming framework as runtimes to accelerate quantum programs for computations on quantum hardware.
- **Showcase** usage of this setup with an example.

IRIS Heterogeneous Runtime Framework



Heterogeneous Multi-Device Multi-Stream Asynchronous Runtime

https://github.com/ORNL/iris https://iris-programming.github.io/ https://code.ornl.gov/brisbane/iris.git

Kim, Jungwon, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-8. IEEE, 2021.

IRIS Heterogeneous Runtime Framework

Heterogeneous Platform Model

CPUs

GPUs: Nvidia, Intel, AMD

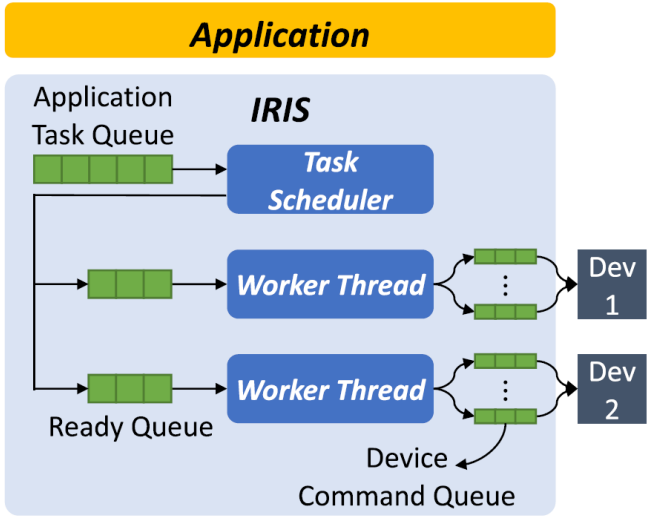
(new) QPUs: Backends accessible via QIR-EE and qiskit

Task Programming Model

Create synchronous / asynchronous tasks

Add dependencies

Create task graphs



Memory Model

Handles heterogenous memory address spaces for application data objects

Automatic data movement (DMEM)

Automatic data flow analysis

Automatic Flush/write of data objects to host

Heterogenous data transfer policies

Execution Model

Supports dynamic and static task mapping policies

Customization of policies

Automatic Multi-device Asynchronous Multi-stream Heterogeneous tasks execution (MASH)

https://code.ornl.gov/brisbane/iris.git

Kim, Jungwon, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-8. IEEE, 2021.

Kernels as Tasks

Classical

Code that contains instructions to be run on cpus/gpus.

Quantum

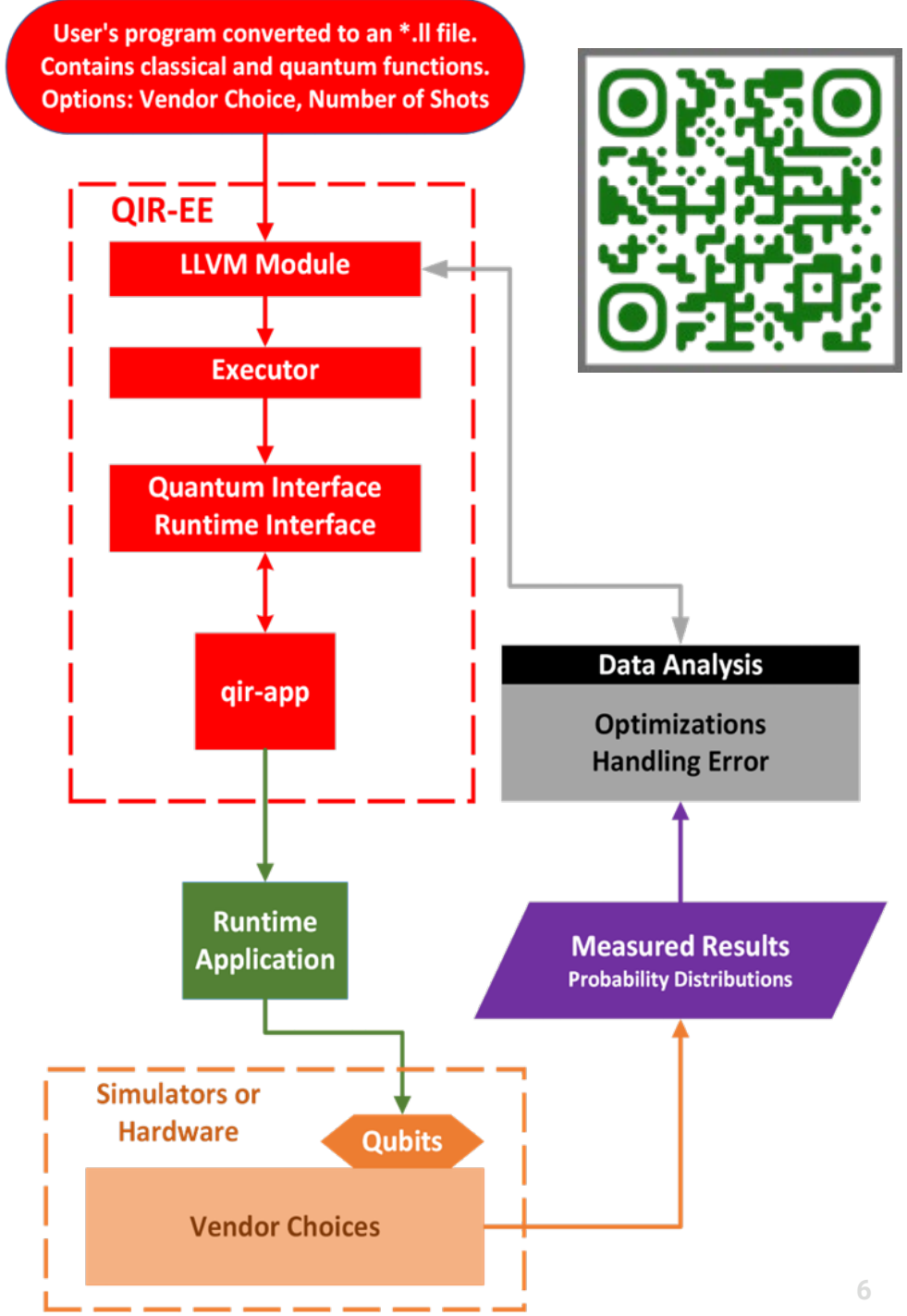
Circuits to be executed on quantum hardware.

Hybrid

Classical-Quantum Programs
QIR or QASM Files

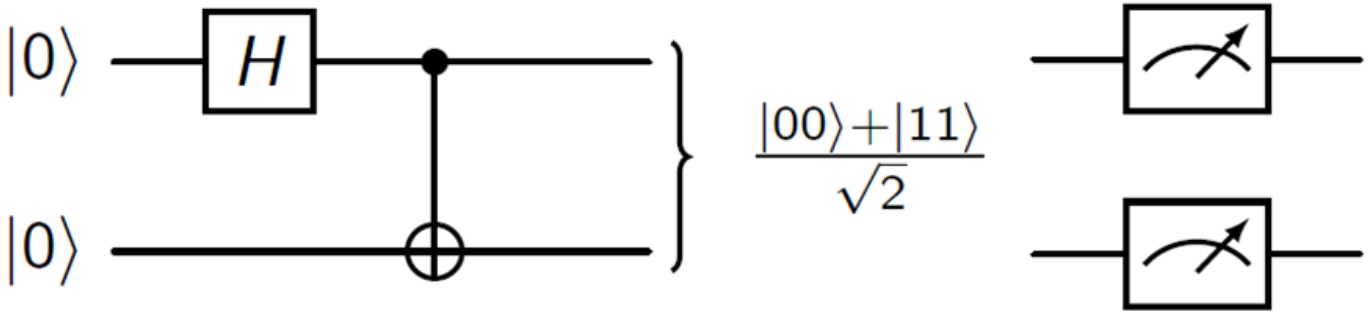
QIR-EE

- ORNL-developed software that helps to control the parsing and execution of a quantum program adhering to the specification for the Quantum Intermediate Representation (QIR).
- QIR defines how to represent quantum programs within the LLVM IR.
- The execution engine for QIR (QIR-EE) enables **testing and evaluation of computing applications across multiple quantum and classical vendor platforms**.
- This provides an alternative to the popular IBM's qiskit.



https://github.com/ORNL-QCI/qiree

What is QIR?

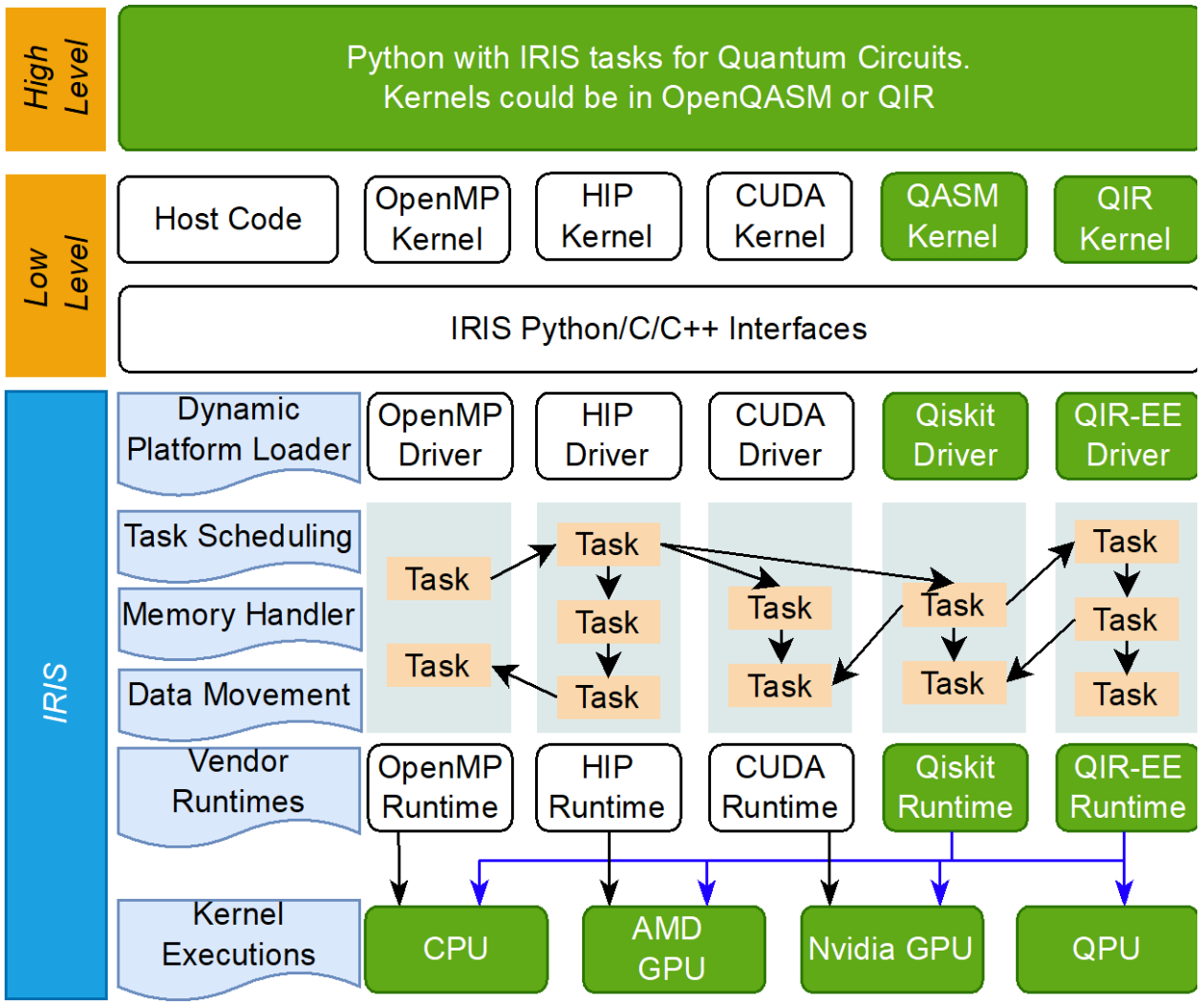



```
%Qubit = type opaque
%Result = type opaque

define void @main() #0 {
entry:
  call void @__quantum__qis__h__body(%Qubit* null)
  call void @__quantum__qis__cnot__body(%Qubit* null, %Qubit* inttoptr (i64 1 to %Qubit*))
  call void @__quantum__qis__mz__body(%Qubit* null, %Result* null)
  call void @__quantum__qis__mz__body(%Qubit* inttoptr (i64 1 to %Qubit*), %Result* inttoptr (i64 1 to %Result*))
  call void @__quantum__rt__array_record_output(i64 2, i8* null)
  call void @__quantum__rt__result_record_output(%Result* null, i8* null)
  call void @__quantum__rt__result_record_output(%Result* inttoptr (i64 1 to %Result*), i8* null)
  ret void
}
```

```
attributes #0 = {“entry_point” “num_required_qubits”=“2” “num_required_results”=“2” “output_labeling_schema” “qir_proiles”=“base”}
```

Merge: Q-IRIS Design

- Supports two kinds of q-kernels:
  - **QASM Kernel through Qiskit runtime**
  - **QIR Kernel through QIR-EE runtime**
- Tasks can be expressed in python, C and C++.
- Q-IRIS preserves IRIS runtimes task based execution semantics.
- Quantum tasks can be on a Task graph along with classical tasks.
- At runtime, quantum tasks are scheduled through Qiskit and/or QIR-EE.



Q-IRIS Python Wrapper

- Function to utilize Python's shell commands to initiate an execution, storing results in a file.
- Tasks involve calling the function with the ability to send execution parameters.
- Sync is a binary variable determining asychronicity in task submission.

```python
import iris

def qiree_task(iris_params, iris_dev):

    params = iris.iris2py(iris_params)
    dev = iris.iris2py_dev(iris_dev)

    # QIR-EE Inputs
    execpath = params[0]
    filepath = params[1]
    accelerator = params[2]
    numshots = params[3]

    # QIR-EE Execution
    results = os.system(execpath+" "+filepath+" -a "+accelerator+" -s "+numshots+" >
    ↪  output.txt")

    return iris.iris_ok

# Begin parallel tasks here
iris.init()

# Parameters should be a list of strings in the following order:
parameters = ["executable", "LL file", "name of accelerator to use", "number of shots"]

# Running in parallel
n = 16

# Create n tasks (no communication with each other)
tasks = [iris.task() for i in range(n)]

# Feed parameters to each task and submit
# sync tells you how long to wait for completion of the command
for i in range(n):
    tasks[i].pyhost(qiree_task, parameters)
    tasks[i].submit(iris.iris_default, sync=0)

iris.synchronize()
iris.finalize()
```

Q-IRIS Qiskit Kernels

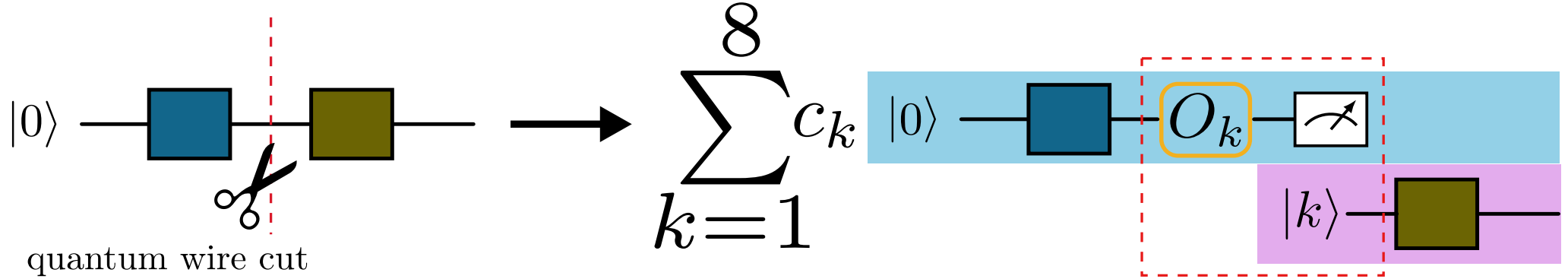
- Loads qiskit libraries upfront.
- A task kernel utilizes wrapper to load an OpenQASM file and call for its execution using qiskit's programming interface.

```
1  import iris
2  import qiskit
3  from qiskit import QuantumCircuit
4  from qiskit_aer import AerSimulator
5
6  def go_quantum(iris_params, iris_dev):
7
8      params = iris.iris2py(iris_params)
9      dev = iris.iris2py_dev(iris_dev)
10
11     circ_qasm = params[0]
12
13     meas = QuantumCircuit(2, 2)
14     meas.measure(range(2), range(2))
15     qc = meas.compose(circ_qasm, range(2), front=True)
16     backend = AerSimulator()
17     job_sim = backend.run(qc, shots=1024)
18     result_sim = job_sim.result()
19     counts = result_sim.get_counts(qc)
20
21     return iris.iris_ok
```

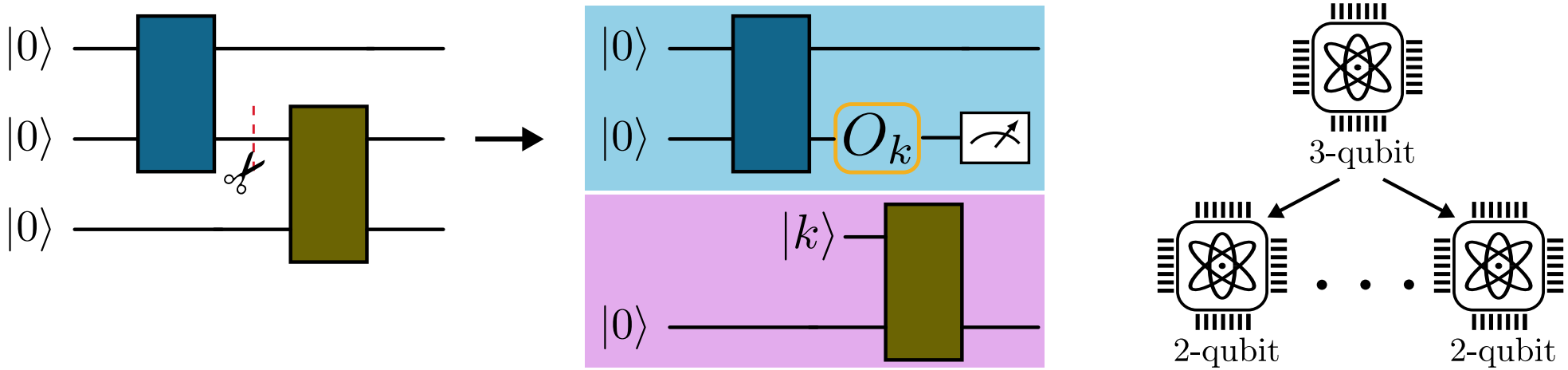
Showcase: Quantum Circuit Decomposition

- Quasi-Probability Decomposition

Split complex quantum circuit into smaller, more manageable parts



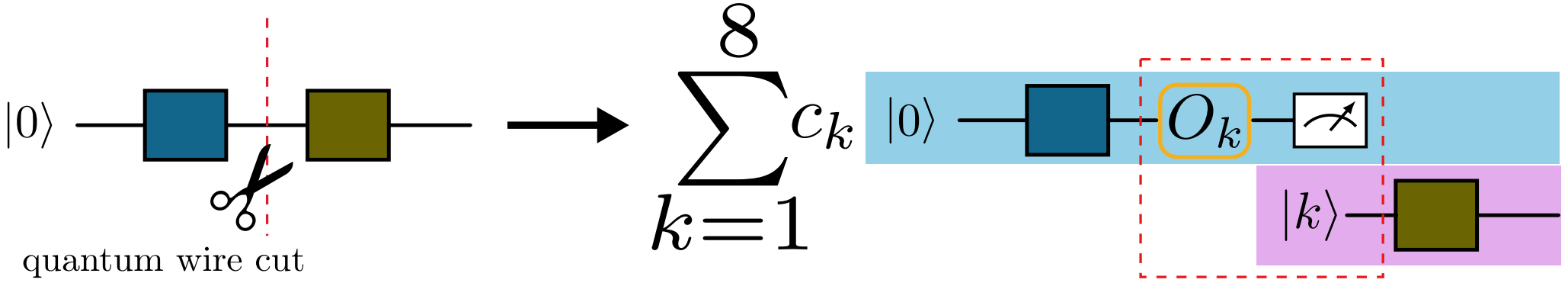
- Every cut creates more quantum instances in smaller QPUs



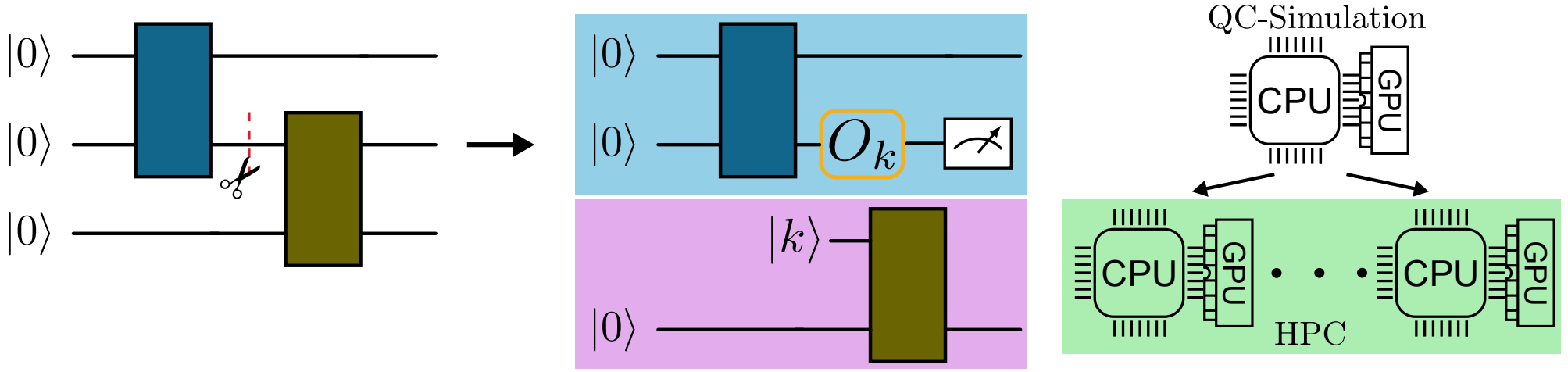
[Harada et al., PRX QUANTUM 5, 040308 (2024)]

Showcase: Quantum Circuit Decomposition and Simulation

- Quasi-Probability Decomposition



- We can simulate quantum instances in a distributed architecture

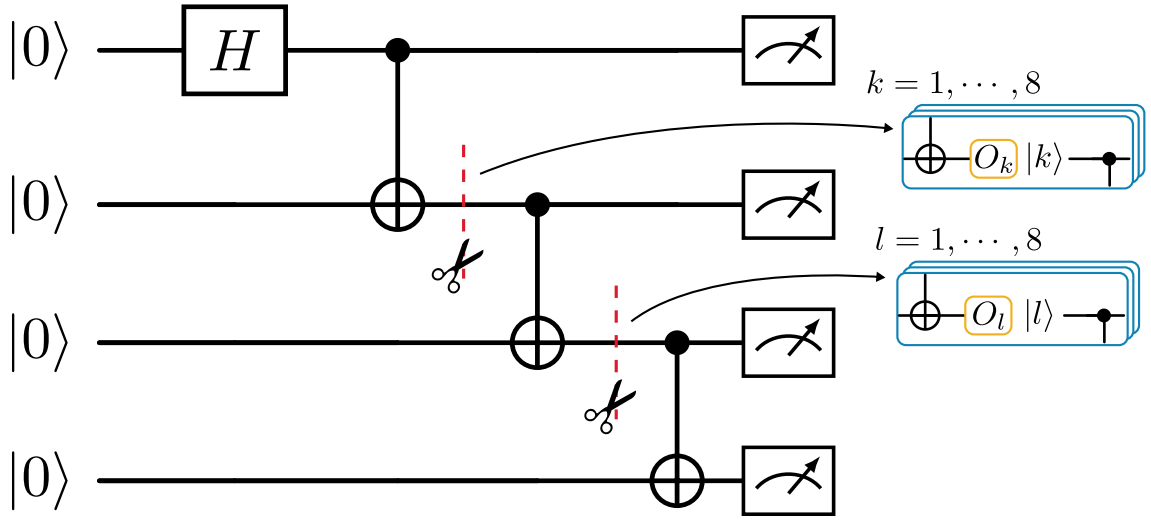


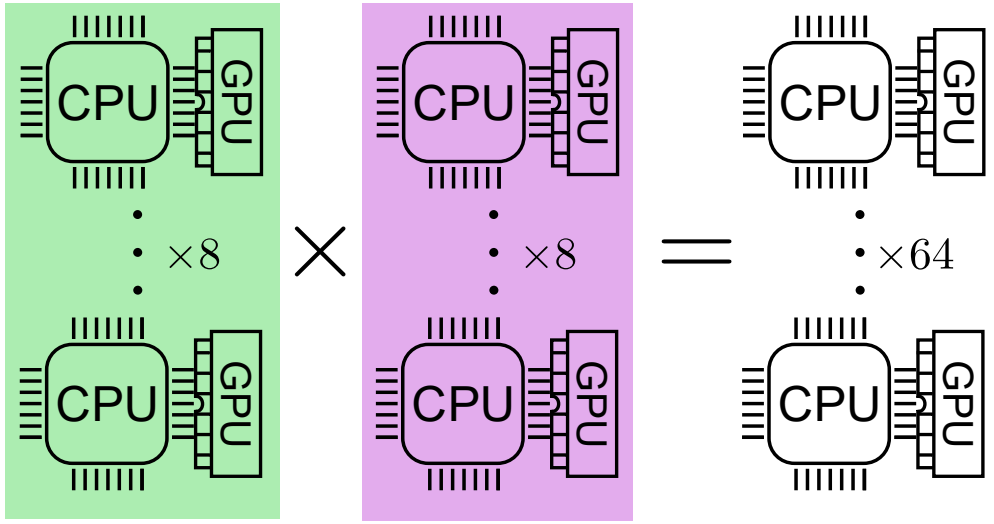
[Harada et al., PRX QUANTUM 5, 040308 (2024)]

Showcase: Experimental Setup

The state of each qubit is directly correlated with the state of the others, even if the qubits are in different places.

[Nielsen & Chuang, Quantum Computation and Quantum Information Book]





GHZ: Greenberger, Horne, and Zeilinger

In simulating the four-qubit experiment, each component experiment (64) will be repeated over thousand of times to gather robust statistical data.

[Harada et al., PRX QUANTUM 5, 040308 (2024)]

4 Qubit GHZ (OpenQASM and also available in QIR)

4 qubit original GHZ

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[4];
creg meas[4];
h q[0];
cx q[0],q[1];
cx q[1],q[2];
cx q[2],q[3];
measure q[0] -> meas[0];
measure q[1] -> meas[1];
measure q[2] -> meas[2];
measure q[3] -> meas[3];
```

One of the 2-qubit cuts (quantum circuit)

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg qpd_meas[1];
creg meas[1];
h q[0];
cx q[0],q[1];
measure q[1] -> qpd_meas[0];
measure q[0] -> meas[0];
```

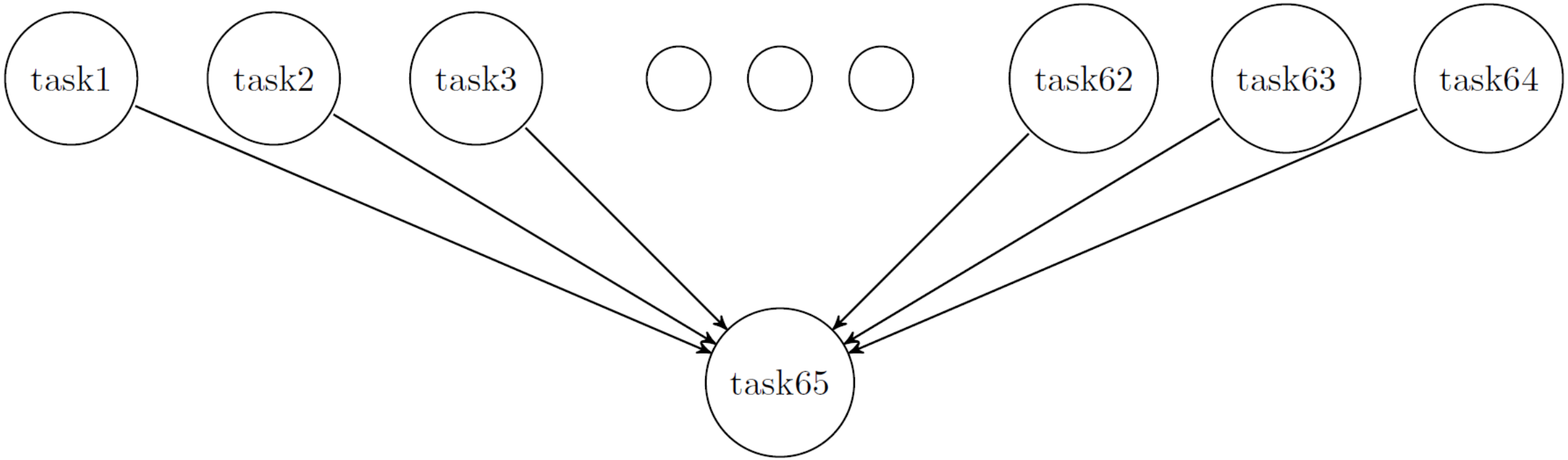
GHZ IRIS-Tasks in Action

```python
iris.init()

home = os.getenv("QIREE")

if not os.path.exists(home):
    home = os.path.join(os.getenv("HOME"), "/qiree")

num_shots = 128
backend = iris.dmem("qsim")

num_tasks = 8
n = num_tasks

# Create n tasks (no communications with each c
tasks = [iris.task() for i in range(n)]
output = [iris.dmem() for i in range(n)]

# Load the QIR content for each task
qir_content = [ iris.dmem_infile(f"subcircuit{i
```

```python
for i in range(n):
    tasks[i] = iris.task("qiree", 1, [], [1], [], [
        (qir_content[i], iris.iris_r), num_shots,
        (output[i], iris.iris_w),
        (backend, iris.iris_r)
    ])
    tasks[i].submit(iris.iris_default, sync=0)

# Classical compute task to do post-processing
result = iris.dmem()
post_process_task = iris.task("qiree_post_processing", 1, [], [1], [],
        [(output[i], iris.iris_r) for i in range(n)] + [ (result, iris.iris_w) ]
)
post_process_task.depends(tasks)
post_process.submit(iris.cpu, sync=0)

iris.synchronize()
print("Result:", result)

iris.finalize()
```

This leverages IRIS's capabilities for heterogeneous computing. Tasks are "in parallel" and asynchronously executed for for future post-processing.

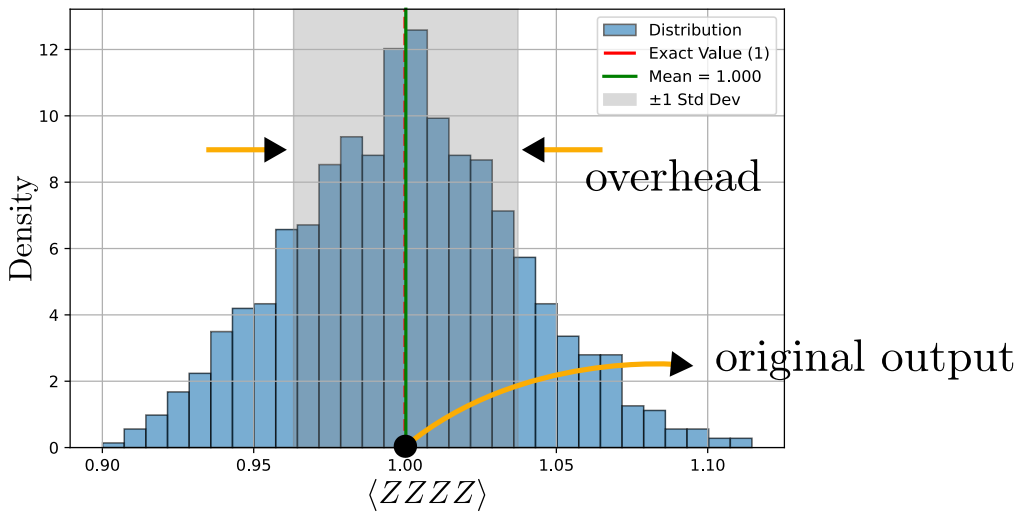
Proudly Parallel

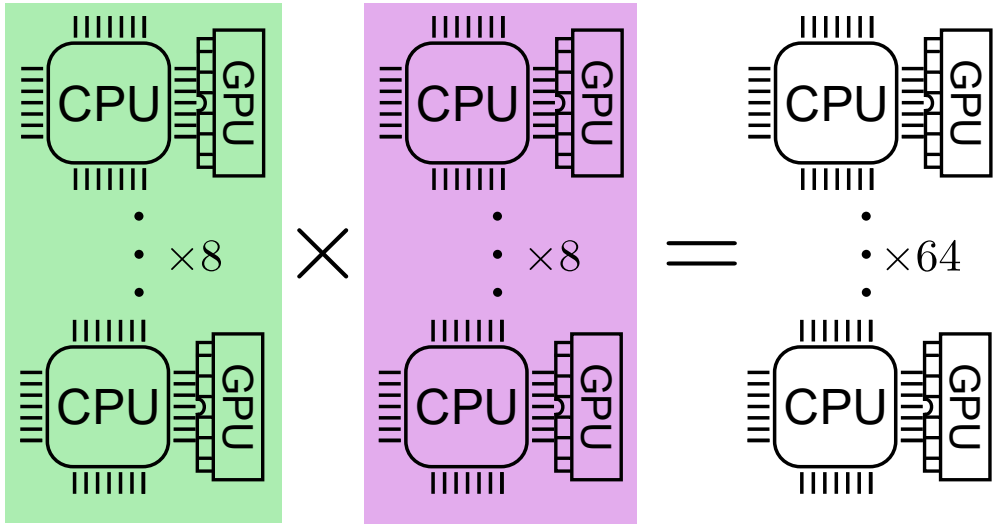


Showcase: Sampling Overhead

The state of each qubit is directly correlated with the state of the others, even if the qubits are in different places.

[Nielsen & Chuang, Quantum Computation and Quantum Information Book]





The overhead decreases with the number of experiments executed; in this instance, we ran 1000 trials at 1024 shots per circuit.

[Harada et al., PRX QUANTUM 5, 040308 (2024)]

Outlook

- Q-IRIS setup will enable us to do parallel and asynchronous heterogeneous quantum computing
- IRIS supports two kinds of quantum kernels
  - **OpenQASM kernels through IRIS-Qiskit**
  - **QIR kernels through IRIS-QIREE**
- Showcase Functionality with Circuit Cutting Example
- Future Work
  - **Explore devices, number of tasks, number of cores, number of measurements.**
  - **For the example, we can optimize number of cuts as well as where to cut.**
  - **How to model quantum channel to enable exchange of exchange information? This will enable better quantum task graphs.**

Acknowledgements

- This work was performed at Oak Ridge National Laboratory, operated by UT-Battelle, LLC under contract DE-AC05-00OR22725 for the US Department of Energy (DOE).
- We acknowledge that support for the work came from DOE's Office of Science, Office of Advanced Scientific Computing Research (ASCR) through the Accelerated Research in Quantum Computing Program MACH-Q project. http://mach-q.org
- This research used resources of the Experimental Computing Laboratory (ExCL) and the Oak Ridge Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory.