PSC

Carnegie Mellon University

University of Pittsburgh

Bridges-2 Webinar

Quantum Approximate Optimization Algorithm (QAOA) Utilizing CUDA-Q on Bridges-2

Dr. Niraj Nepal, Senior Computational Scientist, PSC

March 30, 2026

Pittsburgh Supercomputing Center

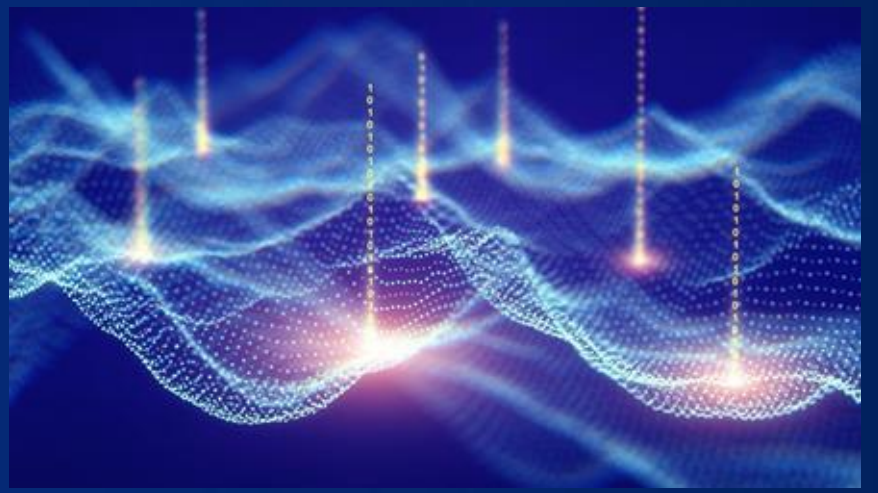
enabling discovery since 1986

The **Pittsburgh Supercomputing Center (PSC)** provides advanced research computing capability, education, and expertise to the national research community.

Since 1986, PSC has provided university, government, and industry researchers with access to some of the most powerful systems available for high-performance computing, enabling discovery across all fields of science.

OUR AREAS OF EXPERTISE

- high-performance and data-intensive computing
- data management technologies
- software architecture, implementation, and optimization
- enabling ground-breaking science, computer science, and engineering
- user support for all phases of research and education
- STEM outreach in data science, bioinformatics, and coding




Welcome!
BRIDGES-2
Bridges-2
Hewlett Packard Enterprise
NSF
Carnegie Mellon University
Hewlett Packard Enterprise is delivering Bridges-2

Bridges-2 Leadership Team

Sergiu Sanielevici
PI & Sr. Scientific Advisor

Robin Scibek
Dir. Comms.
co-PI

Paola Buitrago
Dir. AI & Big Data
co-PI

Edward Hanna
Dir. Adv. Sys. & Ops.
co-PI

Tom Maiden
Dir. Research Computing Support, co-PI

Riaz Khatri
Project Manager

Carnegie Mellon University PSC University of Pittsburgh

Bridges-2 Webinars

- A forum for the Bridges-2 community to learn and share ideas and achievements: Bridges-2 Webinar series | PSC
- Topics and speakers of interest to work that is being done, or that may be done in future.
- Please suggest future speakers (including from your own team) and/or topics (including your own)!

Just email: sergiu@psc.edu

Carnegie Mellon University PSC University of Pittsburgh

Introducing today's presenter

Niraj Nepal received his Ph.D. in Physics from Temple University in 2020, specializing in ab initio electronic structure methods. He held postdoctoral positions at Temple University and Ames National Laboratory before joining the Pittsburgh Supercomputing Center as a Senior Computational Scientist in January 2025. His primary role is to support users and conduct research in ab initio methods, quantum computing, and AI.

Carnegie Mellon University PSC University of Pittsburgh

Q&A Logistics

- **We abide by https://support.access-ci.org/code-of-conduct**
- All of us except our speaker will be muted during their presentation.
- Please type your questions into the Zoom chat.
- After the presentation, our speaker will answer questions live during the final ~10 minutes of this webinar.
- The video recording of this webinar and the slides will be linked from https://www.psc.edu/events/bridges-2-webinar-series/ next week.

Carnegie Mellon University PSC University of Pittsburgh

Outline

- Theoretical Framework
  - Encoding Hamiltonian, Ising Transformation, Adiabatic Theorem, Suzuki-Trotter Expansion, Layered Quantum Circuit, Classical Optimizer
- Max-Cut problem
  - Mathematical Formulation and Python code
- CUDA-Q on Bridges2

Quantum Approximate Optimization Algorithm (QAOA)

Carnegie Mellon University PSC University of Pittsburgh

Introduction

- QAOA is a hybrid quantum–classical algorithm to solve combinatorial optimization problem.
- It uses parameterized quantum circuits combined with classical optimization.
- Useful for near-term noisy intermediate-scale quantum (NISQ) quantum hardware.
- Applied to problems like Max-Cut, scheduling, routing, portfolio optimization, etc.

Flavors of QAOA

Table 1
Summary of ansatz strategies for improving QAOA.

Ansatz	Main Idea	Enhancement & Applications
ma-QAOA [81]	Multi-angle ansatz with a unique parameter for each element of cost and mixer Hamiltonians	Improves approximation ratio for MaxCut while reducing circuit depth
QAOA+ [82]	Augments traditional QAOA with an additional multi-parameter problem-independent layer	Higher approximation ratios for MaxCut on random regular graphs
DC-QAOA [83,84]	Adds a problem-dependent counterdiabatic driving term to the QAOA ansatz	Improves the convergence rate of the approximation ratio while reducing circuit depth
ab-QAOA [85]	Incorporates local fields into the operators to reduce computation time	Computation time reduction for combinatorial optimization
ADAPT-QAOA [86]	Iterative version of QAOA with systematic selection of mixers based on gradient criterion	Can be problem-specific and addresses hardware constraints
Recursive QAOA [87]	Non-local variant of QAOA that iteratively reduces problem size by eliminating qubits	Overcomes locality constraints and achieves better performance
QAOAnsatz [88]	Extends the original formulation with broader families of operators and allows for encoding of constraints	Adaptable to a wider range of optimization problems with hard and soft constraints
GM-QAOA [89]	Uses Grover-like selective phase shift mixing operators	Solves k-Vertex Cover, Traveling Salesperson Problem, Discrete Portfolio Rebalancing
Th-QAOA [90]	Replaces standard phase separator with a threshold function	Solves MaxCut, Max k-Vertex Cover, Max Bisection
Constraint Preserving Mixers [91]	Constructs mixers that enforce hard constraints	Solves optimization problems with hard constraints
WS-QAOA [92]	Modifies the initial state and mixer Hamiltonian based on the optimal solution to the relaxed QUBO problem	Solutions guaranteed to retain the GW bound for the MaxCut problem
FALQON [70]	Uses qubit measurements for feedback-based quantum optimization, avoiding classical optimizers	Produces monotonically improving approximate solutions as circuit depth grows while bypassing classical optimization loops
FALQON+ [93]	Combines FALQON's initialization with QAOA for better parameter initialization	Improves initialization of standard QAOA for non-isomorphic graphs with 8 to 14 vertices
FQAOA [94]	Utilizes fermion particle number preservation to intrinsically impose constraints in QAOA process	Improves performance in portfolio optimization, applicable to Grover adaptive search and quantum phase estimation
Quantum Dropout [95]	Selectively drops out clauses defining the quantum circuit while keeping the cost function intact	Improves QAOA performance on hard cases of combinatorial optimization problems
ST-QAOA [96]	Uses an approximate classical solution to construct a problem instance-specific circuit	Achieves same performance guarantee as the classical algorithm, outperforms QAOA at low depths for MaxCut problem
Modified QAOA [31]	Modifies cost Hamiltonian with conditional rotations	Improves approximation ratio for MaxCut at $p = 1$

ELSEVIER

Physics Reports

Volume 1068, 2 June 2024, Pages 1-66

A review on Quantum Approximate Optimization Algorithm and its variants

Kostas Blekos [a 1], Dean Brand [b 1], Andrea Ceschini [c 1], Chiao-Hui Chou [d 1], Rui-Hao Li [e 1], Komal Pandya [f 1], Alessandro Summer [g 1]

https://doi.org/10.1016/j.physrep.2024.03.002

Original (Standard) QAOA

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

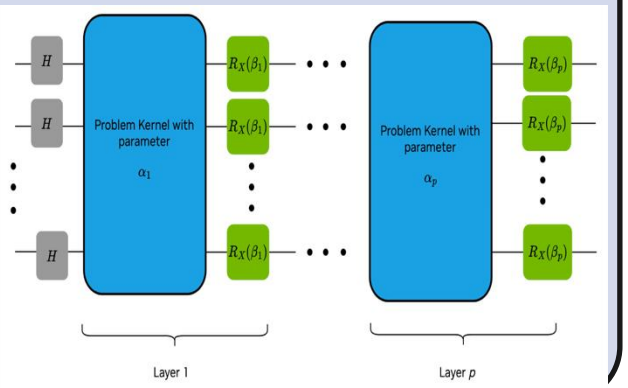
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Combinatorial Optimization Problems

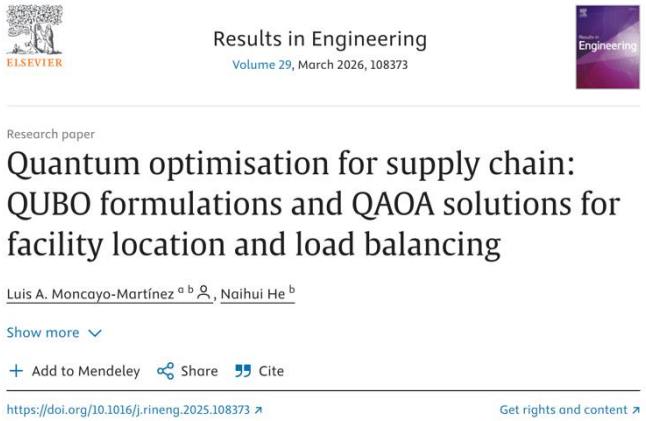
Article | Open access | Published: 06 November 2024

Scaling whole-chip QAOA for higher-order ising spin glass models on heavy-hex graphs

Elijah Pelofske, Andreas Bärtschi, Lukasz Cincio, John Golden & Stephan Eidenbenz

npj Quantum Information **10**, Article number: 109 (2024) | Cite this article

7889 Accesses | **22** Citations | **5** Altmetric | Metrics



Article | Open access | Published: 13 August 2012

Finding low-energy conformations of lattice protein models by quantum annealing

Alejandro Perdomo-Ortiz, Neil Dickson, Marshall Drew-Brook, Geordie Rose & Alán Aspuru-Guzik

Scientific Reports **2**, Article number: 571 (2012) | Cite this article

24k Accesses | **356** Citations | **85** Altmetric | Metrics

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$

$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

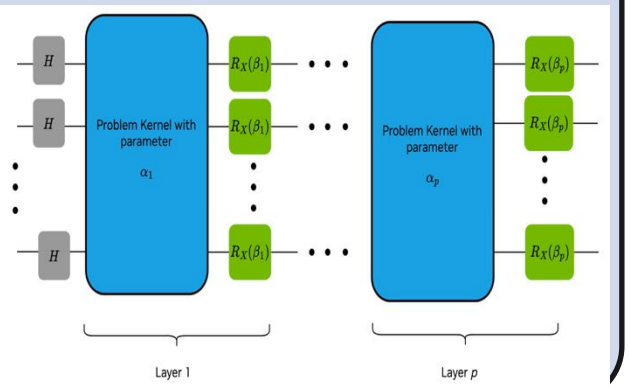
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

QUBO Formalism

Quadratic Unconstrained Binary Optimization

- Many combinatorial problems can be converted to QUBO and solve with quantum computing

- Quadratic objective function
- No constraint
- Target vector should be binary, containing 0 and 1.

- Transform constraints into penalty term (λ) in the cost function.

$$\mathcal{L}(x, \lambda) = \mu^T x - x^T \Sigma x - \lambda g(x)^p,$$

$g(x) \leq 0$ is constraint, where p is integer

Carnegie Mellon University PSC University of Pittsburgh

QUBO Formalism

Quadratic Unconstrained Binary Optimization

- Many combinatorial problems can be converted to QUBO and solve with quantum computing

- Quadratic objective function
- No constraint
- Target vector should be binary, containing 0 and 1.

- Transform constraints into penalty term (λ) in the cost function.

$$\mathcal{L}(x, \lambda) = \mu^T x - x^T \Sigma x - \lambda g(x)^p,$$

$g(x) \leq 0$ is constraint, where p is integer

Carnegie Mellon University PSC University of Pittsburgh

QUBO Formalism

Quadratic Unconstrained Binary Optimization

- Many combinatorial problems can be converted to QUBO and solve with quantum computing

- Quadratic objective function
- No constraint
- Target solution should be binary vector, containing 0 and 1.

- If problem has constraints, transform it into penalty term (λ) and add to the cost function.

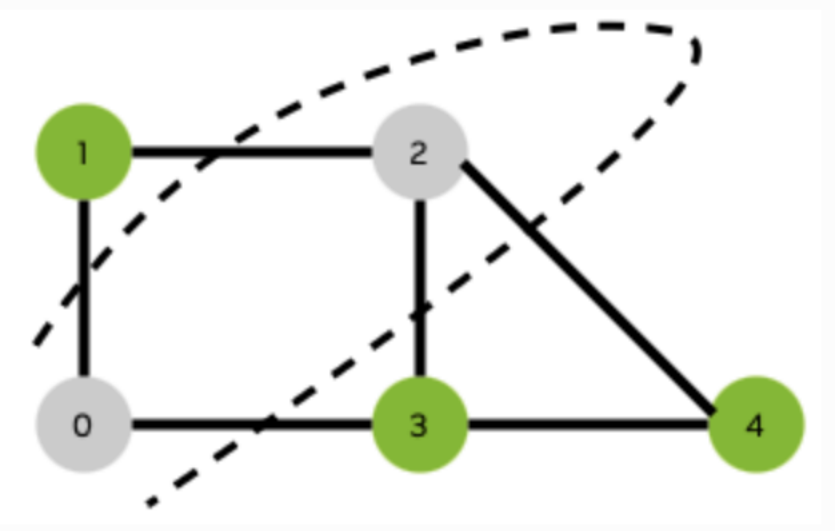
  For example: Budget constraint in Portfolio optimization

$$\mathbf{C}(x, \lambda) = x^T Q x + b^T x + \lambda(Ax - d)^2,$$

$Ax \leq d$ is constraint

Carnegie Mellon University PSC University of Pittsburgh

Max-Cut Problem



$$C(x) = \sum_{i,j=1}^{v} w_{ij}x_i(1-x_j) = \sum_{i,j=1}^{v} x_i Q_{ij} x_j + \sum_{i=1}^{v} b_i x_i$$

$$Q_{ij} = -w_{ij}, b_i = \sum_j^v w_{ij}, \mathbf{i = 1, 2, \ldots |v|}$$

$$\mathbf{C(x) = x^T Q x + b^T x}$$ **(Matrix form)**

Graph G = $(v, \mathcal{E})$ with v is node (vertex) and $\mathcal{E}$ is edge.

Partition vertices $\{x_i\}$ into 2 complementary subsets labeled by 0 and 1

Maximizing the number of edges connecting nodes of different types

Maximize C(x) or Minimize –C(x)

Carnegie Mellon University PSC University of Pittsburgh

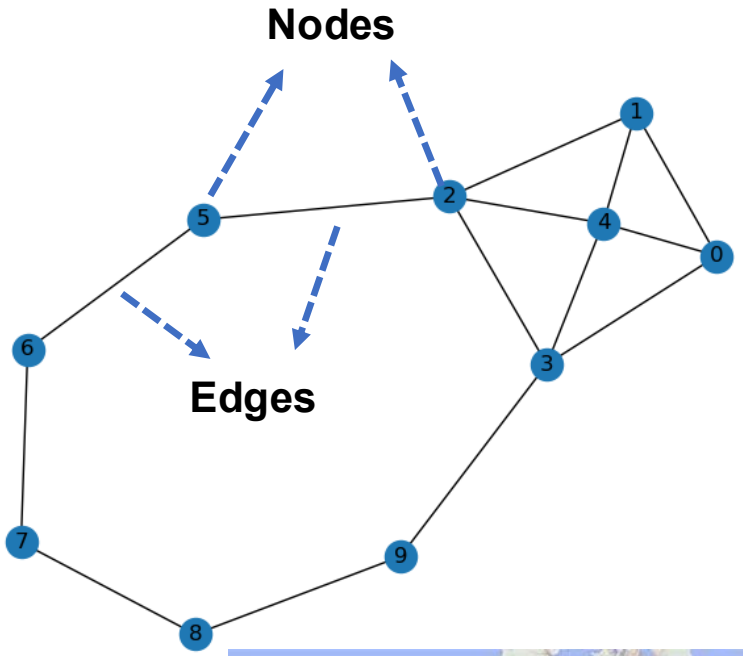
2. Define Graph

In [13]:

```
nodes = list(range(10))

edges = [
    [0,1],[1,2],[2,3],[3,0],
    [4,0],[4,1],[4,2],[4,3],
    [5,6],[6,7],[7,8],[8,9],
    [2,5],[3,9]
]

edges_src = [e[0] for e in edges]
edges_tgt = [e[1] for e in edges]
```

(10 nodes)

(14 edges)



Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\;\hat{H}_c$$

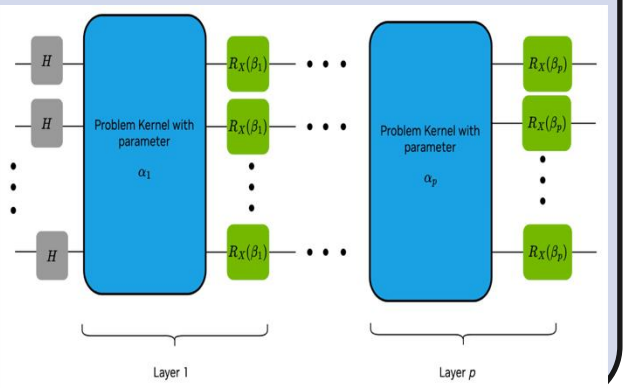
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Cost Hamiltonian

Encoding cost Hamiltonian

- $\hat{H}_c|x\rangle = C(x)\,|x\rangle$
- $C(x) = \sum_{i,j=1}^{v} w_{ij} x_i (1 - x_j)$
- **$|x\rangle$ is the quantum state encoding bit string**

  $x = x_1 x_2 x_3 \ldots x_n,$
  $x_i \in \{0,1\}$
- $C(x) = \langle x|\hat{H}_c|x\rangle$

Ising Transformation

- **Pauli spin-operator $\hat{Z}_i$,**

  $\hat{Z}_i|x_i\rangle = (-1)^{x_i}|x_i\rangle = z_i|x_i\rangle$
- $(-1)^{x_i} = 1 - 2x_i, x_i \in \{0,1\}$
- $1 - 2x_i = z_i \Longrightarrow x_i = \frac{1-z_i}{2}$
- **Substituting x_i in C(x), and changing z_i to Z_i**

$$\hat{H}_c = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} w_{ij}(\mathbb{I} - Z_i Z_j)$$

Goal

- **We need to find highest-energy state of cost Hamiltonian**

OR

- **Lowest-energy state of**

$-\hat{H}_c = \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} w_{ij}(Z_i Z_j - \mathbb{I})$

Carnegie Mellon University PSC University of Pittsburgh

Cost Hamiltonian

Encoding cost Hamiltonian

- $\hat{H}_c|x\rangle = C(x)\,|x\rangle$
- $C(x) = \sum_{i,j=1}^{v} w_{ij} x_i (1 - x_j)$
- **$|x\rangle$ is the quantum state encoding bit string**

  $x = x_1 x_2 x_3 \ldots x_n,$

  $x_i \in \{0,1\}$
- $C(x) = \langle x|\hat{H}_c|x\rangle$

Ising Transformation

- **Pauli spin-operator $\hat{Z}_i$,**

  $$\hat{Z}_i|x_i\rangle = (-1)^{x_i}|x_i\rangle = z_i|x_i\rangle$$
- $(-1)^{x_i} = 1 - 2x_i, x_i \in \{0,1\}$
- $1 - 2x_i = z_i \Longrightarrow x_i = \frac{1 - z_i}{2}$
- **Substituting x_i in C(x), and changing z_i *to* Z_i**

$$\hat{H}_c = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} w_{ij}(\mathbb{I} - Z_i Z_j)$$

Goal

- **We need to find highest-energy state of cost Hamiltonian**

  OR
- **Lowest-energy state of**

  $-\hat{H}_c = \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} w_{ij}(Z_i Z_j - \mathbb{I})$

Carnegie Mellon University PSC University of Pittsburgh

Cost Hamiltonian

Encoding cost Hamiltonian

- $\hat{H}_c|x\rangle = C(x)\,|x\rangle$
- $C(x) = \sum_{i,j=1}^{v} w_{ij} x_i (1 - x_j)$
- **$|x\rangle$ is the quantum state encoding bit string**

  $x = x_1 x_2 x_3 \ldots x_n,$

  $x_i \in \{0,1\}$
- $C(x) = \langle x|\hat{H}_c|x\rangle$

Ising Transformation

- **Pauli spin-operator $\hat{Z}_i$,**

  $\hat{Z}_i|x_i\rangle = (-1)^{x_i}|x_i\rangle = z_i|x_i\rangle$
- $(-1)^{x_i} = 1 - 2x_i, x_i \in \{0,1\}$
- $1 - 2x_i = z_i \Longrightarrow x_i = \frac{1 - z_i}{2}$
- **Substituting x_i in C(x), and changing z_i *to* Z_i**

$$\hat{H}_c = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} w_{ij}\left(\mathbb{I} - Z_i Z_j\right)$$

Goal

- **We need to find highest-energy state of cost Hamiltonian**

  OR
- **Lowest-energy state of**

  $-\hat{H}_c =$

$$\frac{1}{2} \sum_{(i,j)\in\mathcal{E}} w_{ij}\left(Z_i Z_j - \mathbb{I}\right)$$

Carnegie Mellon University PSC University of Pittsburgh

$$\widehat{H}_C = \frac{1}{2} \sum_{(u,v)\in\mathcal{E}} (Z_u Z_v - \mathbb{I}),\ w_{uv} = 1\ , (u, v) \in \epsilon$$

3. Cost Function & Ising Hamiltonian

$$Z_u \rightarrow \mathbf{spin.z(u)}$$
$$\mathbb{I} \rightarrow \mathbf{spin.i(u)}$$

```
from cudaq import spin
```

In [15]:

```
def hamiltonian_max_cut(edges_src, edges_tgt):
    H = 0
    for u,v in zip(edges_src, edges_tgt):
        H += 0.5 * (spin.z(u) * spin.z(v) - spin.i(u) * spin.i(v))
    return H

hamiltonian = hamiltonian_max_cut(edges_src, edges_tgt)
hamiltonian
```

Out[15]:

```
<cudaq.mlir._mlir_libs._quakeDialects.cudaq_runtime.SpinOperator at 0x7f02dcd626f0>
```

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

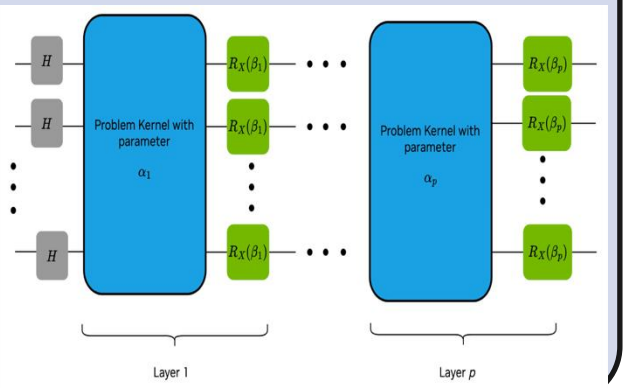
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Why Adiabatic Theorem?

- **Solving the Schrödinger equation for a complex $\hat{H}_C$ is computationally intractable.**

$$\hat{H}_c|x\rangle = C(x)\,|x\rangle$$

- One can start from a simple $\hat{H}_M$ and slowly evolve into complex $\hat{H}_C$ over the time.

- The system adiabatically follows from the initial ground-state of $\hat{H}_M$ to that of $\hat{H}_C$ without jumping.

Challenges

- **Lack of robust state preparation**
- **Searching solution through a Hilbert space of dimension 2^N**
- **System getting trapped in a local minima**
- **Requires infinite precision and infinite time. Not possible in NISQ-era quantum hardware.**

$$|x\rangle \sim 2^N, \hat{H}_c \sim 2^N \times 2^N$$

Carnegie Mellon University PSC University of Pittsburgh

Why Adiabatic Theorem?

- **Solving the Schrödinger equation for a complex $\hat{H}_C$ is computationally intractable.**

$$\hat{H}_c|x\rangle = C(x)\,|x\rangle$$

- **One can start from a simple $\hat{H}_M$ and slowly evolve into complex $\hat{H}_C$ over the time.**

- **The system adiabatically follows from the initial ground-state of $\hat{H}_M$ to that of $\hat{H}_C$ without jumping.**

Why Adiabatic Theorem?

- Solving the Schrödinger equation for a complex $\hat{H}_C$ is computationally intractable.

$$\hat{H}_c|x\rangle = C(x)\,|x\rangle$$

- One can start from a simple $\hat{H}_M$ and slowly evolve into complex $\hat{H}_C$ over the time.

- The system adiabatically follows from the initial ground-state (GS) of $\hat{H}_M$ to that of $\hat{H}_C$ without jumping.

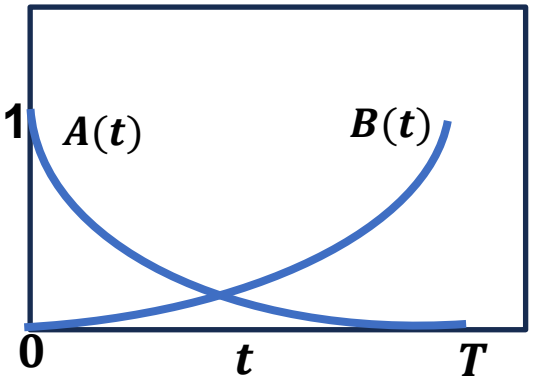
Destination is unknown !!!

Carnegie Mellon University PSC University of Pittsburgh

Adiabatic Theorem

$$\hat{H}(t) = A(t)\hat{H}_M + B(t)\hat{H}_C = \left(1 - \frac{t}{T}\right)\hat{H}_M + \frac{t}{T}\hat{H}_C$$
$$= \hat{H}_M(t) + \hat{H}_c(t)$$

$0 \le t \le T$



Mixer Hamiltonian

$$\hat{H}_M = -\sum_{j \in v} X_j$$

- **Ground-state (GS) $|+\rangle^{\otimes v}$ is easy to prepare.**
- $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \hat{H}|0\rangle$
- Applying Hadamard gate to all qubit for each vertex.

Cost Hamiltonian

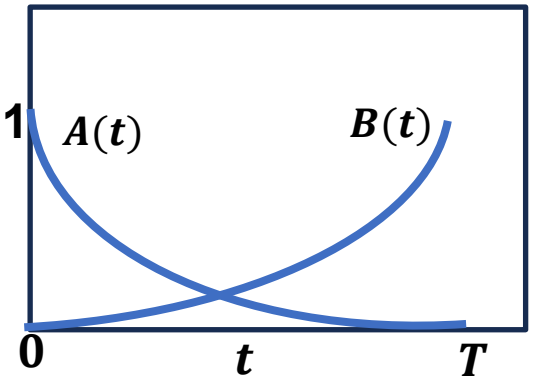
$$\hat{H}_C = \sum_{(i,j) \in \mathcal{E}} w_{ij}\left(Z_i Z_j - \mathbb{I}\right)$$

Carnegie Mellon University PSC University of Pittsburgh

Adiabatic Theorem

$$\hat{H}(t) = A(t)\hat{H}_M + B(t)\hat{H}_C = \left(1 - \frac{t}{T}\right)\hat{H}_M + \frac{t}{T}\hat{H}_C$$
$$= \hat{H}_M(t) + \hat{H}_c(t)$$

$0 \le t \le T$



Mixer Hamiltonian

$$\hat{H}_M = -\sum_{j \in v} X_j$$

- **Ground-state (GS) $|+\rangle^{\otimes v}$ is easy to prepare.**
- $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \hat{H}|0\rangle$
- Applying Hadamard gate to all qubit for each vertex.

Slowly evolve in time

- **Faster time-evolution can lead system to move to excited state**
- **Time scale depends on minimum energy gap $(E_1 - E_0)$**

Cost Hamiltonian

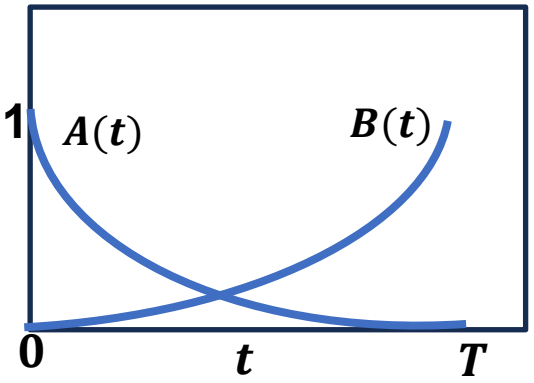
$$\hat{H}_C = \sum_{(i,j) \in \mathcal{E}} w_{ij}(Z_i Z_j - \mathbb{I})$$

Carnegie Mellon University PSC University of Pittsburgh

Adiabatic Theorem

$$\hat{H}(t) = A(t)\hat{H}_M + B(t)\hat{H}_C = \left(1 - \frac{t}{T}\right)\hat{H}_M + \frac{t}{T}\hat{H}_C$$
$$= \hat{H}_M(t) + \hat{H}_c(t)$$

$0 \le t \le T$



Mixer Hamiltonian

$$\hat{H}_M = -\sum_{j \in v} X_j$$

- **Ground-state (GS) $|+\rangle^{\otimes v}$ is easy to prepare.**
- $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \hat{H}|0\rangle$
- Applying Hadamard gate to all qubit for each vertex.

Slowly evolve in time

GS of $\hat{H}_M$ evolves into GS of $\hat{H}_C$

Cost Hamiltonian

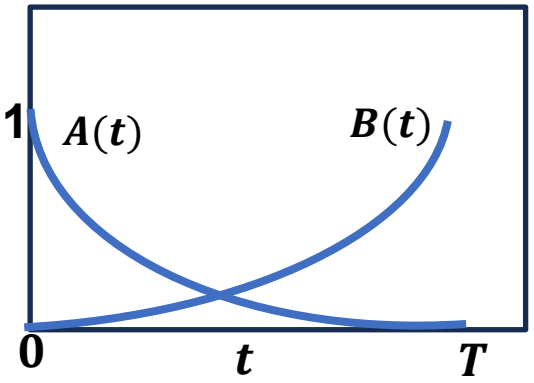
$$\hat{H}_C = \sum_{(i,j) \in \mathcal{E}} w_{ij}(Z_i Z_j - \mathbb{I})$$

Carnegie Mellon University PSC University of Pittsburgh

Adiabatic Theorem

$$\hat{H}(t) = A(t)\hat{H}_M + B(t)\hat{H}_C = \left(1 - \frac{t}{T}\right)\hat{H}_M + \frac{t}{T}\hat{H}_C$$
$$= \hat{H}_M(t) + \hat{H}_c(t)$$

$0 \le t \le T$



Mixer Hamiltonian

$$\hat{H}_M = -\sum_{j \in v} X_j$$

- **Ground-state (GS) $|+\rangle^{\otimes v}$ is easy to prepare.**
- $|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \hat{H}|0\rangle$
- Applying Hadamard gate to all qubit for each vertex.

Slowly evolve in time

GS of $\hat{H}_M$ evolves into GS of $\hat{H}_C$

Final $\hat{H}(t)$ becomes time-dependent.

Cost Hamiltonian

$$\hat{H}_C = \sum_{(i,j) \in \mathcal{E}} w_{ij}(Z_i Z_j - \mathbb{I})$$

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp(-i\int_0^t dt'\ \hat{H}(t'))$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp(-i\int_0^t dt'\ \hat{H}(t'))$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp(-i\int_0^t dt'\ \hat{H}(t'))$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

The time-ordering operator $\mathcal{T}$ is needed to keep the evolution in the correct order, when a time-dependent Hamiltonian doesn't commute at different times.

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp(-i\int_0^t dt'\ \hat{H}(t'))$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

The time-ordering operator $\mathcal{T}$ is needed to keep the evolution in the correct order, when a time-dependent Hamiltonian doesn't commute at different times.

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp(-i\int_0^t dt'\ \hat{H}(t'))$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

The time-ordering operator $\mathcal{T}$ is needed to keep the evolution in the correct order, when a time-dependent Hamiltonian doesn't commute at different times.

Carnegie Mellon University PSC University of Pittsburgh

Evolution of State

$$\hat{H}(t)|\psi(t)\rangle = i\,\frac{\partial|\psi(t)\rangle}{\partial t}$$

$$i = \sqrt{-1}$$

$$|\psi(t)\rangle = \hat{U}_t|\psi(0)\rangle$$

Time-dependent $\hat{H}(t)$

$$\hat{U}_t = \mathcal{T}\exp\left(-i\int_0^t dt'\ \hat{H}(t')\right)$$

Discretization

$$\int_0^t dt'\ \hat{H}(t') \approx \sum_{j=1}^{P} \Delta t\ \hat{H}(j\Delta t)$$

$$\Delta t = \frac{t}{P}$$

P-layers of product of exponentials

$$\hat{U}_t = \mathcal{T}\prod_{j=1}^{P} e^{-i\,\Delta t\,[\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t)]}$$

For non-commuting operators, exponential operator with multiple terms in the sum can't be applied sequentially. Break down is required !

The time-ordering operator $\mathcal{T}$ is needed to keep the evolution in the correct order, when a time-dependent Hamiltonian doesn't commute at different times.

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

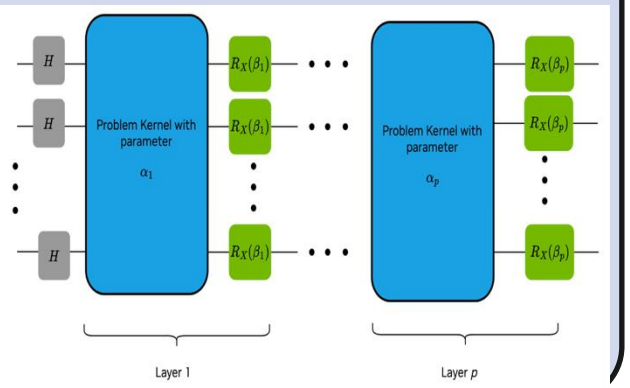
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Suzuki-Trotter expansion

Suzuki-Trotter expansion (1st order)

For non-commuting operators $\hat{H}_M(t)$ and $\hat{H}_C(t)$

$$e^{-i\Delta t\,(\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t))} = e^{-i\Delta t\hat{H}_M(j\Delta t)}.\,e^{-i\Delta t\,\hat{H}_C(j\Delta t)} + O(\Delta t^2)$$

P-layers of product of exponentials

$$|\psi(t)\rangle \approx \left[e^{-i\Delta t\,\hat{H}_M(P\Delta t)}e^{-i\Delta t\,\hat{H}_C(P\Delta t)}\right](\mathrm{p^{th}\ term}) \ldots\ldots \left[e^{-i\Delta t\,\hat{H}_M(\Delta t)}e^{-i\Delta t\,\hat{H}_C(\Delta t)}\right](1^{\mathrm{st}}\ \mathrm{term})|\psi\rangle$$

Non-commuting

$$\hat{A}\,\hat{B}|\psi\rangle \neq \hat{B}\,\hat{A}|\psi\rangle$$

$$XZ|0\rangle = X(Z|0\rangle) = X|0\rangle = +1|1\rangle$$
$$ZX|0\rangle = Z(X|0\rangle) = Z|1\rangle = -1|1\rangle$$
$$XZ|0\rangle \neq ZX|0\rangle$$

$X|0\rangle = |1\rangle, X|1\rangle = |0\rangle$ (Bit-flip (NOT) gate)
$Z|0\rangle = |0\rangle, Z|1\rangle = -1|1\rangle$ (Phase-flip gate)

https://jmsh.springeropen.com/articles/10.1186/s41313-021-00032-6

Carnegie Mellon University PSC University of Pittsburgh

Suzuki-Trotter expansion

Suzuki-Trotter expansion (1st order)

For non-commuting operators $\hat{H}_M(t)$ and $\hat{H}_C(t)$

$$e^{-i\Delta t\,(\hat{H}_M(j\Delta t)+\hat{H}_C(j\Delta t))} = e^{-i\Delta t\hat{H}_M(j\Delta t)}.\,e^{-i\Delta t\,\hat{H}_C(j\Delta t)} + O(\Delta t^2)$$

P-layers of product of exponentials

$$|\psi(t)\rangle \approx \left[e^{-i\Delta t\,\hat{H}_M(P\Delta t)}e^{-i\Delta t\,\hat{H}_C(P\Delta t)}\right](P^{th}\ term)$$

$$\ldots\ldots\left[e^{-i\Delta t\,\hat{H}_M(\Delta t)}e^{-i\Delta t\,\hat{H}_C(\Delta t)}\right](1^{st}\ term)|\psi\rangle$$

Non-commuting

$$\hat{A}\,\hat{B}|\psi\rangle \neq \hat{B}\,\hat{A}|\psi\rangle$$

$$XZ|0\rangle = X(Z|0\rangle) = X|0\rangle = +1|1\rangle$$
$$ZX|0\rangle = Z(X|0\rangle) = Z|1\rangle = -1|1\rangle$$
$$XZ|0\rangle \neq ZX|0\rangle$$

$X|0\rangle = |1\rangle, X|1\rangle = |0\rangle$ (Bit-flip (NOT) gate)
$Z|0\rangle = |0\rangle, Z|1\rangle = -1|1\rangle$ (Phase-flip gate)

https://jmsh.springeropen.com/articles/10.1186/s41313-021-00032-6

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

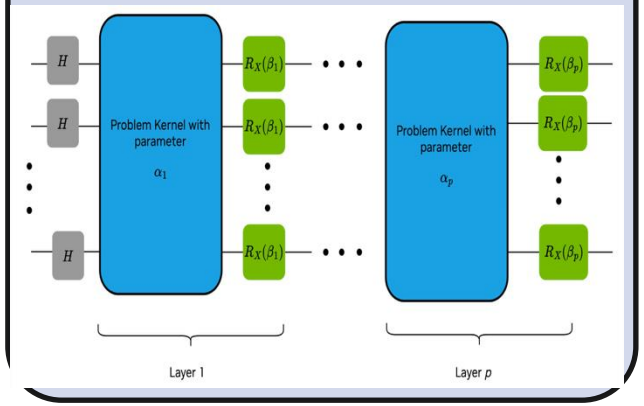
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

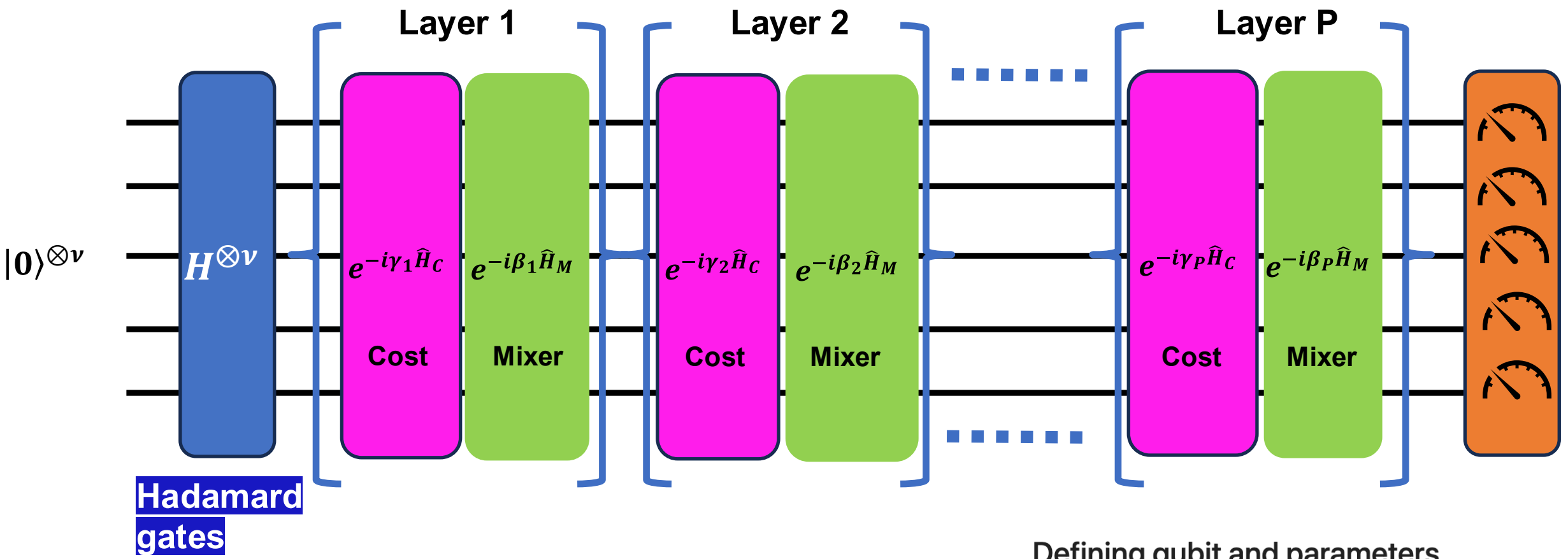
- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Ansatz



Here, we have parametrized Δt with β and γ.

Defining qubit and parameters

```
In [19]: qubit_count = len(nodes)
         layer_count = 5
         parameter_count = 2 * layer_count
```

Carnegie Mellon University PSC University of Pittsburgh

Decomposing each layer of Quantum Circuit

Cost (1 edge)

- Edge (i,j): e_{ij}
- $e^{-i\gamma_{e_{ij}}(\hat{H}_C)_{ij}}$
- $\text{const} \times e^{-i\frac{\gamma_{e_{ij}}}{2} Z_i Z_j}$
- $R_{Z_i Z_j}(\gamma_{e_{ij}})$

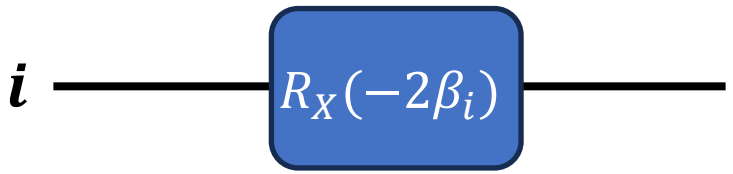
Mixer (1 node)

- vertex v_i
- $e^{-i\beta_i(\hat{H}_M)_i}$
- $e^{-i(-\beta_i)X_i}$
- $R_{X_i}(-2\beta_i)$

$$(\hat{H}_C)_{ij} = \frac{1}{2}(Z_i Z_j - \mathbb{I})$$

$$(\hat{H}_M)_i = -X_i$$

Mixer term of a vertex



1-qubit X-rotation gate

A ZZ rotation gate is **native** on some quantum hardware [IONQ (Forte)] but must be **decomposed to available native gates** on others.

Carnegie Mellon University PSC University of Pittsburgh

2-qubit ZZ-rotation gate

- $R_{Z_iZ_j}(\theta) = e^{-i\frac{\theta}{2}Z_iZ_j}$
- $Z_iZ_j|ij\rangle = \lambda|ij\rangle$

- $R_{Z_iZ_j}(\theta)|ij\rangle = e^{-\frac{i\theta\lambda}{2}}|ij\rangle,$

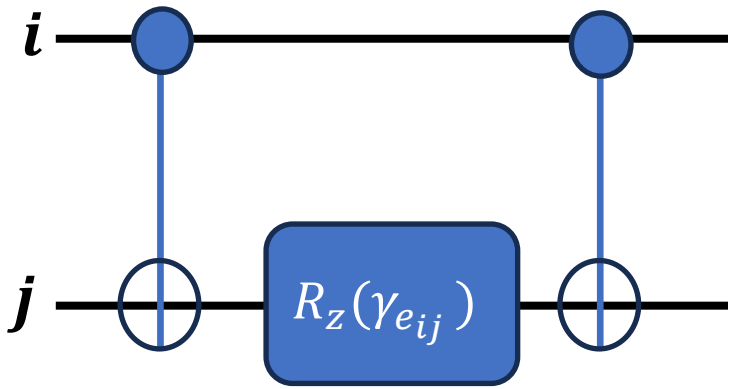
$$\lambda = \begin{cases} 1, i = j \\ -1, i \neq j \end{cases}$$

- $R_Z(\theta) = e^{-i\frac{\theta}{2}Z}$
- $Z|j\rangle = (-1)^j\,|j\rangle$

- $R_Z(\theta)|j\rangle = e^{-i\frac{\theta}{2}(-1)^j}\,|j\rangle$

- $CNOT(i,j)R_{Z_j}(\theta)\,CNOT(i,j)|ij\rangle = e^{-\frac{i\theta\lambda}{2}}|ij\rangle$

$\|ij\rangle$	$R_{Z_iZ_j}(\theta)$ $\|ij\rangle$	CNOT(I,j) $\|ij\rangle$	$R_{Z_j}(\theta)$ CNOT(I,j) $\|ij\rangle$	$CNOT(i,j)R_{Z_j}(\theta)$ $CNOT(i,j)\|ij\rangle$
$\|00\rangle$	$e^{-i\theta/2}\|00\rangle$	$\|00\rangle$	$e^{-i\theta/2}\|00\rangle$	$e^{-i\theta/2}\|00\rangle$
$\|01\rangle$	$e^{+i\theta/2}\|01\rangle$	$\|01\rangle$	$e^{+i\theta/2}\|01\rangle$	$e^{+i\theta/2}\|01\rangle$
$\|10\rangle$	$e^{+i\theta/2}\|10\rangle$	$\|11\rangle$	$e^{+i\theta/2}\|11\rangle$	$e^{+i\theta/2}\|10\rangle$
$\|11\rangle$	$e^{-i\theta/2}\|11\rangle$	$\|10\rangle$	$e^{-i\theta/2}\|10\rangle$	$e^{-i\theta/2}\|11\rangle$



$$CNOT(i,j)R_{Z_j}\left(\gamma_{e_{ij}}\right)CNOT(i,j)$$

Carnegie Mellon University PSC University of Pittsburgh

Cost Hamiltonian for a single edge

In [16]:

```
@cudaq.kernel
def qaoa_problem(qubit_0: cudaq.qubit, qubit_1: cudaq.qubit, gamma: float):
    x.ctrl(qubit_0, qubit_1)  CNOT
    rz(2.0 * gamma, qubit_1)  Rz rotation
    x.ctrl(qubit_0, qubit_1)  CNOT
```

5. Full QAOA Circuit

In [17]:

```
@cudaq.kernel
def qaoa_kernel(qubit_count:int, layer_count:int, edges_src:List[int], edges_tgt:List[int], thetas:List[float]):
    qreg = cudaq.qvector(qubit_count)
    h(qreg)

    for i in range(layer_count):
        # Cost unitary
        for e in range(len(edges_src)):
            qaoa_problem(qreg[edges_src[e]], qreg[edges_tgt[e]], thetas[i])
        # Mixer
        for q in range(qubit_count):
            rx(2.0 * thetas[i + layer_count], qreg[q])
```

Preparing Ground-state of Mixer Hamiltonian

Cost layer

Mixer layer

Carnegie Mellon University PSC University of Pittsburgh

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

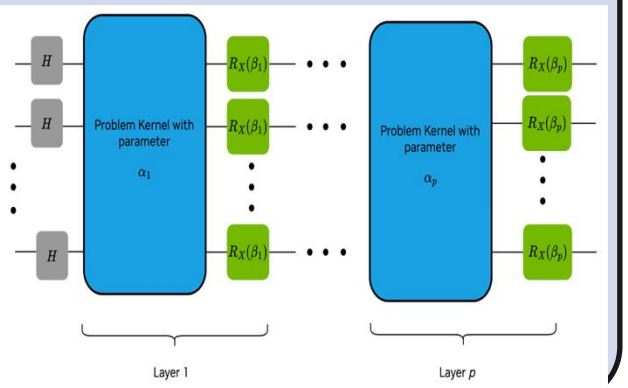
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

Classical Optimization

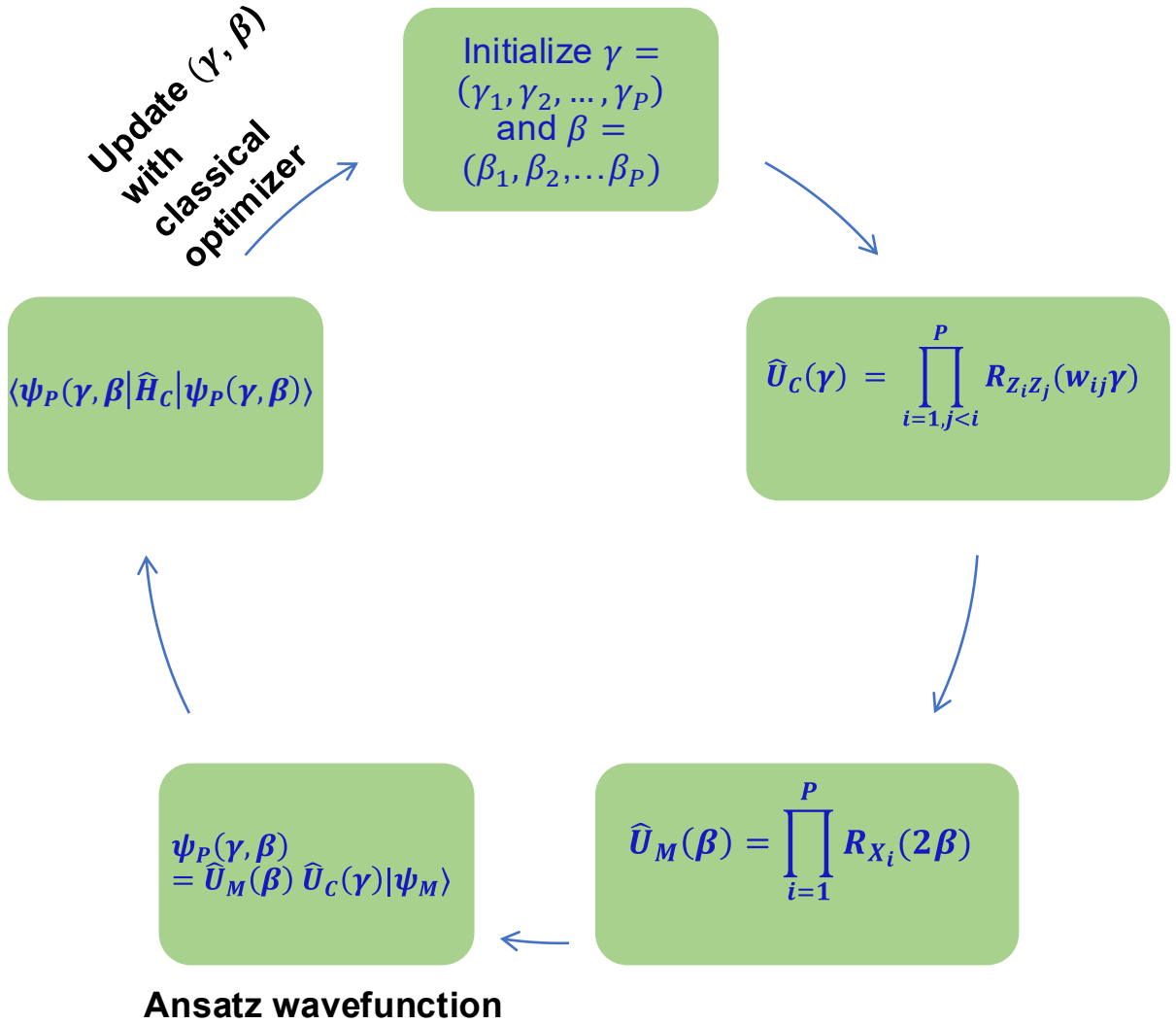
Evaluate energy for ansatz wavefunction and update parameters for next iterations.

Process is repeated iteratively until either convergence criteria or maximum iteration is achieved.

Carried out in classical computing machines.

Optimizers mainly fall into 2 categories:

- Direct search (constrained-based): inexpensive, but takes longer
- Gradient based search: expensive but achieves convergence sooner



Carnegie Mellon University PSC University of Pittsburgh

6. Objective Function

In [18]:

```
def objective(parameters):
    return cudaq.observe(
        qaoa_kernel, hamiltonian,
        qubit_count, layer_count, edges_src, edges_tgt, parameters
    ).expectation()
```

Computing Expectation value of Hamiltonian with ansatz state

7. Classical Optimization

```
import time
begin = time.time()
optimizer = cudaq.optimizers.COBYLA()
optimizer.initial_parameters = np.random.uniform(-np.pi/2.0,np.pi/2.0,parameter_count)

opt_val, opt_params = optimizer.optimize(
    dimensions=parameter_count,
    function=objective
)

end = time.time()

print("Optimal expectation:", opt_val)
print("Estimated MaxCut:", -opt_val)
print("Optimal parameters:", opt_params)
print("Time:", end-begin)
```

Constrained Optimization BY Linear Approximations (COBYLA)

```
Optimal expectation: -10.335277630990412
Estimated MaxCut: 10.335277630990412
Optimal parameters: [-1.6298700480064305, -0.36306700537762, 2.2389707330511417, -0.12625871749645728, -0.5285594709457159, -0.4266
673818927459, 0.1573541504547635, 1.9843694275201256, 0.8309058016026942, 0.17891382628859076]
Time: 3.2922561168670654
```

The initial parameters (IP) plays a key role in the convergence.

Range (-1,1): convergence with 2 layers.

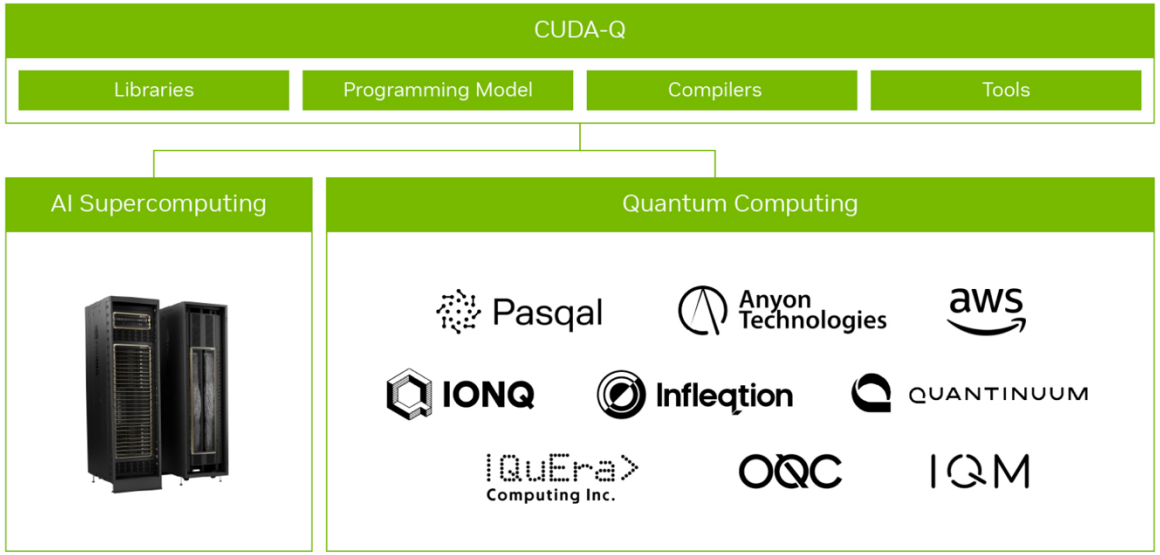
Range $(-\frac{\pi}{2}, +\frac{\pi}{2})$: convergence requires more than 3 layers.

Carnegie Mellon University PSC University of Pittsburgh

CUDA-Q on Bridges2

Carnegie Mellon University PSC University of Pittsburgh

What is CUDA-Q?



- **NVIDIA's Hybrid Quantum-Classical Platform**
- Run on **CPU/GPU simulators or real Quantum Hardware.**

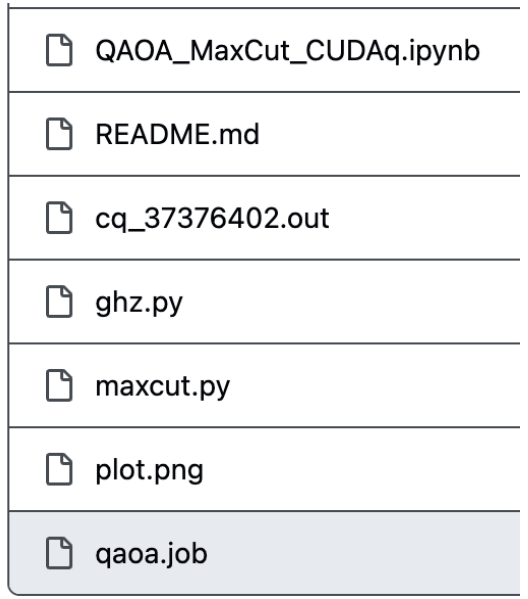
- **Open-source, integrates with HPC & AI workflows.**
- **Supports error correction & noise modeling.**
- **Usage:** import cudaq
- developer.nvidia.com/cuda-q

Carnegie Mellon University PSC University of Pittsburgh

CUDA-Q on Bridges2

- Download tutorial files and cd to QAOA_webinar folder.

```
git clone https://github.com/Neraaz/QAOA_webinar.git
```



- Follow README file in GitHub repository

https://github.com/Neraaz/QAOA_webinar

- Access pre-build CUDA-Q container in Bridges2. Alternatively download here

```
ls /ocean/containers/cuda-quantum.sif
```

Carnegie Mellon University PSC University of Pittsburgh

- Interactive CPU session with 1 RM node and 128 cores

```
interact -p RM –N 1 --ntasks-per-node=128 -t 120:00 (CPU)
```

- Interactive GPU session with 1 V100-32 GB GPUs for 30 minutes:

```
interact -p GPU-shared -N 1 --gres=gpu:v100-32:1 -t 30:00
```

Other available GPUs are v100-16, l40s-48, and h100-80

Note: H100-80 is charged at twice the credit rate compared to other GPUs.

- Open an Apptainer shell (no --nv flag for CPU):

```
apptainer shell --cleanenv --no-home --nv /ocean/containers/cuda-quantum.sif
```

- Set different CUDA-Q backend.

```
cudaq.set_target('qpp-cpu') #CPU

cudaq.set_target('nvidia') #1 GPU
```

```
apptainer> python maxcut.py
```

- Use "exec" instead of "shell" to directly run the program.

```
apptainer exec --cleanenv --no-home --nv /ocean/containers/cuda-quantum.sif
python maxcut.py
```

❑ qaoa.job

```
#!/bin/bash
#SBATCH --job-name=qaoa-job       # Job name
#SBATCH -N 1 # Number of nodes
#SBATCH -p GPU-shared
#SBATCH --gpus=v100-32:1
#SBATCH -t 00:10:00             # Max time for the job
#SBATCH --output=cq_%j.out      # Standard output file


apptainer exec --cleanenv --nv --no-home /ocean/containers/cuda-quantum.sif python3 maxcut.py
```

❑ Submit job to the cluster

```
sbatch qaoa.job
```

Theoretical Framework

Combinatorial optimization problem: Finding optimal solution among a set of feasible solutions

Define cost function

$$C(x) = x^T Q x + b^T x,$$
$$Ax \leq d$$

Hamiltonian

$$\hat{H}_c|x\rangle = C(x)|x\rangle$$

Hamiltonian for Quantum Circuit

$$x_i = \frac{1 - Z_i}{2}$$

Adiabatic Theorem

$$\hat{H} = A(t)\,\hat{H}_M + B(t)\,\hat{H}_c$$

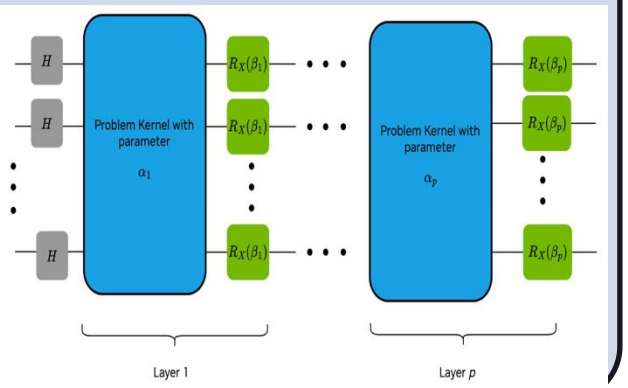
Suzuki-Trotter expansion (For non-commuting operators)

$$\hat{U}(t) \sim \prod_{k=0}^{r-1} e^{-i\hat{H}(k\Delta\tau)\Delta\tau}$$

Classical optimizer
Update parameters

Layered Quantum Circuit

- **Prepare ansatz state**
- **Measurement**



Optimized Results

Carnegie Mellon University PSC University of Pittsburgh

8. Sampling the Optimal Circuit

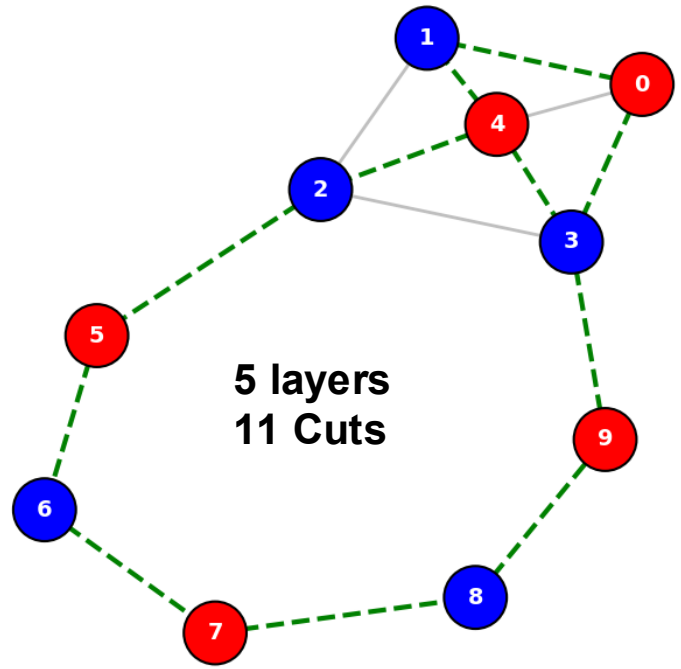
In [21]:

```
counts = cudaq.sample(
    qaoa_kernel, qubit_count, layer_count,
    edges_src, edges_tgt, opt_params
)

best_bitstring = max(counts, key=lambda x: counts[x])
print("Most likely solution:", best_bitstring)
```

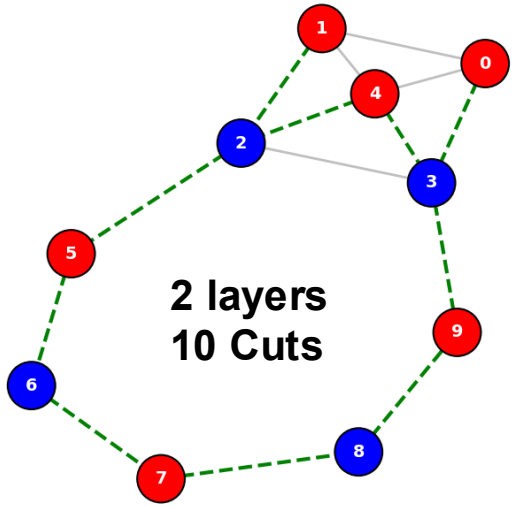
```
Most likely solution: 1000110101
```

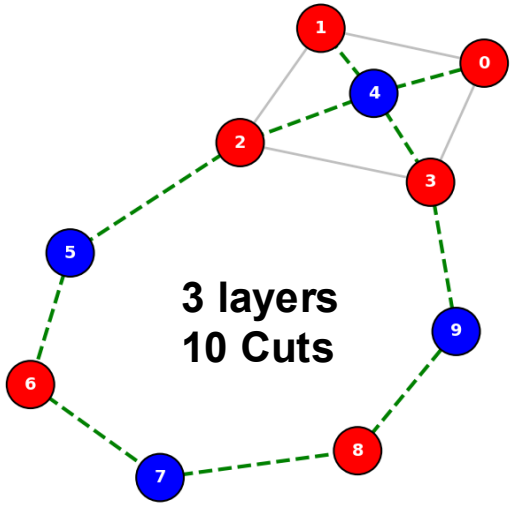
1 → 

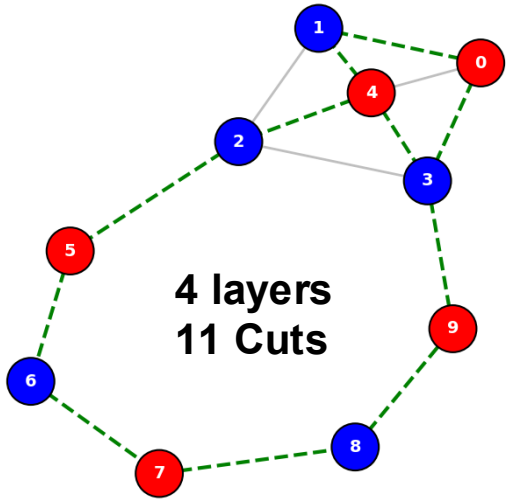
0 → 



MaxCut vs layer







- 10 nodes and 14 edges
- Initial parameters: $(-\pi/2,+\pi/2)$
- At least 4 layers quantum circuit is needed.
- Number of maximum cuts is 11

Carnegie Mellon University PSC University of Pittsburgh

More ……

CUDA-Q Applications

This page contains a number of different applications implemented using CUDA-Q. All notebooks can be found here.

Filter by Domain:

All | Optimization | Chemistry | Fundamental Algorithms | AI for Quantum | Quantum for AI | Dynamics | Community

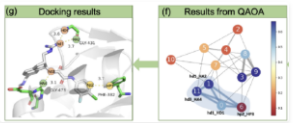
Filter by Backend:

Noiseless Simulator ▾ | Noisy Simulator ▾ | QPUs ▾

QAOA for Max Cut Problem

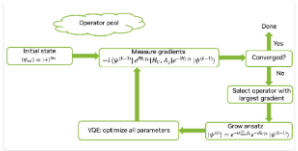
Learn the theory behind the Quantum Approximate Optimization Algorithm (QAOA) and how it can be used to solve the Max Cut problem.
#optimization #noiseless #gpu

Digitized Counterdiabatic QAOA



Learn how the DC-QAOA algorithm is used to predict molecules that might be good candidates for drugs based on their interactions with proteins.
#chemistry #optimization #noiseless #gpu

ADAPT QAOA

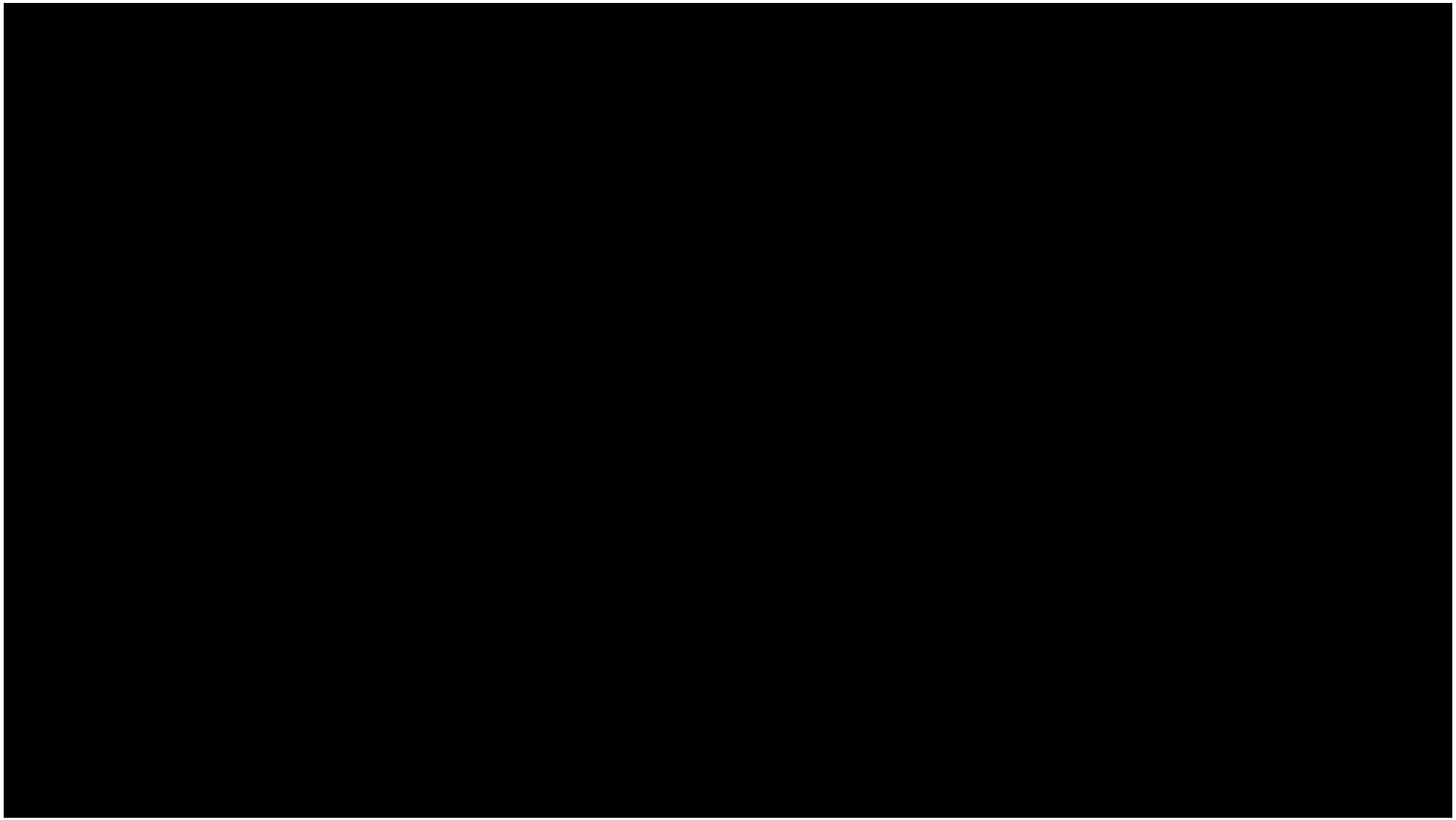


Learn how to implement the Adaptive Derivative-Assembled Pseudo-Trotter (ADAPT) ansatz QAOA using CUDA-Q. The method iteratively builds an ansatz to more efficiently converge to the ground state of a problem Hamiltonian.
#optimization #noiseless #gpu

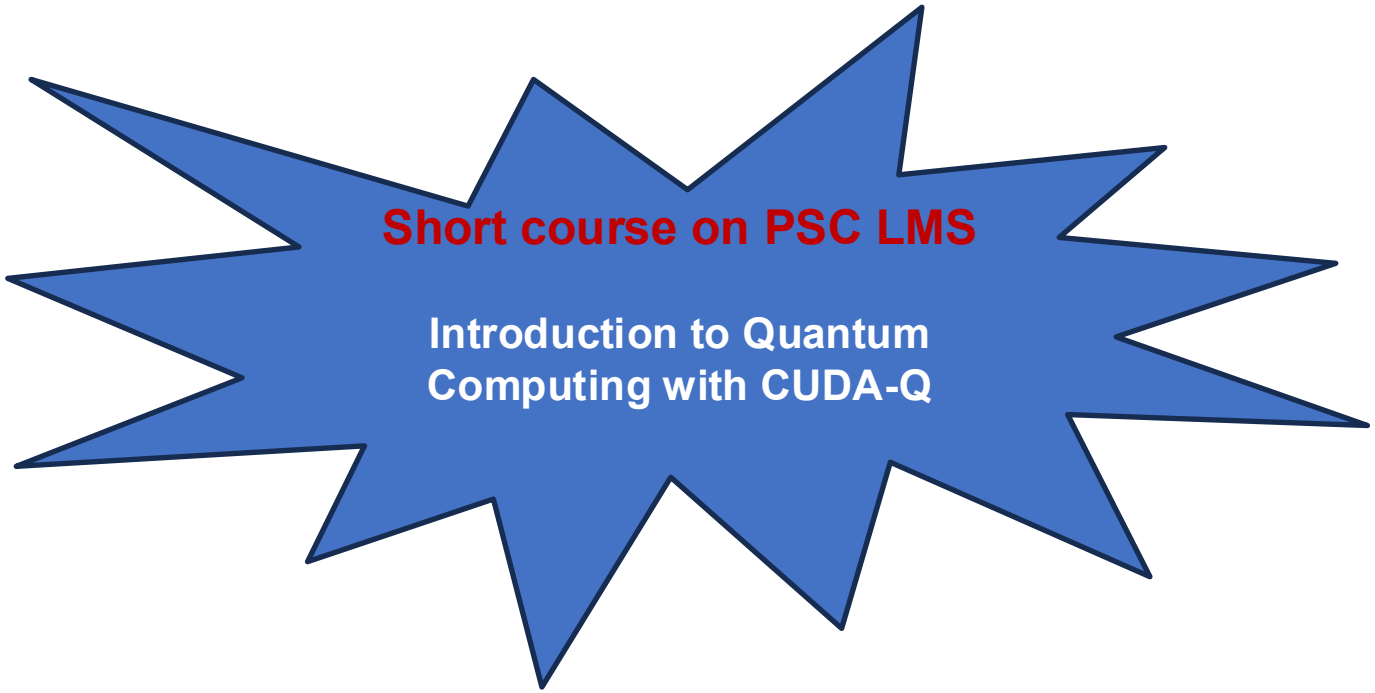
https://nvidia.github.io/cuda-quantum/latest/using/applications.html

Carnegie Mellon University | PSC PITTSBURGH SUPERCOMPUTING CENTER | University of Pittsburgh

Demo (Video Link)



Carnegie Mellon University PSC University of Pittsburgh



CUDA-Q Basics

+

Different Simulation Backends:

qpp-cpu, nvidia, mgpu, mqpu, tensornet, tensornet-mps

Carnegie Mellon University PSC University of Pittsburgh

References

- Blekos, Kostas, et al. "A review on quantum approximate optimization algorithm and its variants." Physics Reports 1068 (2024): 1-66.
- https://nvidia.github.io/cuda-quantum/latest/applications/python/qaoa.html
- https://github.com/Neraaz/CUDA-Q-PSC
- CUDA-Q Installation

Thank you!

PSC
PITTSBURGH SUPERCOMPUTING CENTER

Carnegie Mellon University

University of Pittsburgh

CUDA-Q Basics

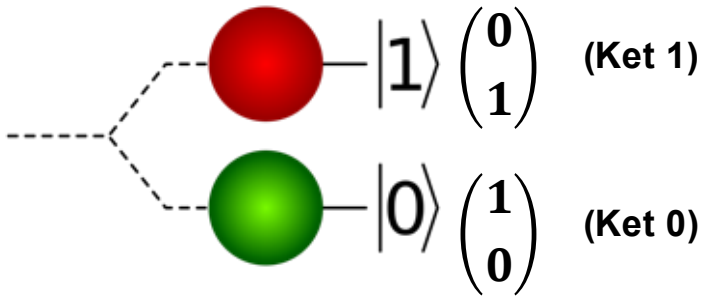
Carnegie Mellon University PSC University of Pittsburgh

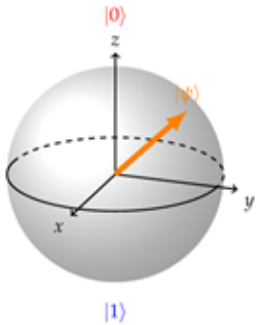
Qubits

- **Qubit**: the fundamental unit of quantum information
  Two-level quantum system: energy levels, atomic or photonic polarizations, etc.
  Platforms: superconducting circuits, Laser-trapped ions, neutral atoms, or photons, etc.

- The quantum state of a qubit system can be described as a superposition of its computational basis states.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$





Bloch Sphere

- An N-qubits state has 2^N **basis states**, its quantum state in matrix form is represented by a column vector of 2^N **complex coefficients**.

- For a 2 qubits system: the **basis is**: $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, $|ij\rangle = |i\rangle \otimes |j\rangle$

Carnegie Mellon University PSC University of Pittsburgh

Qubits in CUDA-Q

`cudaq.qubit()` → single qubit

`qubits = cudaq.qvector(N)` → build a register of N qubits.

`qubits[0]` → accessing the first qubit

`cudaq.qvector(vec)` → passing complex vectors (amplitude)

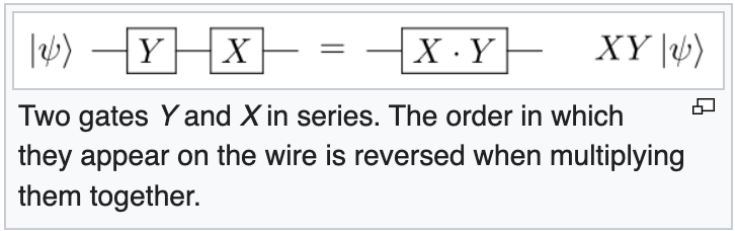
`cudaq.qvector(state)` → passing state from another kernel,
`State=cudaq.get_state(kernel_initial)`

Quantum Gates

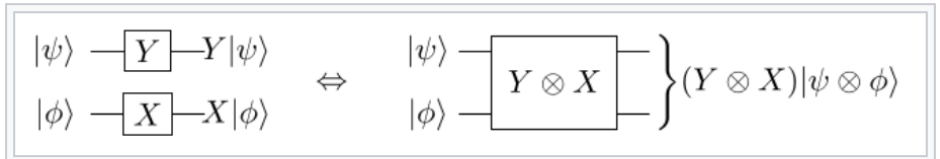
- **Quantum gates are used to manipulate the state of one or more qubits.**

$$G|\psi\rangle \rightarrow |\psi'\rangle$$

$2^N \times 2^N$ Matrix



Two gates *Y* and *X* in series. The order in which they appear on the wire is reversed when multiplying them together.



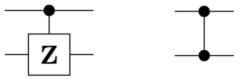
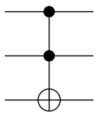
Two gates Y and X in parallel is equivalent to the gate $Y \otimes X$.

- **Quantum gates are Unitary Operators**

$G^\dagger G = GG^\dagger = I$ (reversible, probability preserved)

$G^\dagger = (G^*)^T$ (conjugate transpose)

- **They enable superposition, entanglement, and other quantum effects.**

Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
π/8 (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

https://en.wikipedia.org/wiki/Quantum_logic_gate

Carnegie Mellon University PSC University of Pittsburgh

Single qubit gates

- **Pauli's Gates**

$$X|0\rangle = |1\rangle, X|1\rangle = |0\rangle \text{ (Bit-flip (NOT) gate)}$$
$$Z|0\rangle = |0\rangle, Z|1\rangle = -1|1\rangle \text{ (Phase-flip gate)}$$
$$Y|0\rangle = i|1\rangle, Y|1\rangle = -i|0\rangle, \text{ (Y = iXZ)}$$

- For example, to build an equal **superposition from a single qubit state**, we use the Hadamard gate

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle,$$
$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle$$

- **Rotation gates**

$$R_j(\theta) = e^{-\frac{i\theta}{2}j}, j = \{X, Y, Z\}$$

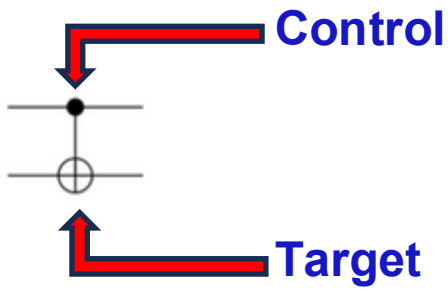
CUDA-Q

- `G(qubit)` → **G is quantum gates**
- **Pauli's X-gate on second qubit:**
  `x(qubits[1])`
- **Hadamard (h) gate on first and second qubit,**
  h(qubits[0])
  h(qubits[1])
- `RG(angle, qubit)` → **rotation gates. RG=rx, ry, rz**
- **List of other operations: here**

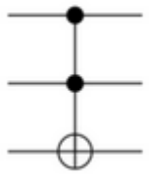
Multi qubit gates

- To create entanglement, we need gates that operate on 2+ qubits

Controlled Not
(CNOT, CX)



Toffoli
(CCNOT,
CCX, TOFF)



CUDA-Q

❑ **CNOT (CX)**

```
x.ctrl(qubits[0], qubits[1])
```

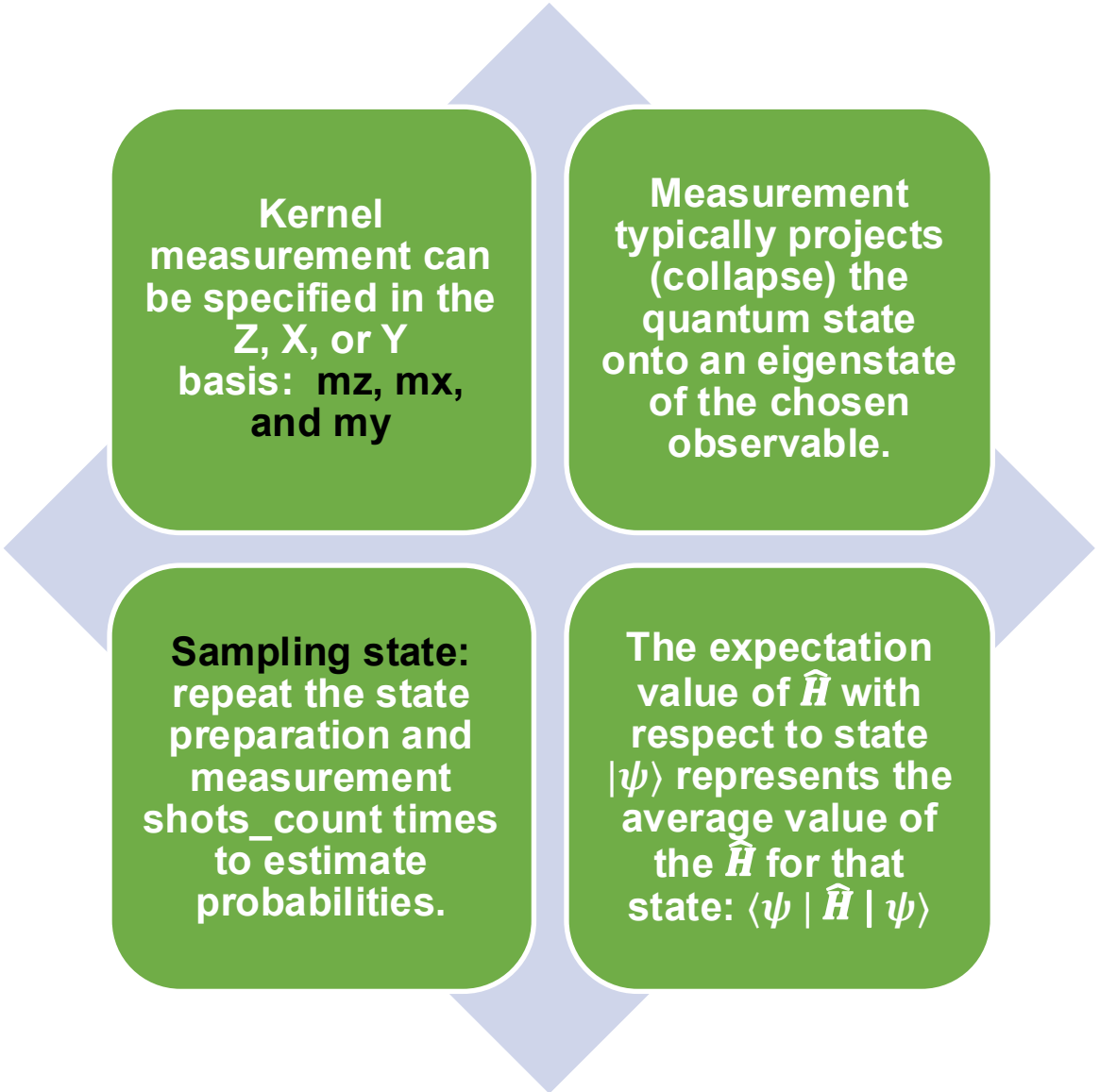
X is applied to the second qubit if the first qubit is in |1> state.

❑ **CCX**

```
x.ctrl([qubits[0], qubits[1]], qubits[2])
```

X is applied to third, if the first and second qubits are in |1> state.

Measurement



mz(qubits) → measurement performed in the **Z** basis (default) and collapses each qubit to $|0\rangle$ or $|1\rangle$

cudaq.sample(kernel, *kernel_args, shots_count) **More details**

cudaq.observe(kernel, Hamiltonian, *kernel_args, execution).expectation(): **More details**

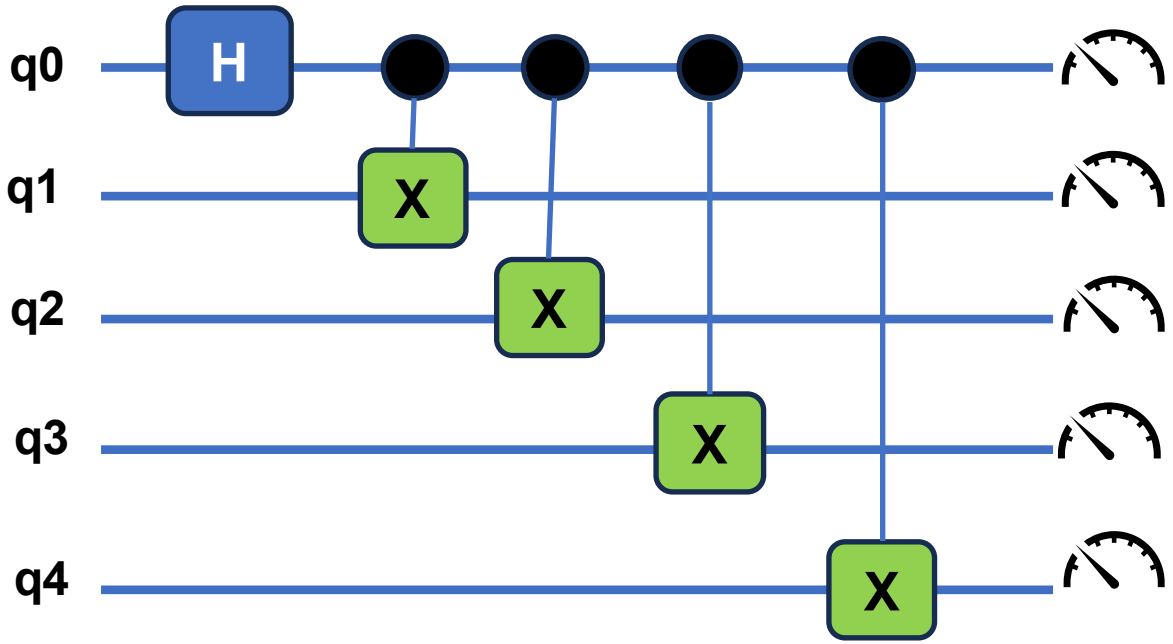
$$\langle\psi|\hat{H}|\psi\rangle = \int \psi^*(\boldsymbol{r})\, \hat{H}\, \psi(\boldsymbol{r})\, d\boldsymbol{r}$$

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

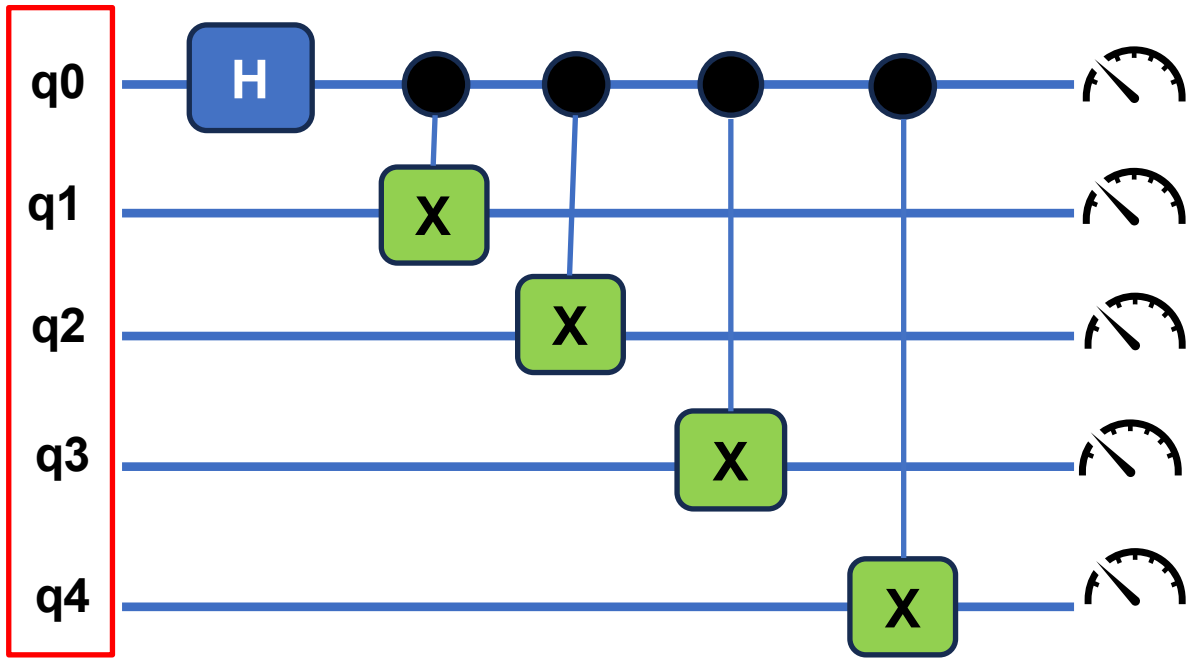
```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

Computational basis notation

$|00000\rangle$

Or

Tensor product notation

$|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle$

Or

Short tensor product notation

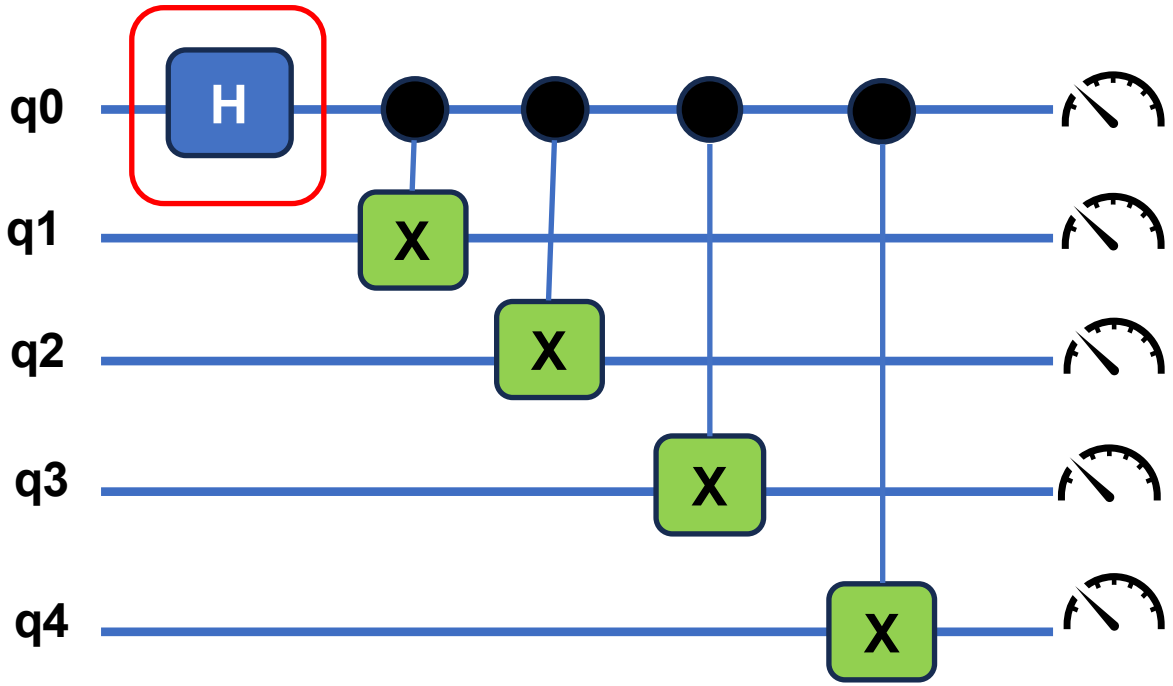
$|0\rangle^{\otimes 5}$

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

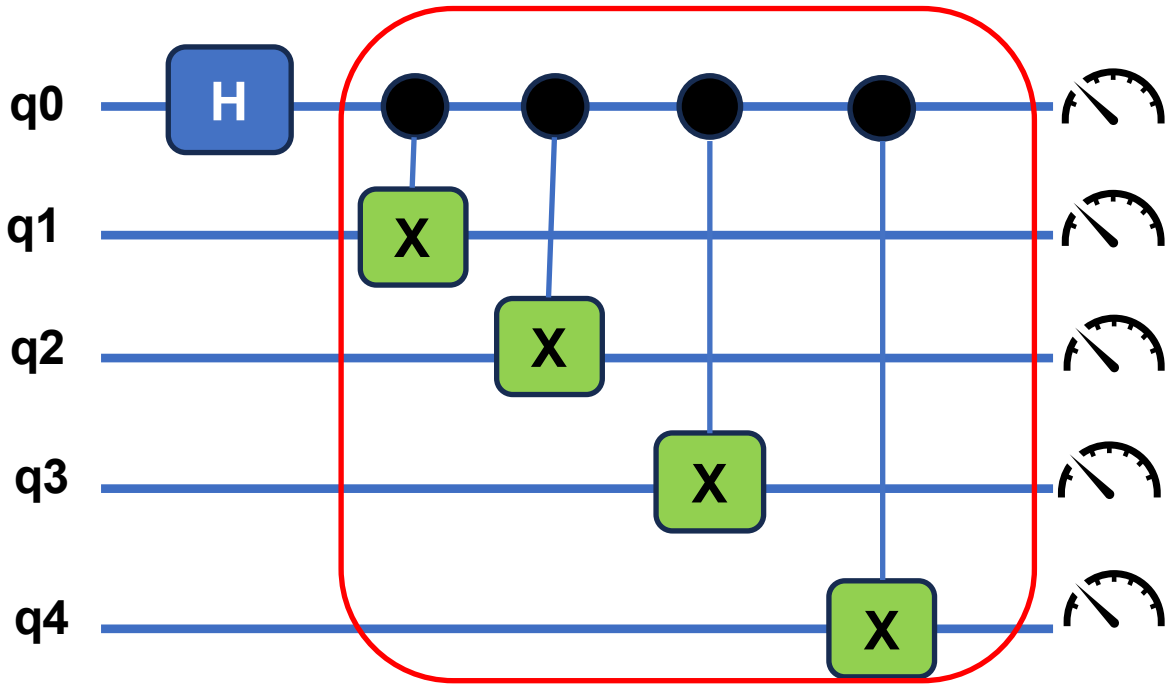
$$\frac{1}{2^{1/2}}[\,|0\rangle + |1\rangle] \otimes |0\rangle^{\otimes 4}$$

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

```
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

$$\frac{1}{2^{1/2}}[\,|0\rangle + |1\rangle] \otimes |0\rangle^{\otimes 4}$$

$$\mathbf{CNOT}(q_0, q_i)$$

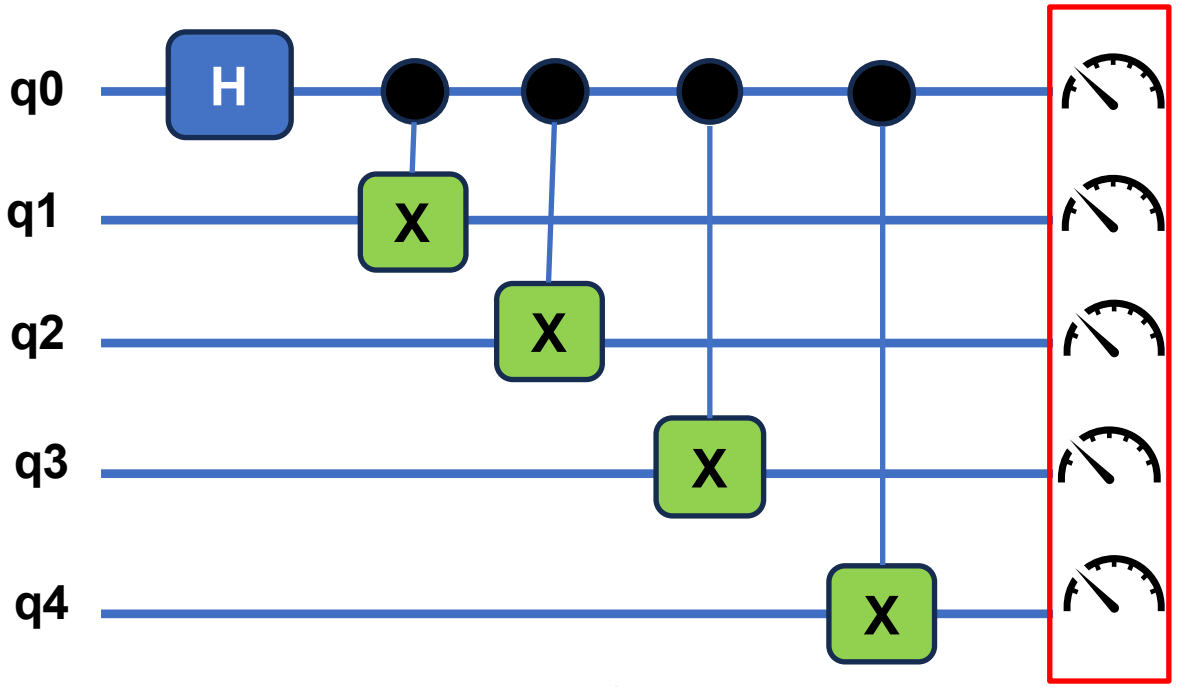
$$\frac{1}{2^{1/2}}[\,|0\rangle^{\otimes 5} + |1\rangle^{\otimes 5}]$$

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

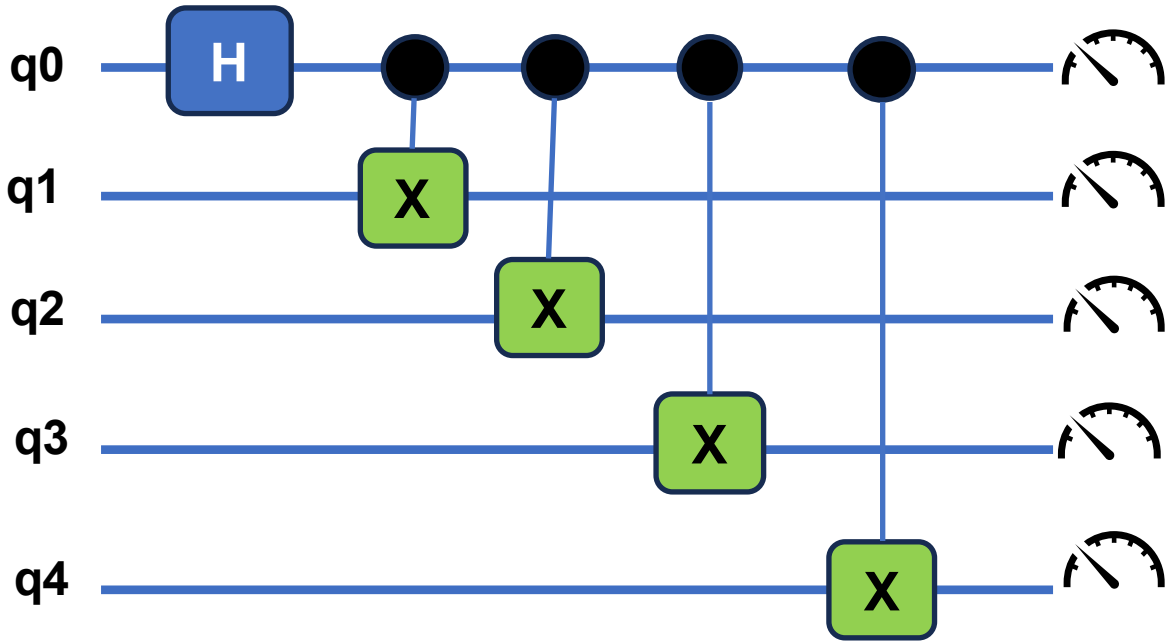
```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

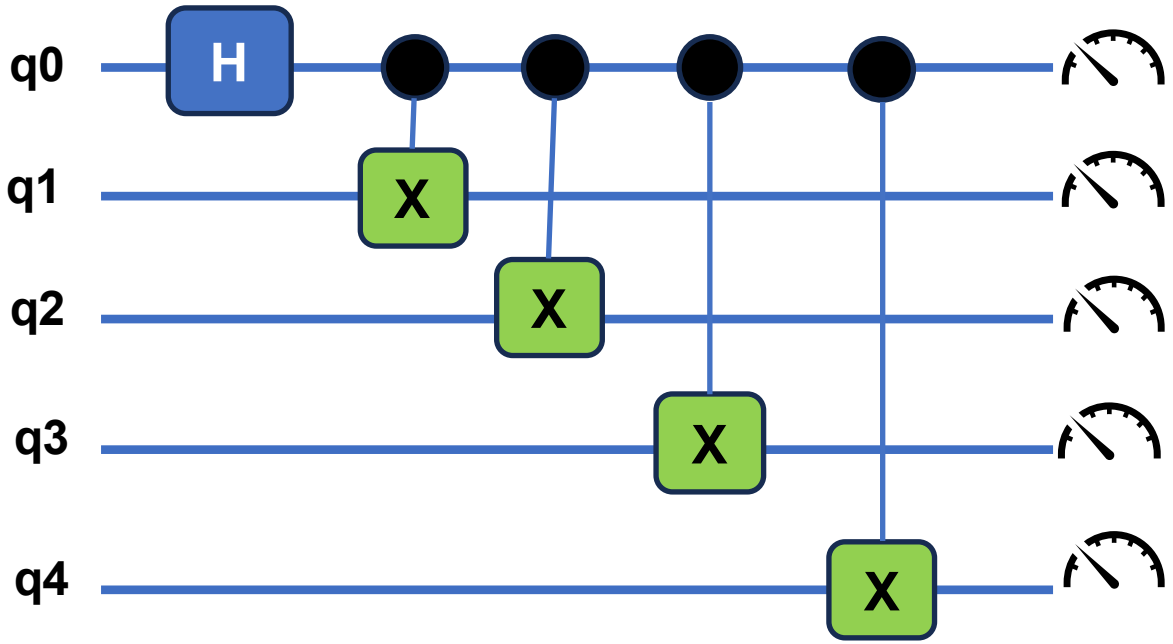
Other backends include nvidia, mgpu, mqpu, tensornet, tensornet-mps

Carnegie Mellon University PSC University of Pittsburgh

Quantum Circuit (CUDA-Q kernel)

A quantum circuit (program) is a sequence of operations (gates) on an initial quantum state, ending with a measurement.

Preparing Greenberger–Horne–Zeilinger (GHZ) state



$$|GHZ_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$$

```python
import cudaq
# Define a quantum kernel that returns an integer
@cudaq.kernel
def ghz_kernel(qubit_count: int):
    # Allocate qubits
    qubits = cudaq.qvector(qubit_count)

    # Create GHZ state
    h(qubits[0])
    for i in range(1, qubit_count):
        x.ctrl(qubits[0], qubits[i])

    # Measure qubits in z-basis
    mz(qubits)

if __name__ == "__main__":
    #CPU only backend
    cudaq.set_target('qpp-cpu')

    #Change qubit count
    qubit_count =  5

    #Draw circuit
    #print(cudaq.draw(ghz_kernel, qubit_count))

    #Sample the circuit
    results = cudaq.sample(ghz_kernel, qubit_count, shots_count=1000)
```

Results: { 00000:495 11111:505 }

2 states out of $2^5 = 32$

Carnegie Mellon University PSC University of Pittsburgh