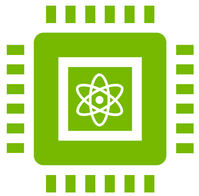
Simulating Quantum Computer on GPUs

Dr. Khaldoon Ghanem | Senior DevTech Engineer, Nvidia

Munich Quantum Software Forum - 21.10.2025

Simulating Quantum Computers

Importance & Applications



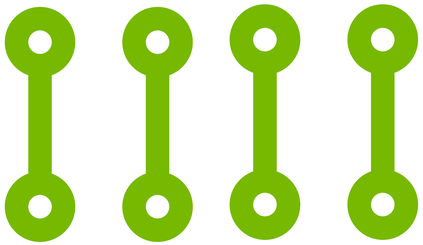
Quantum Hardware Design and Benchmarking

Quantum Algorithm Development and Research

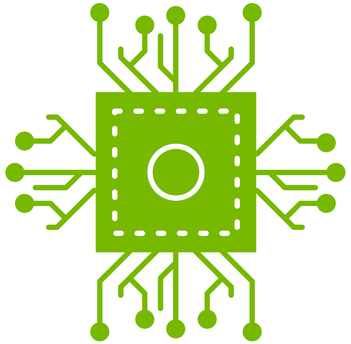
Educational Tool for Quantum Computing

Simulating Quantum Computers

Why GPUs?

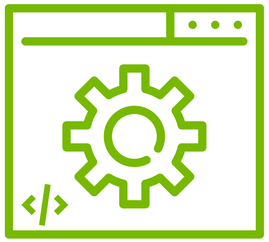


Massive Parallelism



High Memory and Interconnect Bandwidths

Energy Efficiency



Optimized GPU-Accelerated Libraries

Simulating Quantum Computers

Simulation Levels in NVIDIA Quantum Platform

High-level framework

CUDA-Q
Open-Source Software Platform for Quantum-Accelerated Supercomputers

Enable

Accelerate

- Quantum Researchers
- Quantum Algorithm Developers
- End users of QPUs:

Low level libraries

cuQuantum SDK
Accelerated Quantum Simulation/Emulation on GPU

cuStateVec
State Vectors

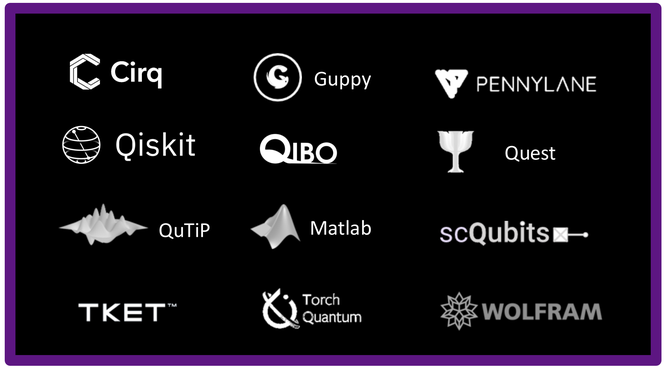
cuTensorNet
Tensor Networks

cuDensityMat
Quantum Dynamics

Digital simulation of quantum circuits

Analog simulation of physical qubits/hardware

- Developers of Quantum Simulation Software:



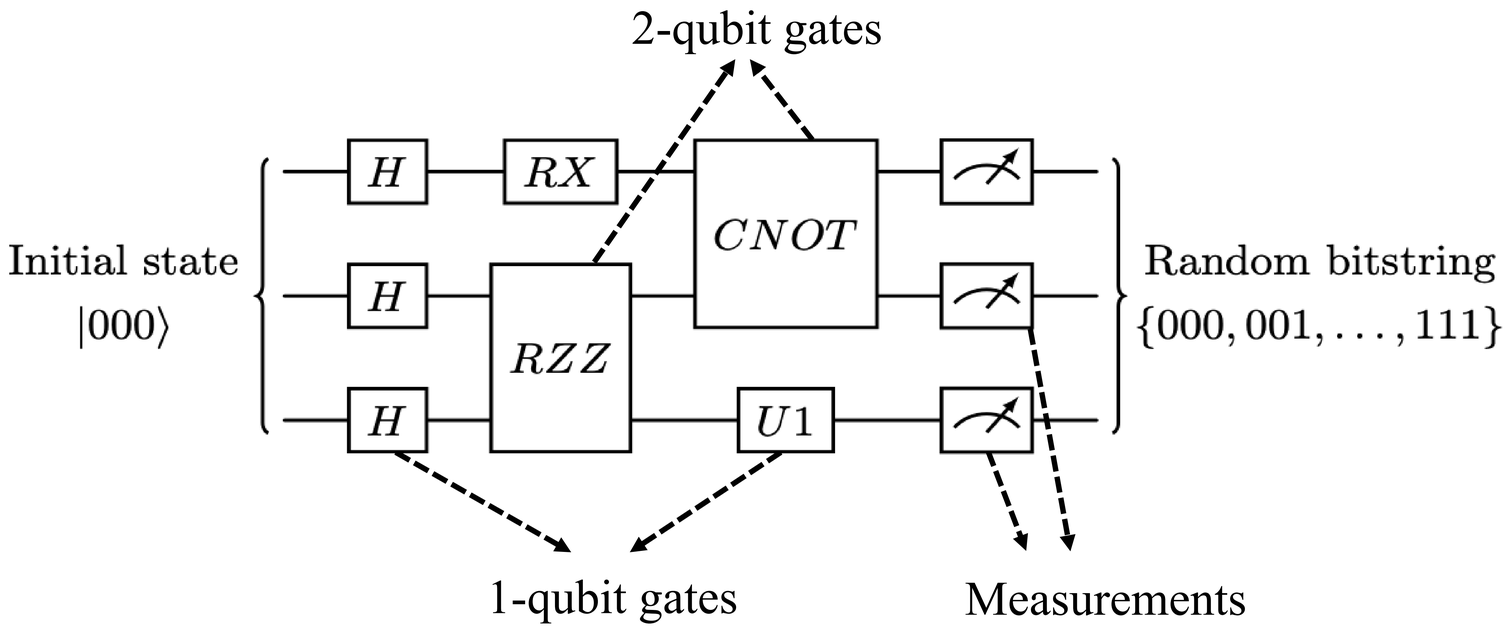
Target Audience

NVIDIA

Quantum Circuit Simulation

Quick Recap

- In gate-based quantum computing, programs are expressed as quantum circuits.
- A circuit of n-qubits is visually represented by n wires.
- Operations on the qubits, called gates, are represented by boxes on the corresponding wires.
- Gates often act on single or two qubits. Higher-qubit gates are used less often.
- At the end of the circuit, qubits are measured to produce a random bit string following the quantum state of the circuit.
- **Example**:



Quantum Circuit Simulation

State Vector Model

- The pure state of n-qubits quantum system is a weighted superposition of all possible classical bits.
- It is represented as a vector of complex number of size 2^n.

Example: 2-qubits state $|\psi\rangle = c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle \iff \psi = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

- Each number, called quantum amplitude, whose squared amplitude $|c_i|^2$ represents the probability of observing the corresponding bit string i.
- Memory requirement for n −qubits: single precision $2^n \times 2 \times 4 = 2^{n+3}$ bytes. Double precision 2^{n+4} bytes.
  - **Example:** 30-qubits requires 8 GB in single precision and 16GB in double precision.

- Gates on n-qubits are represented by $2^n \times 2^n$ unitary matrix

Example: 1-qubit Hadamard Gate

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Example: 2-qubit CNOT gate

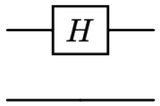
$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Quantum Circuit Simulation

State Vector Model

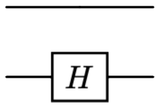
- **State Vector Simulation – Main Loop:**
  - Initialize state vector to the state |00..0> (i.e. vector [1, 0, 0, …, 0]^T)
  - For each gate (from left to right):
    - Multiply the state vector by the corresponding matrix.
  - Convert state vector into cumulative distribution function and sample e.g. via **Inverse transform sampling**

- **Gate Application:** How to apply m-qubit gate ($2^m \times 2^m$ matrix) to n-qubit state (2^n vector)?
  - Formally, the applied matrix is the tensor product of the gate matrix with identity matrices on the skipped qubits.
  - Example 1: Applying Hadamard on the **first** qubit of a 2-qubit state:



$$\mathrm{I} \otimes \mathrm{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \mathrm{H} = \begin{bmatrix} \mathrm{H} & 0 \\ 0 & \mathrm{H} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Example 2: Applying Hadamard on the **second** qubit of a 2-qubit state:



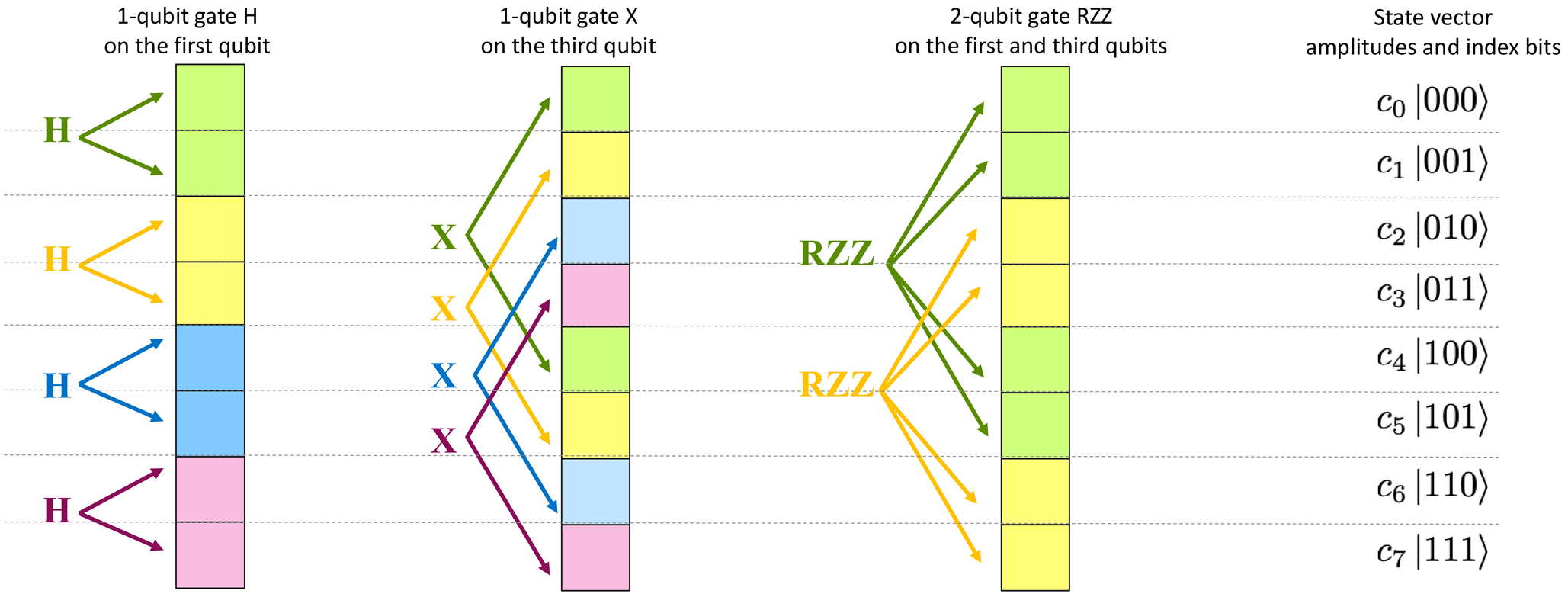
$$\mathrm{H} \otimes \mathrm{I} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \otimes \mathrm{I} = \begin{bmatrix} h_{11}\,\mathrm{I} & h_{12}\,\mathrm{I} \\ h_{21}\,\mathrm{I} & h_{22}\,\mathrm{I} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

  - Naively, this leads huge sparse matrices ($2^n \times 2^n$).

State Vector Simulation

Gate Application

- In practice, the sparse matrices are never explicitly formed. Instead, state vector of size 2^n is divided into 2^{n-m} substate vectors of size 2^m and the gate matrix of size $2^m \times 2^m$ is applied to each substate vector independently.
- Each substate vectors is composed of amplitudes whose index bit representation differ by exactly m bits. The location of these m bits are determined by the m qubits the gate is working on.
- **Examples:** Three qubit state vector



Gate Application

GPU Implementation

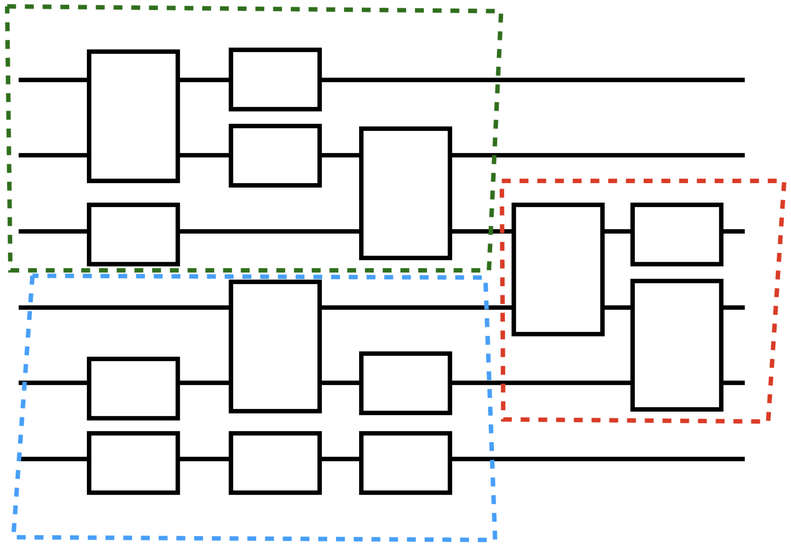
- Basic GPU implementation for applying an m-qubit gate:
  - Launch CUDA kernel with 2^{n-m} threads.
  - Each thread fetches 2^m amplitudes, apply the gate matrix and writes the amplitudes back to memory.

- **cuStateVec** is a high-performance C library dedicated to operations on state vectors on Nvidia GPUs.

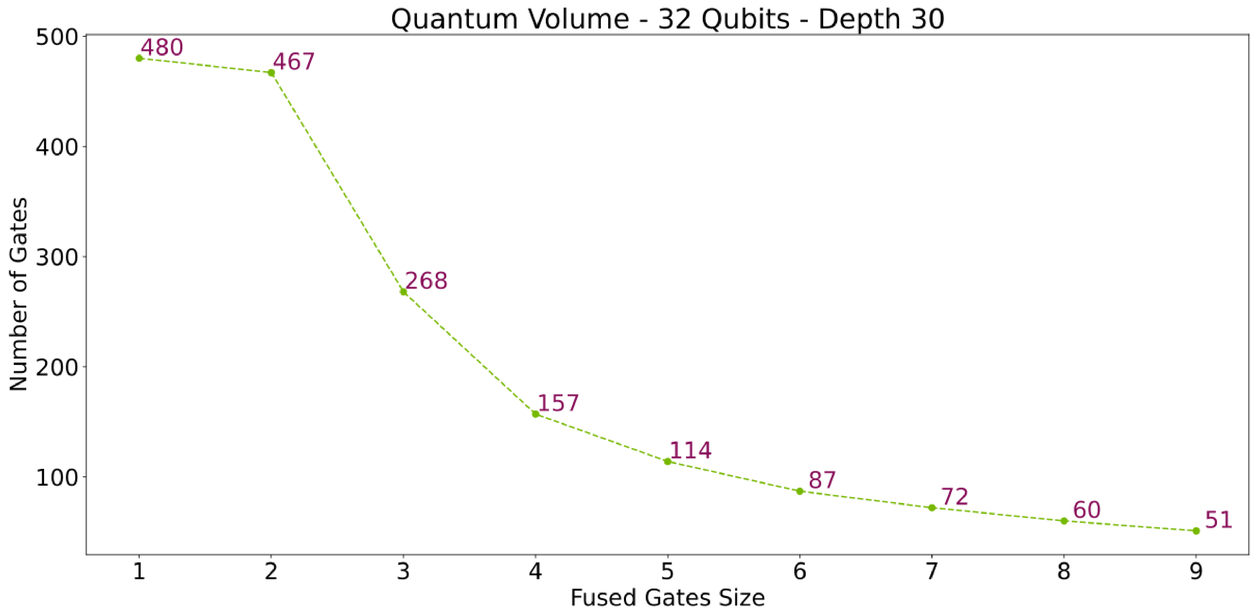
- Gate application in cuStateVec is highly-optimized:
  - Hardware-Specific Tuning:
    - Use of CUDA-specific intrinsics and fast math
    - Use shared memory, memory access coalescing
    - Tailoring kernels to specific GPU architectures (e.g., Hopper vs. Blackwell)
    - Leverage tensor cores and use emulation when beneficial
  - High-level optimizations:
    - Special handling of control qubits:
      - Reduces the computation
    - Special handling of diagonal gates:
      - Skips zero elements
    - Facilitate **gate fusion** by supporting arbitrarily large gates and customized kernels depending on gate size and hardware.

Gate Fusion

Accelerating gate applications



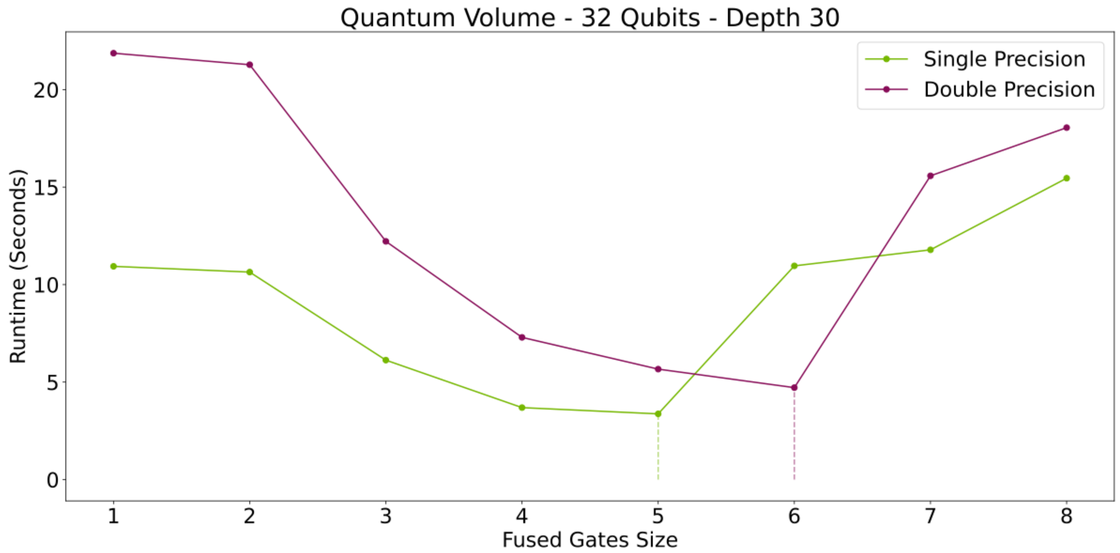
- **Issue:** almost all circuits are expressed in terms of 1 and 2-qubits gates:
  - Low arithmetic intensity => memory-bound kernels => low utilization of GPU performance.
- **Solution:** gate fusion
  - Combine several neighboring gates in a quantum circuit into fewer larger gates.
  - It is parameterized by fusion size: the largest number of qubits of fused gate.
- **Goal:** speedup calculation by
  - Reducing the total number of applied gates.
  - Increasing the arithmetic intensity of each applied gate.
- CUDA-Q provides **nvidia** backend: a cuQuantum-based simulator with automatic gate-fusion.
- **Example:**
  - Quantum Volume Circuit on 32 qubits with 30 layers.
  - Original circuit has 480 2-Qubit Gates.
  - Gate fusion reduces the number of gate applications almost proportionally to fusion size.



Gate Fusion

Impact of fusion parameter

- **Example:** Runtime on H100 with different gate fusion sizes

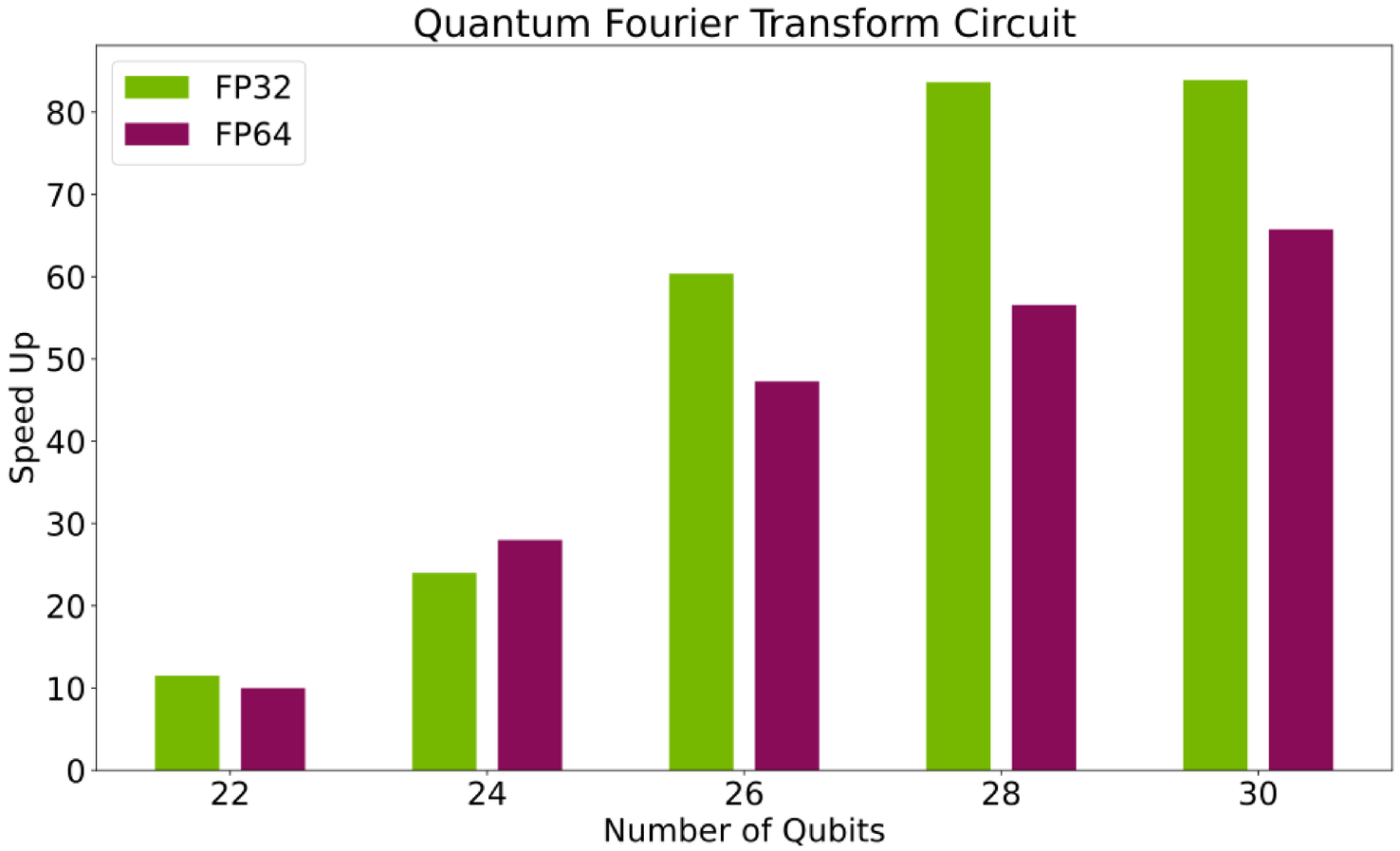


- Optimal gate fusion here is 5 for FP32 and 6 for FP64.
- Why runtime increase after that?
  - Gate fusion increases the total number of FLOPs (larger gates => exponentially larger matrices)
  - This increase in FLOPs is harmless, **as long as the kernel is memory-bound** (its effect on runtime is cancelled by the increased arithmetic intensity).
  - Beyond that point, runtime increases.
- CUDA-Q **nvidia** backend automatically selects the optimal gate fusion depending on the GPU architecture.
  - If necessary, it can be overridden via environment variable : CUDAQ_FUSION_MAX_QUBITS

State Vector Simulation

CPU vs. GPU

- GPUs generally shows considerable speed up against CPUs for state vector simulations above 20 qubits
- CUDA-Q on B200 GPU vs. Qiskit on high-end 128-Cores CPU



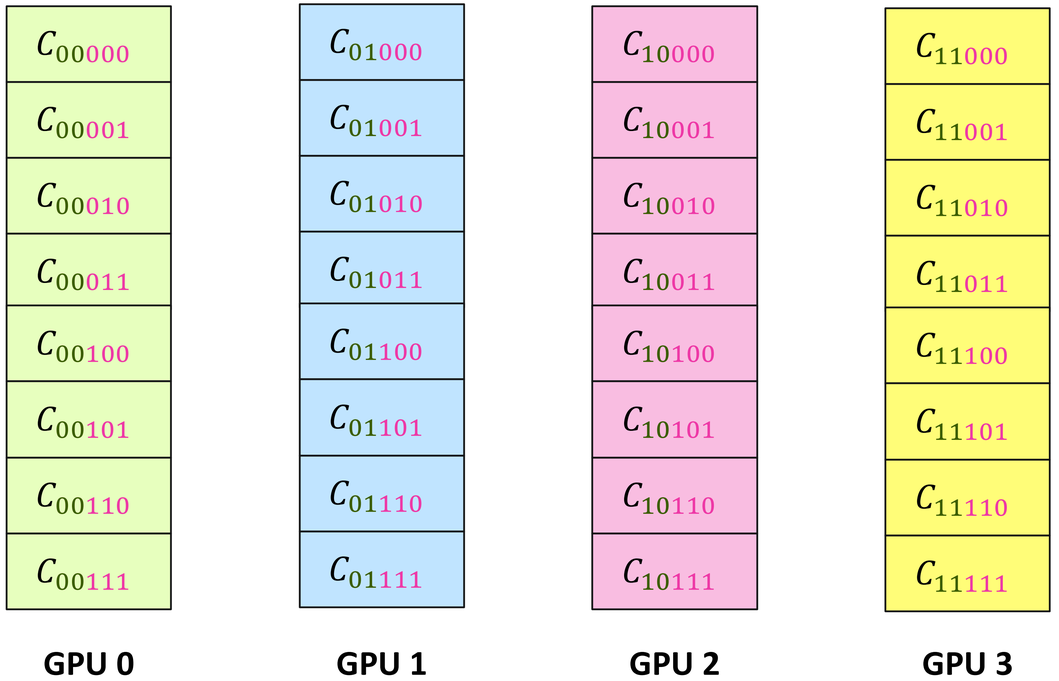
- Benchmarked using: https://github.com/SRI-International/QC-App-Oriented-Benchmarks

Multi-GPU State Vector Simulation

- Why need multi-GPU Simulations?
  - Memory requirement: State vectors are huge. Largest GPU memory 192 GB can accommodate at most 34 qubits (single precision).
  - Performance: Speedup simulation of large deep circuits.

- How to to do it?
  - In most quantum simulators, including cuQuantum, the number of GPUs need to be a power of two. It makes things much easier.
  - Each GPU is typically associated with an MPI process. Data is exchanged via MPI or CUDA IPC.
  - Statevector is split equally between all 2^g GPUs.
  - This effectively divides the n bit representation of full state vector indices into: g bits for GPU index (global qubits) and $(n-g)$ bits for local vector index (local qubits).

- **Example:**
  - 5-qubit state vector distributed over 4 GPUs.
  - 2 global qubits.
  - 3 local qubits.

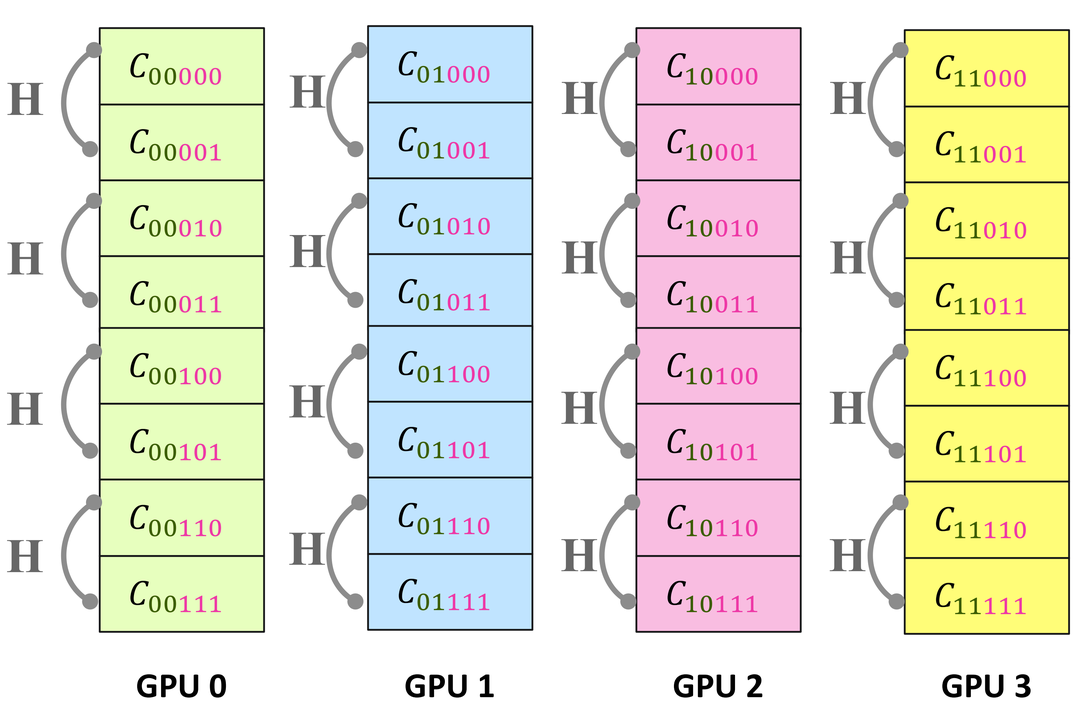


Multi-GPU State Vector Simulation

Local vs. global qubits

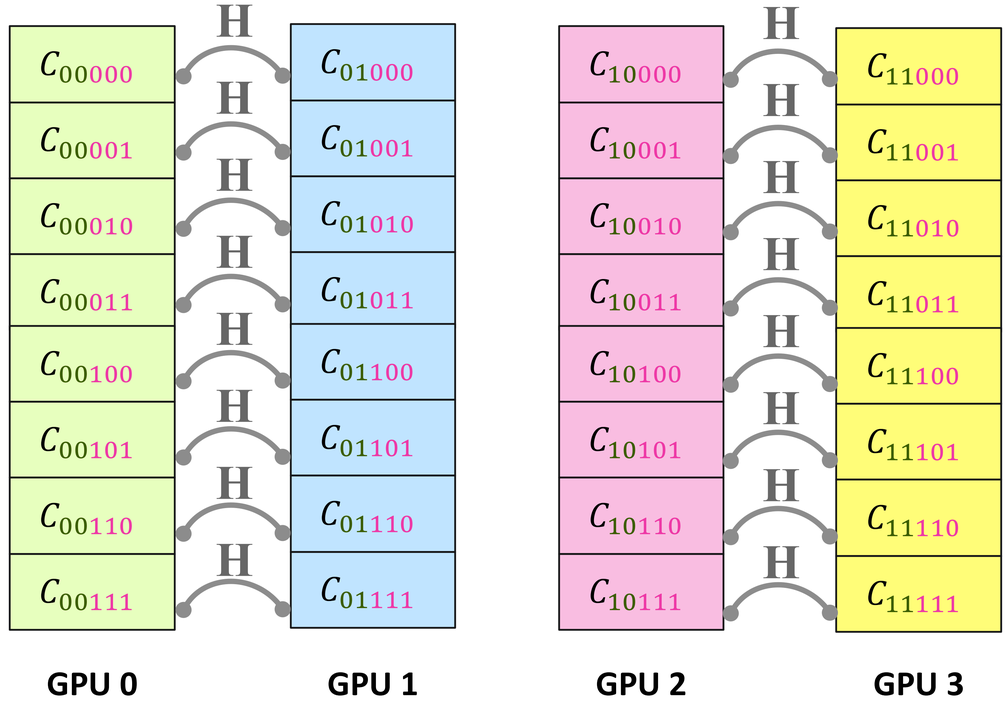
- Applying gates to local qubits does not require communication between GPUs. Each GPU can apply the gate to its sub-statevector. Hence the name local.
- Applying gates to global qubits requires amplitudes stored on different GPUs => communication is required.
- Multi-qubit gates that involve both local and global qubits also require communication.

Example:



Applying Hadamard gate to first qubit (qubit 0)

local qubit – no communication



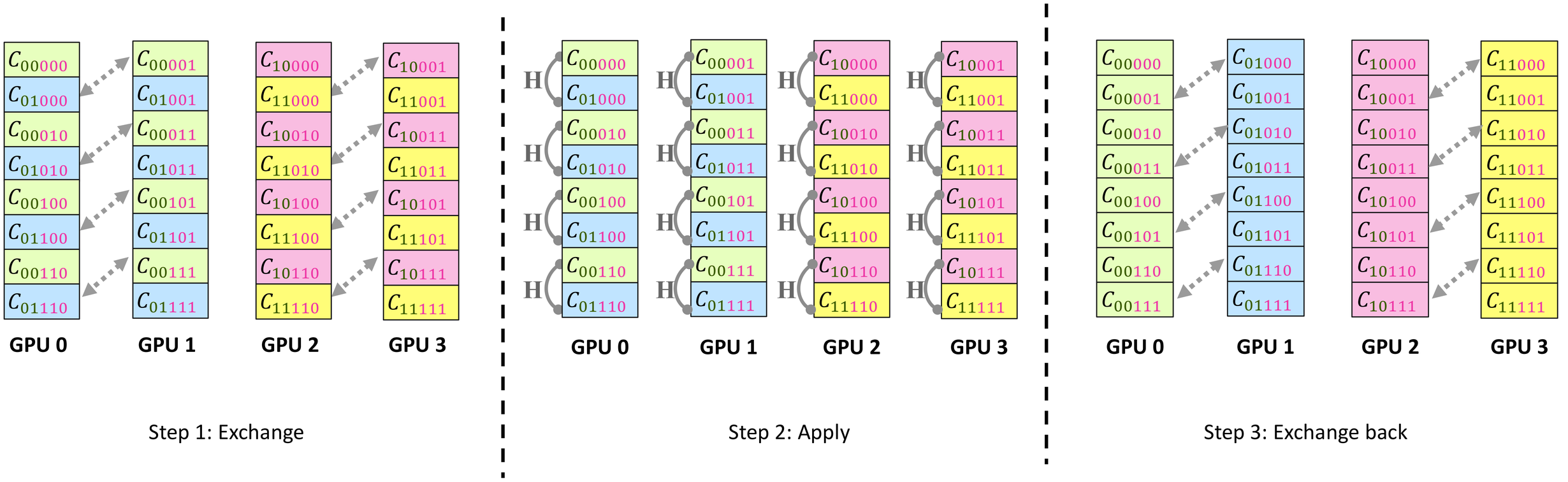
Applying Hadamard gate to fourth qubit (qubit 3)

global qubit –communication needed

Multi-GPU State Vector Simulation

Naïve gate application on global qubits

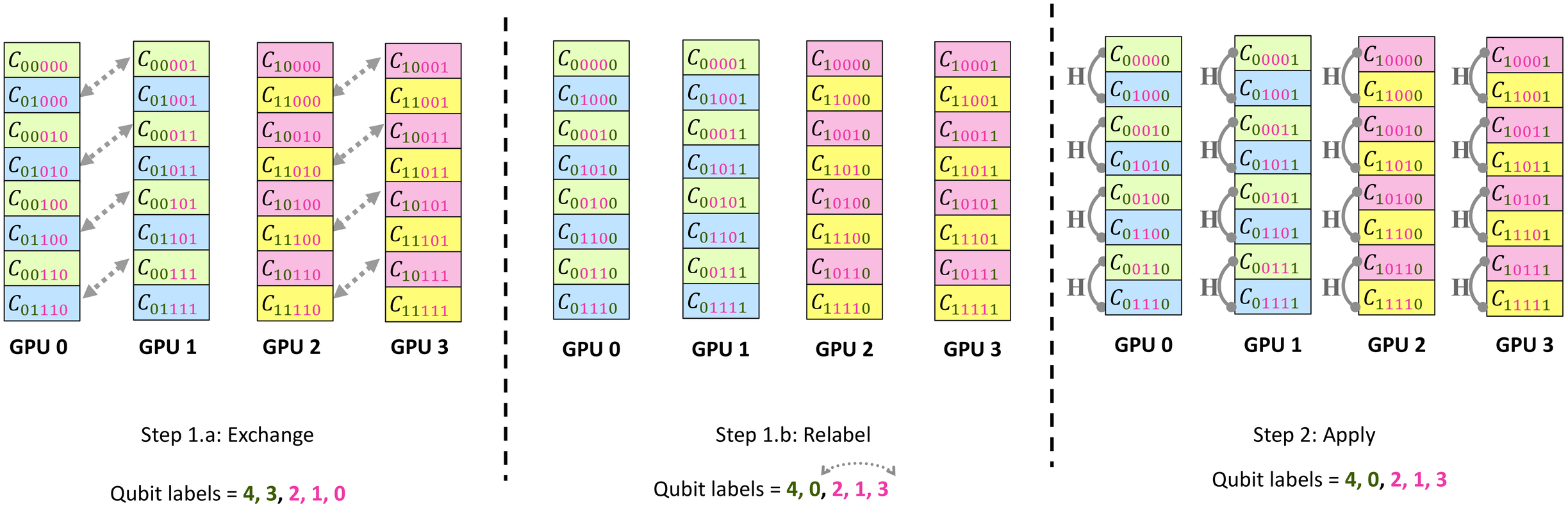
- Naïve way to implement single-qubit gates on global qubits:
  1. Each pair of GPUs exchange half of their amplitudes.
  2. Apply the gate locally.
  3. Exchange the amplitudes back.
- **Example:** Applying Hadamard gate to fourth qubit (qubit 3)



Multi-GPU State Vector Simulation

Qubit swaps

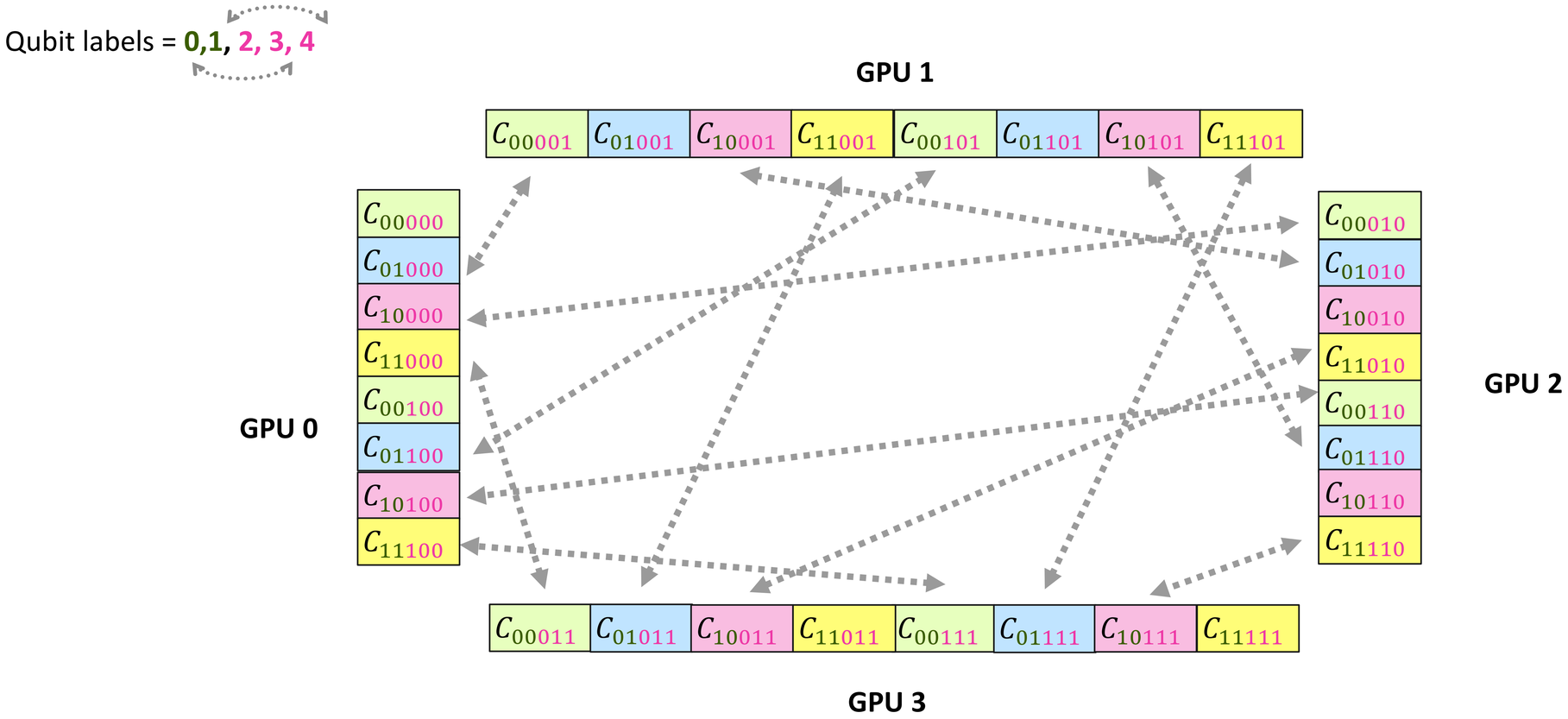
- Better approach: Qubit swap and relabeling -- no need to exchange back:
  1. Swap the global qubit with a local one:
     a) Exchange amplitudes
     b) Change the labels of qubits.
  2. Apply the gate to the now local qubit.
- **Example:** Applying Hadamard gate to fourth qubit (qubit 3)



Multi-GPU State Vector Simulation

Multi-Qubit Swaps

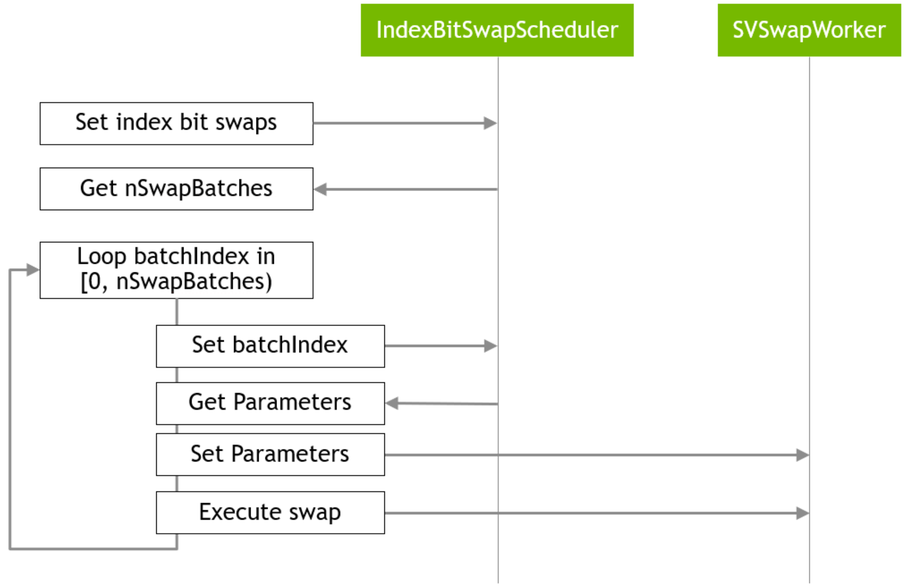
- Multiple qubits can be exchanged at once
  - Divide GPUs into groups of 2^k participants.
  - Each GPU exchanges $1/2^k$ of its amplitudes with each of the (2^k-1) GPUs in the group.
- **Example:** Simultaneous swap of global qubits 4,3 with local qubits 0,1



Multi-GPU State Vector Simulation

Support in Nvidia quantum libraries

- **cuStateVec** library contains the API for distributed qubit swap, which is used to distribute state vector simulations over multiple devices and nodes.
  - It provides optimized API for for intranode communication with CUDA IPC allowing GPUDirect P2P.
  - It also provides more general API using CUDA-aware MPI that works for both intranode and internode communication.
- Nvidia recently released **cuStateVec Ex** API to facilitate further the development of state vector simulators:
  - Advanced capabilities for managing qubit ordering and its permutation.
  - Encapsulates simulator pipeline to accelerate state vector updates.
  - Gate matrix fusion.
- **CUDA-Q** builds on top of cuQuantum and provide an optimized multi-node multi-GPU backend for state vector simulations:
  - Given network topology, CUDA-Q automatically uses the fastest available communication path: GPUDirect P2P for intranode communication and MPI for internode communications.
  - Group gates to minimize the necessary qubit swaps and reduce communication overhead.

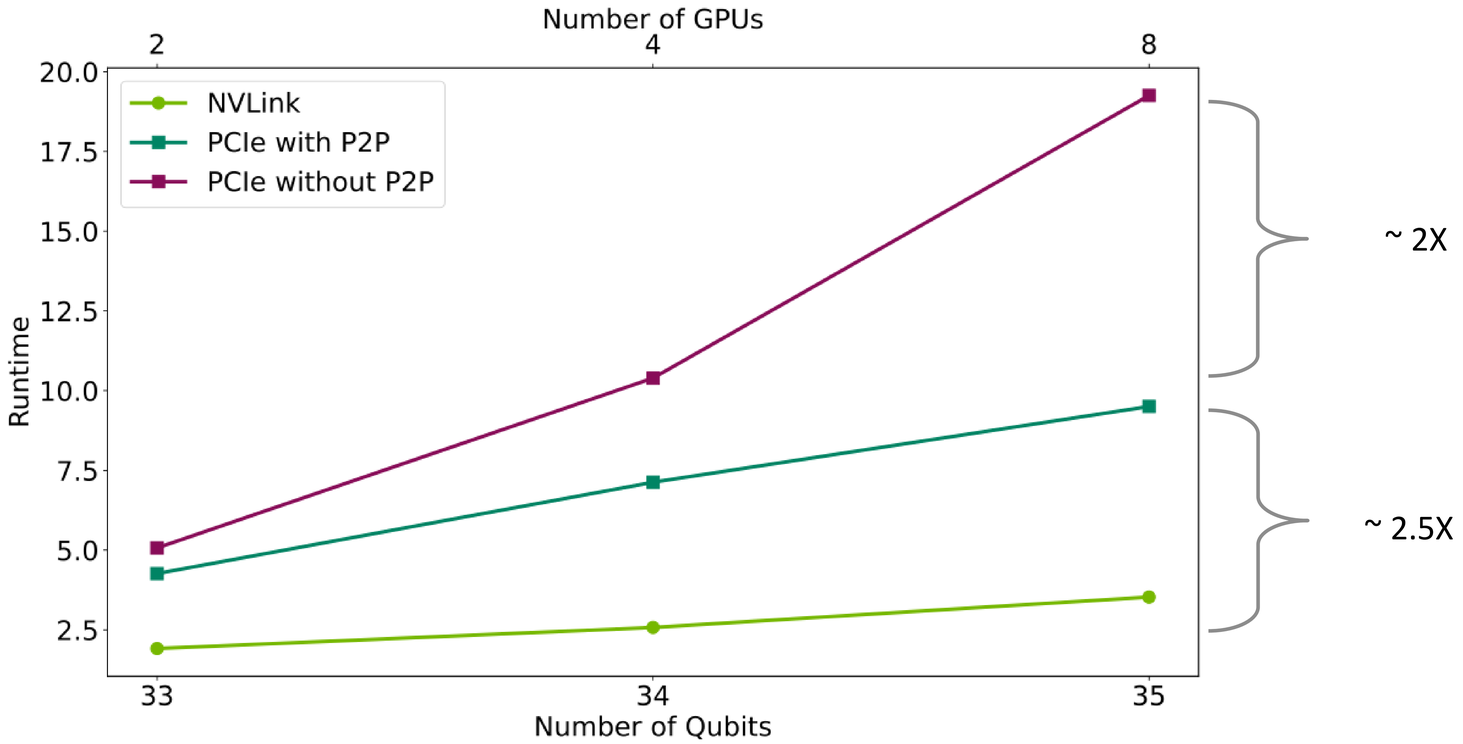


The main loop for distributed index swap in cuStateVec

Impact of Interconnect on MG Simulations

Intranode: PCIe vs NVLink

- For multi-GPU quantum simulation, the bandwidth of the interconnect between GPUs play a crucial role in the performance.
- For intranode communication NVLink between GPUs provide much higher bandwidth than PCIe.
- **Example:** Quantum Phase Estimation (FP64) running on a DGX H100 node.



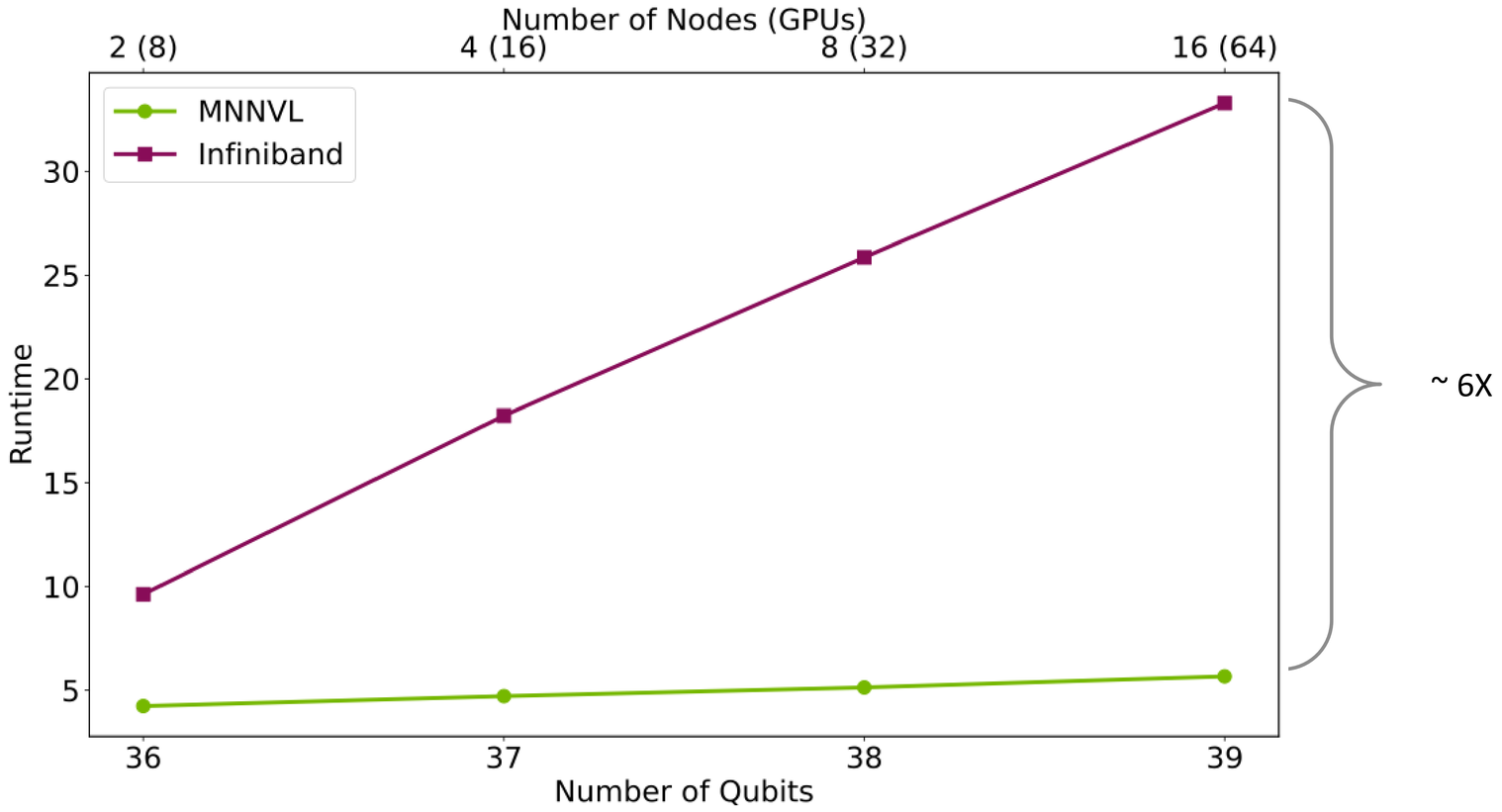
* PCIe with P2P: communication is done via NICs (`UCX_TLS=rc,self`).

** PCIe without P2P: communication is staged via CPU buffers by disabling CUDA IPC (`UCX_TLS=cuda_copy,sm,self`)

Impact of Interconnect on MG Simulations

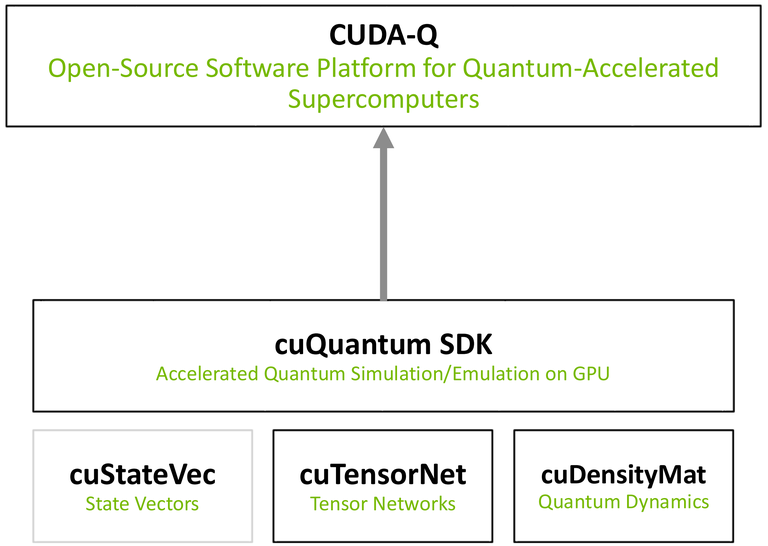
Internode: Infiniband vs. MNNVL

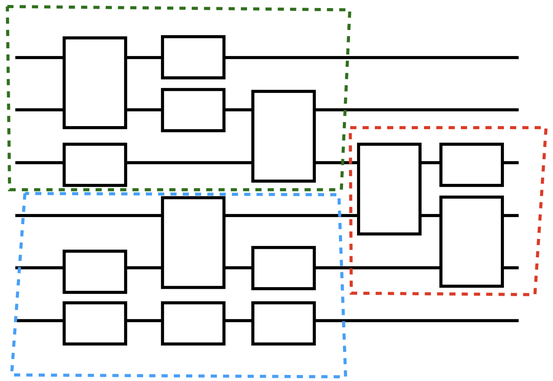
- Multi-node NVLink provide superior performance compared to Infiniband networks for multinode state vector simulations.
- **Example:** Quantum Phase Estimation (FP64) running on GB200 NVL72

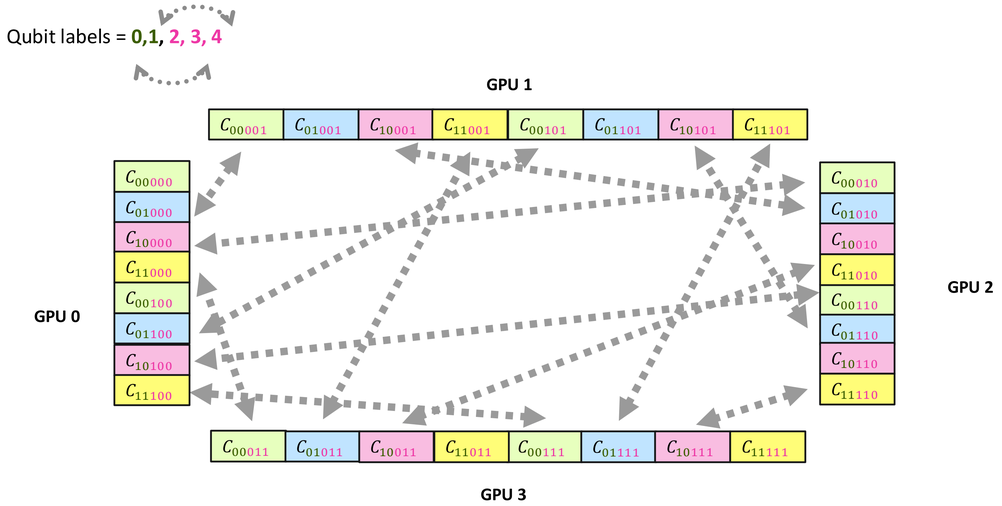


Summary

- State vector simulation are memory-bound.
- Gate fusion allows utilizing the full power of GPUs by increasing arithmetic intensity and reducing the number of gate applications.
- Single-GPU state vector simulations provide up to 80x speed up over single-CPU simulations.
- Multi-GPU are necessary for simulating circuits with high qubit counts.
- Qubit swaps and relabeling allow extending single-GPU simulators in multi-GPU ones efficiently.
- Communication speed between GPUs is critical for the performance of multi-GPU simulations.
- cuStateVec provide highly-optimized API for circuit simulation library developers.
- CUDA-Q provides an easy-to-use and performant state vector simulators for end users.







Thanks for your attention

Questions?