qTask: Task-parallel Quantum Circuit Simulation with Incrementality

Dr. Tsung-Wei (TW) Huang, Assistant Professor
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT
https://tsung-wei-huang.github.io/

Today's Quantum is in a Noisy World

- **We are at the *noisy intermediate-scale quantum (NISQ)* era**
  - Quantum processors/states are very sensitive to the environment
  - Noise is likely to exist anyway… can we still do something meaningful?
- **Quantum computing is fundamentally different from classical**

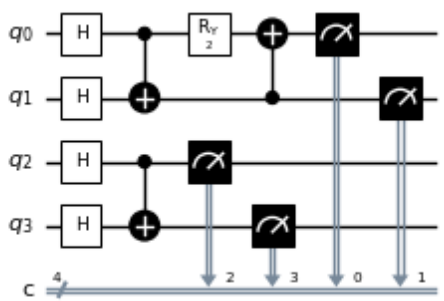
Logic element	Bit: classical bit (transistor)	Qubit: quantum bit (any coherent two-level system)
State	0 "**or**" 1	|0> "**AND**" |1>

Measurement	• **Discrete** states
	• **Deterministic** measurement
	  • 1 (high voltage) is 1
	  • 0 (low voltage) is 0
		• **Superposition** states
		• **Probabilistic** measurement
		  • Ex: 50% |0> and 50% |1>
		  • Last only for a few seconds

Is Quantum Computing Not Possible?

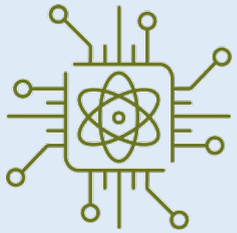
- **Don’t know, but it needs large classical software support!**

Quantum program

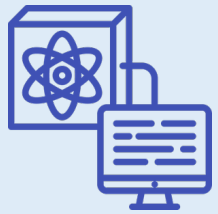


Classical support for quantum computing

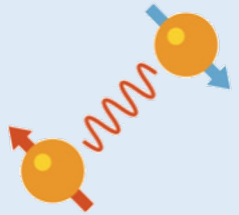
Simulation



Optimization



Transpilation

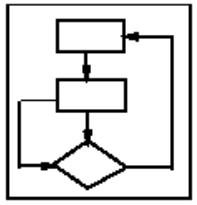


Visualization

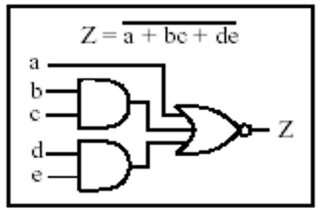
Gate-based Quantum computer

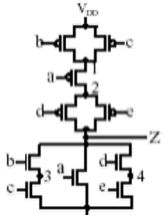
Analogous to the classical design flow

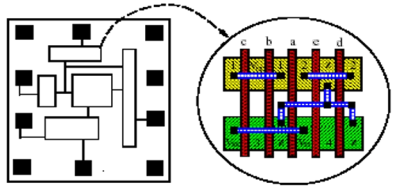
System spec → VHDL Design → RTL Synthesis → Circuits → Physical Design → Fabrication

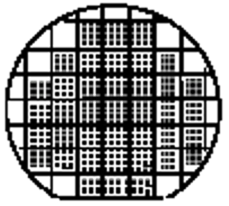


```
module fz(a,b,c,d,e,Z);
input a,b,c,d;
output Z;
assign Z = ~(a|(b&C));
endmodule
```



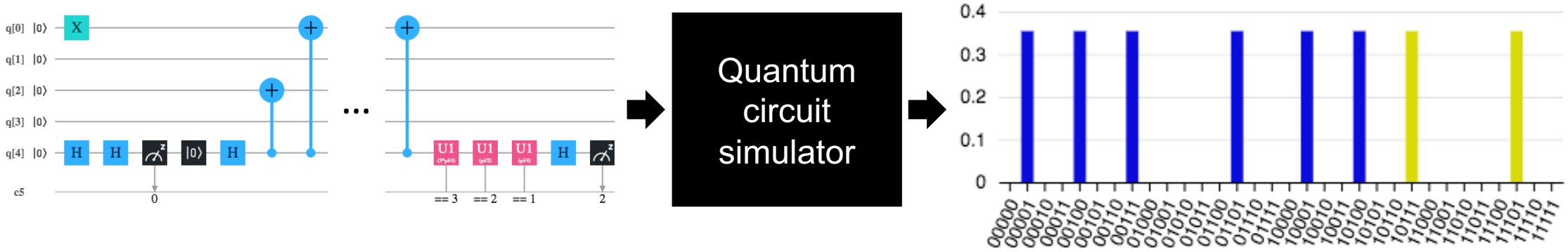






Classical Quantum Circuit Simulation

- **A critical component in the design flow of quantum computers**
  - Allow us to understand how quantum operations work without access to expensive quantum computers

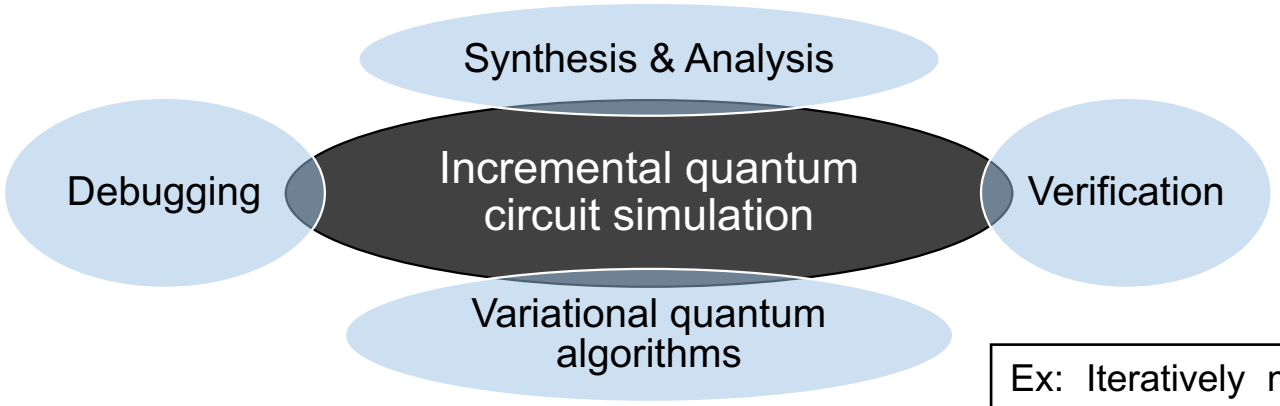


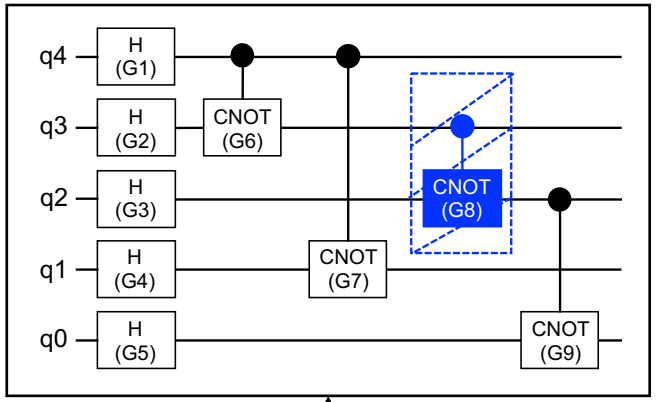
Quantum circuit for Shor's algorithm (Ex: **5** input qubits) → What is the output? (ideal & noisy) → Output state amplitude distribution (Ex: **2^5 = 32** output states)

Superposition

Why is Quantum Simulation Challenging?

- **Quantum simulation demands VERY LARGE computation**
  - # output states is exponentially proportional to the # qubits (*superposition*)
    - Ex: *n* qubits give 2^n states
  - Parallelism cannot be easily described out of the structure (*entanglement*)
- **Incrementality has emerged as an important tool for NISQ apps**
  - Unfortunately, largely ignored by existing simulators

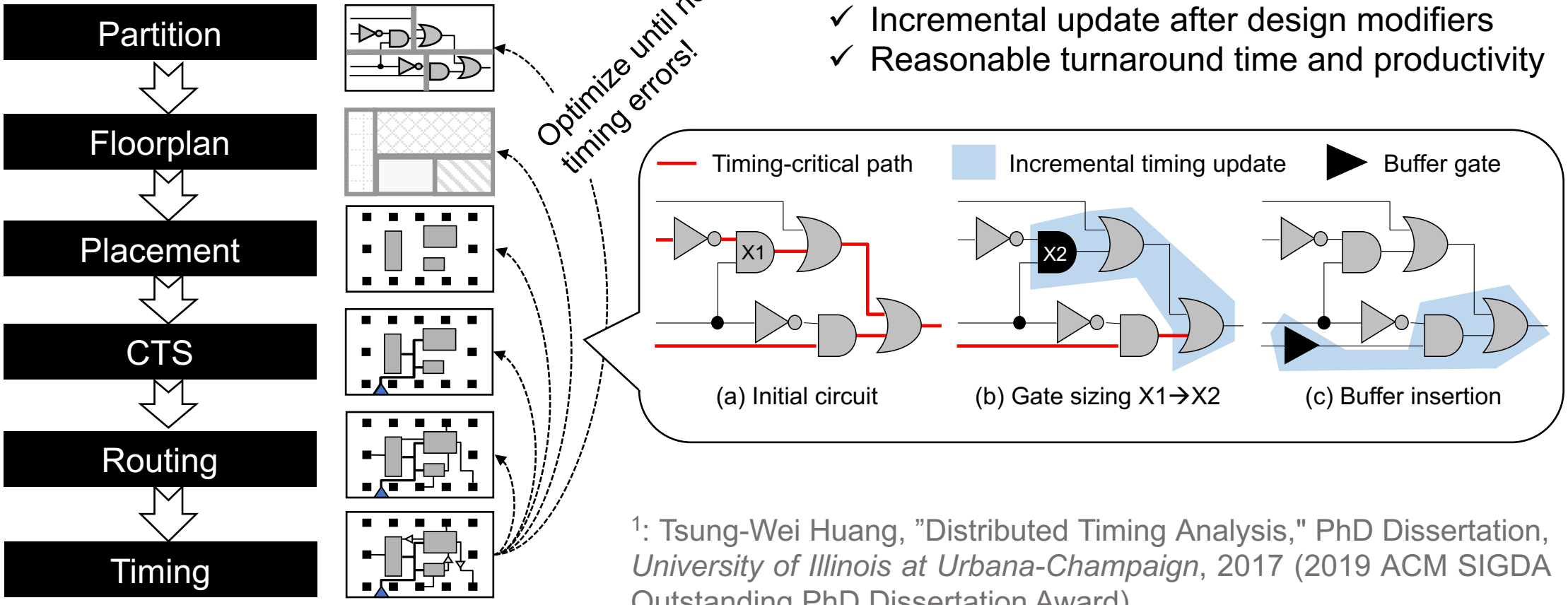




Ex: Iteratively modify a circuit and re-simulate state amplitudes ***incrementally*** for reasonable turnaround time

Incrementality in Classical Design Flow[1]

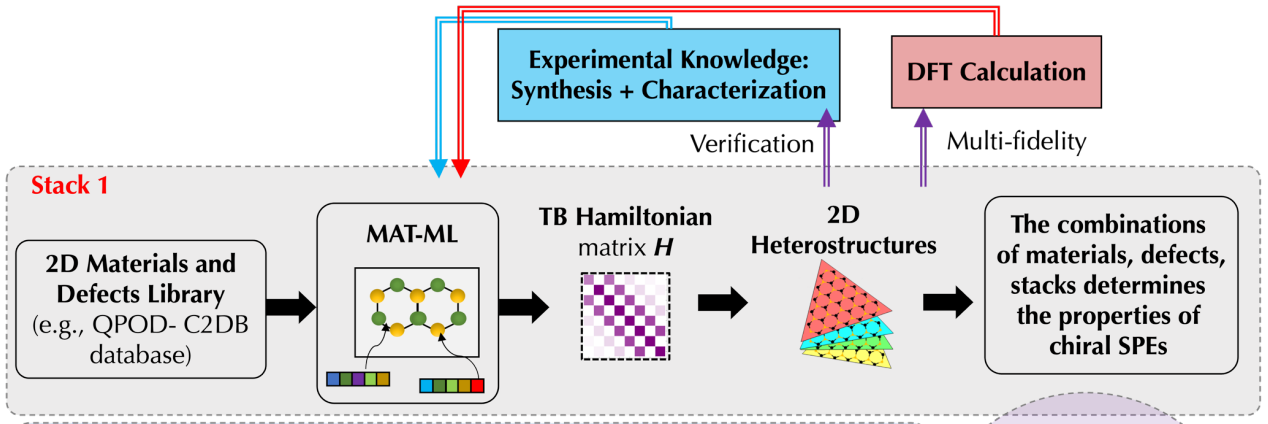
- **Incrementality is critically important for circuit optimization**

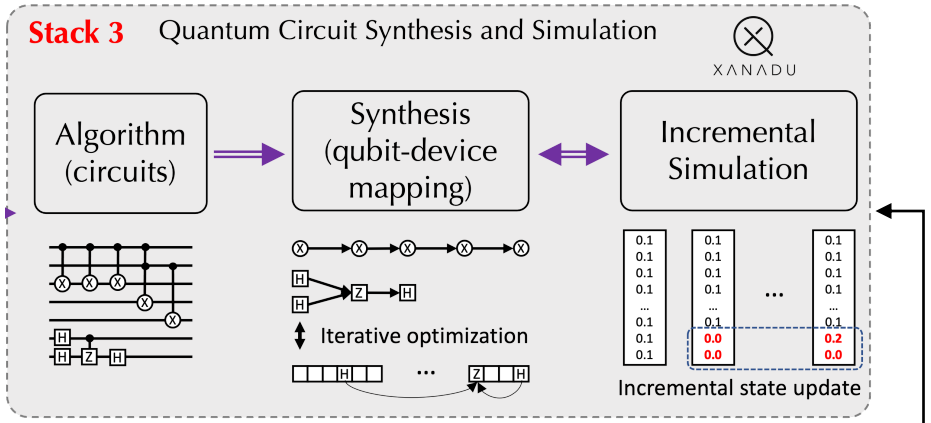


[1]: Tsung-Wei Huang, ”Distributed Timing Analysis," PhD Dissertation, *University of Illinois at Urbana-Champaign*, 2017 (2019 ACM SIGDA Outstanding PhD Dissertation Award)

Incrementality in our NSF FuSe Project[1]

- **Quantum software-hardware co-design on photonic devices**





Materials: Condensed Matter Theory	Materials: Condensed Matter Experiments	Photonic Devices	Photonic Circuits	Workforce Development
Perebeinos, Rajan	Kono, Hone, S. Huang	Gao	T. Huang	Rasmussen

RICE · COLUMBIA UNIVERSITY · THE UNIVERSITY OF UTAH · UB · XANADU · AIM photonics

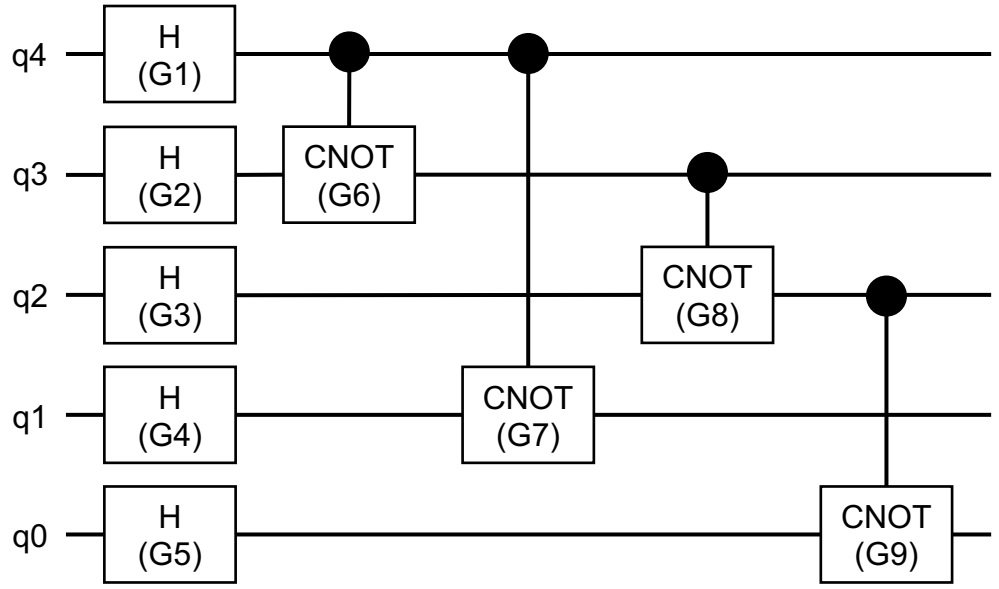
[1]: Co-PI, "FuSe-TG: Co-Design of Chiral Quantum Photonic Devices and Circuits Integrated with 2D Material Heterostructures," $400K, NSF FuSe, 2023

qTask: Incremental Quantum Simulator

- **C++ Programming model**

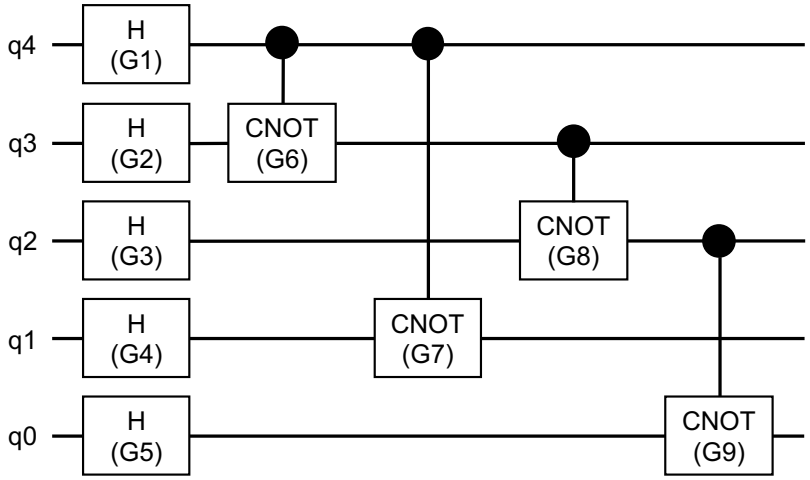
```
qTask ckt(5);
auto [q4, q3, q2, q1, q0] = ckt.qubits();
// create five nets and eight gates
auto net1 = ckt.insert_net(ckt.nets().begin());
auto net2 = ckt.insert_net(net1);
auto net3 = ckt.insert_net(net2);
auto net4 = ckt.insert_net(net3);
auto net5 = ckt.insert_net(net4);
auto G1 = ckt.insert_gate(H, net1, q4);
auto G2 = ckt.insert_gate(H, net1, q3);
auto G3 = ckt.insert_gate(H, net1, q2);
auto G4 = ckt.insert_gate(H, net1, q1);
auto G5 = ckt.insert_gate(H, net1, q0);
auto G6 = ckt.insert_gate(CNOT, net2, q3, q4);
auto G7 = ckt.insert_gate(CNOT, net3, q1, q4);
auto G8 = ckt.insert_gate(CNOT, net4, q2, q3);
auto G9 = ckt.insert_gate(CNOT, net5, q0, q2);
ckt.dump_graph(std::cout);
ckt.update_state();  // full update
// modify the circuit
ckt.remove_gate(G8);
auto G10 = ckt.insert_gate(CNOT, net4, q1, q2);
ckt.update_state();  // incremental update
```

Method	Functionality
insert_net	insert a new empty net to the circuit
remove_net	remove a net and all its gates from the circuit
insert_gate	insert a net gate to an existing net
remove_gate	remove a gate from its net and the circuit
update_state	update the state value, incrementally
dump_graph	dumps the current partition graph

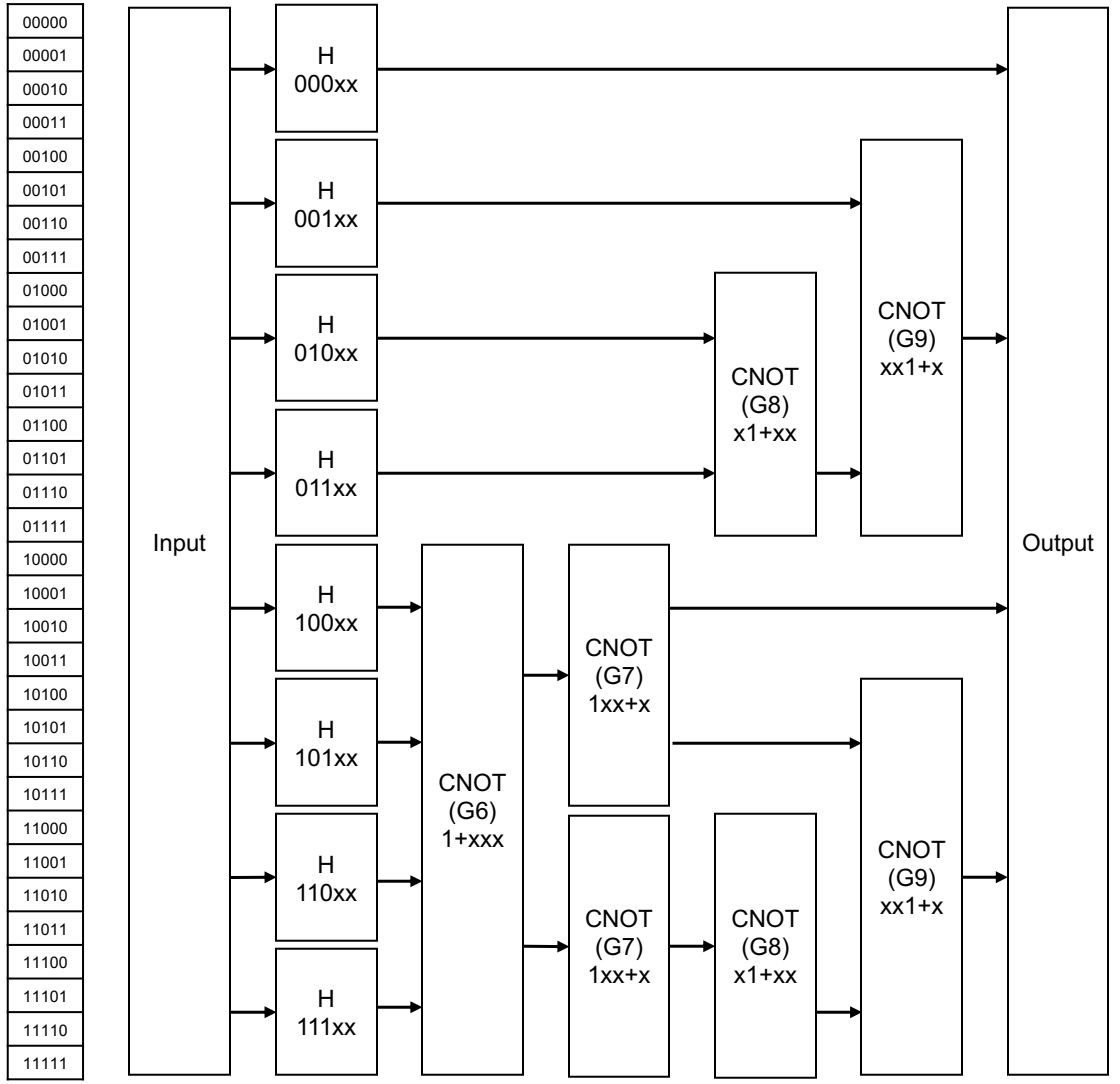


Task-parallel Partitioning Strategy

- Divide state vector into equal-size disjoint blocks
- Group consecutive blocks to form partitions
- Each partition spawns many tasks to perform gate operations



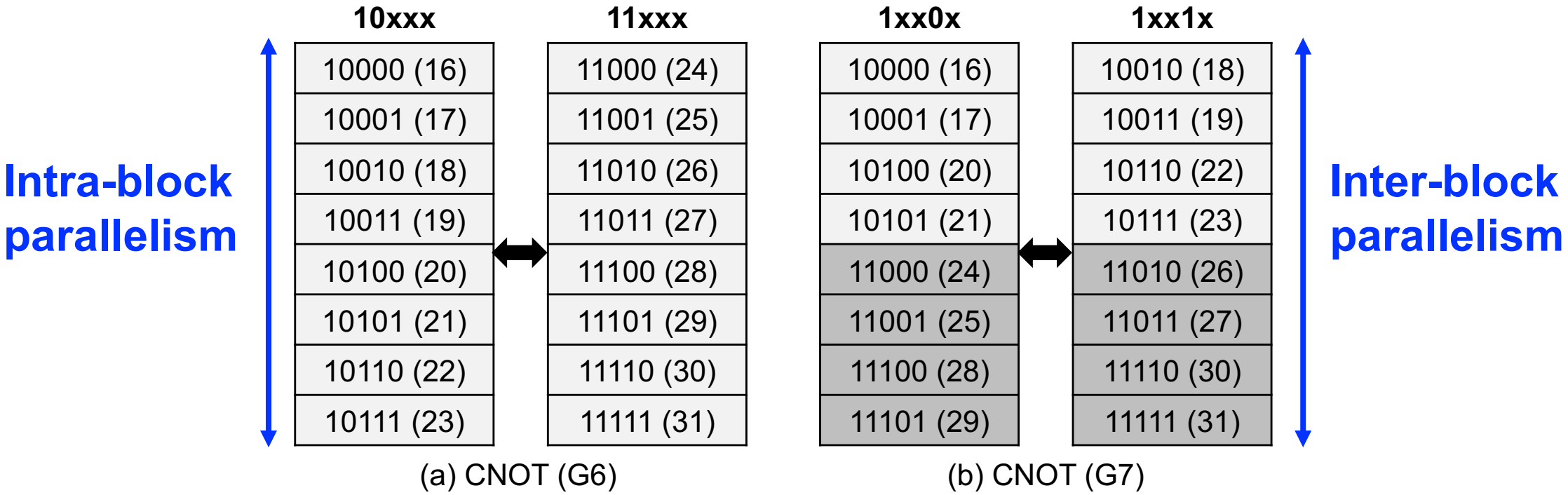
Block Size = 4



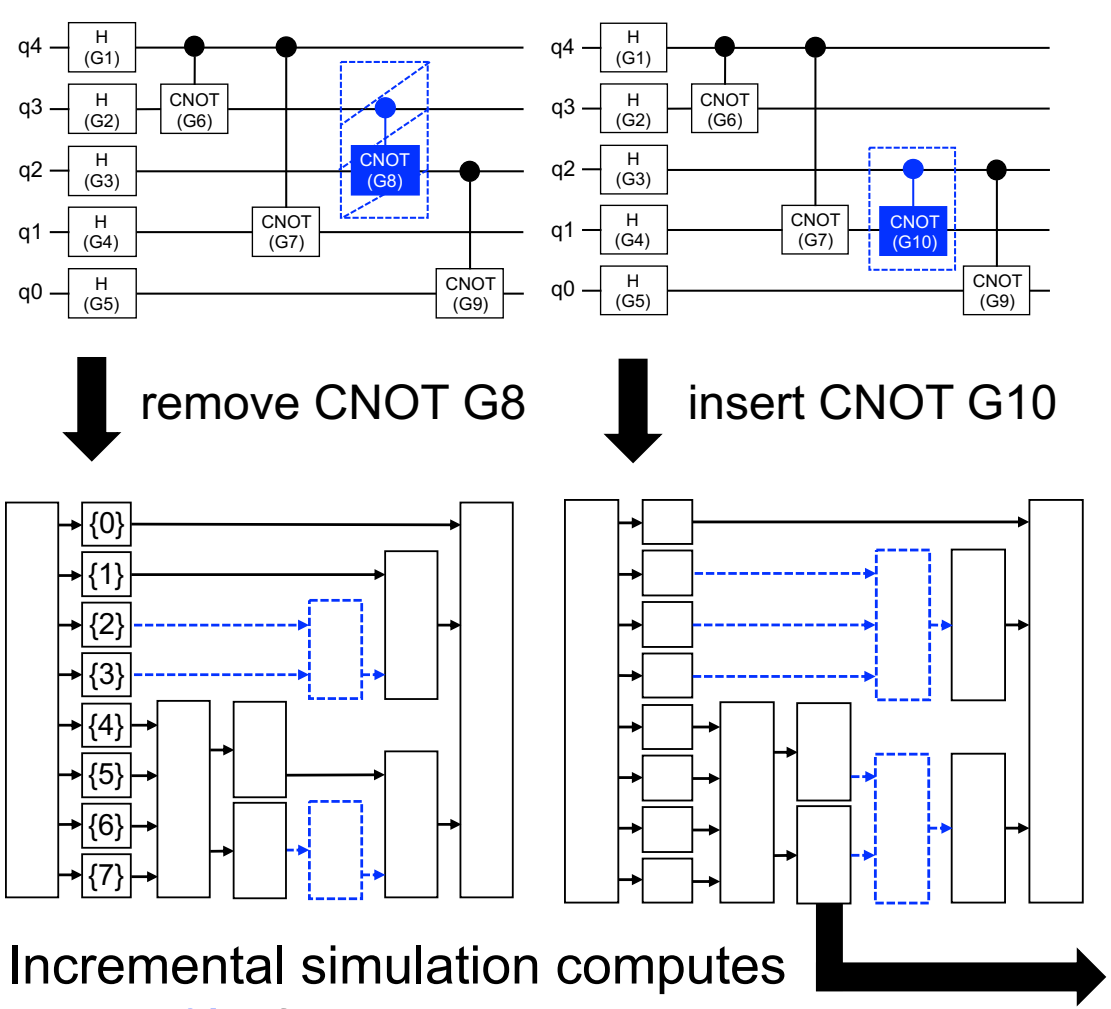
Example: Group Blocks to Partitions

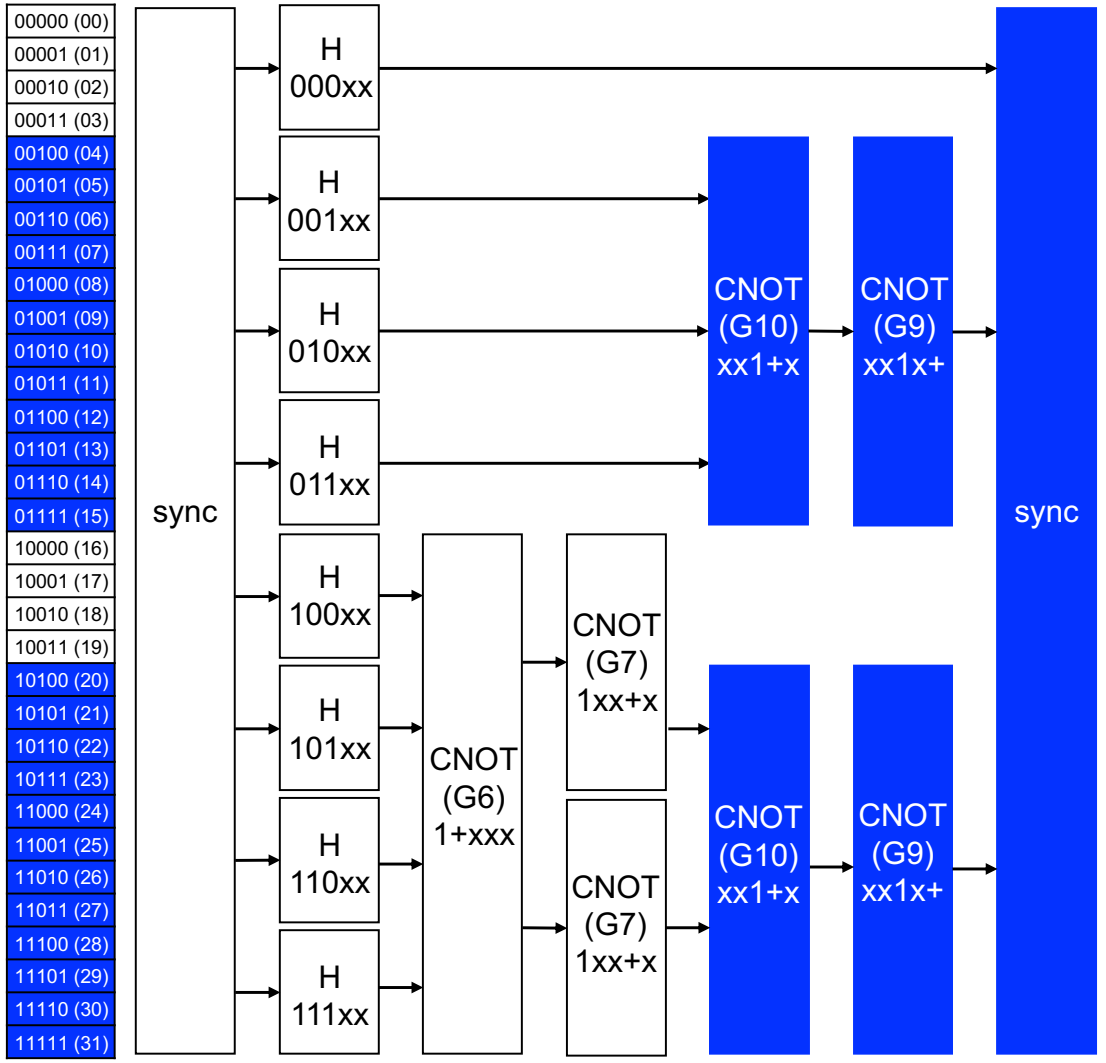
(a) CNOT gate G6 forms a partition of four blocks

(b) CNOT gate G7 forms two partitions each of two blocks

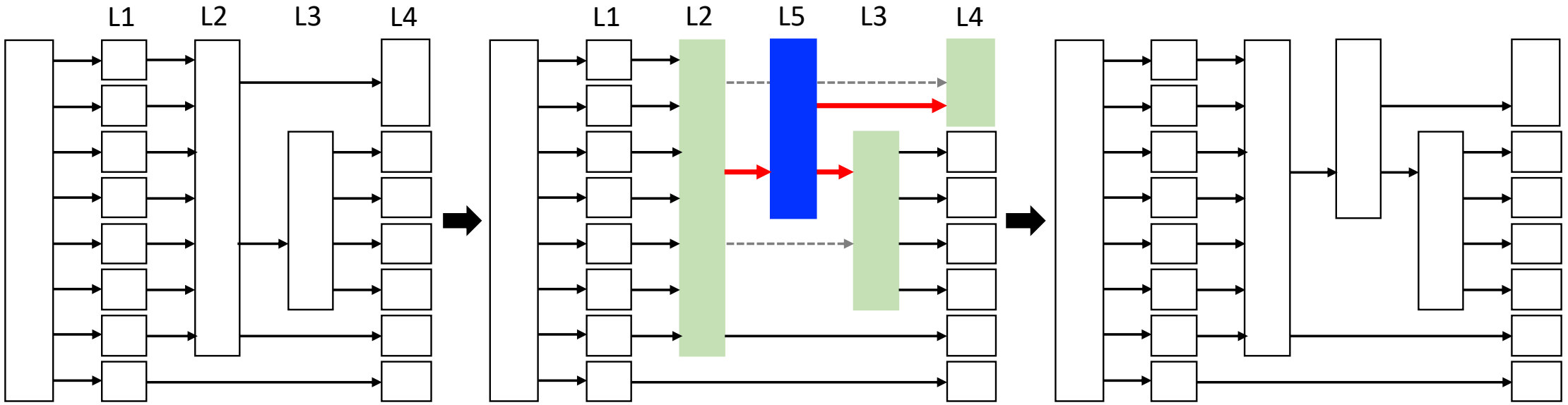


Partitions to Update for Circuit Modifiers





Reconnect Partitions after Circuit Modifiers

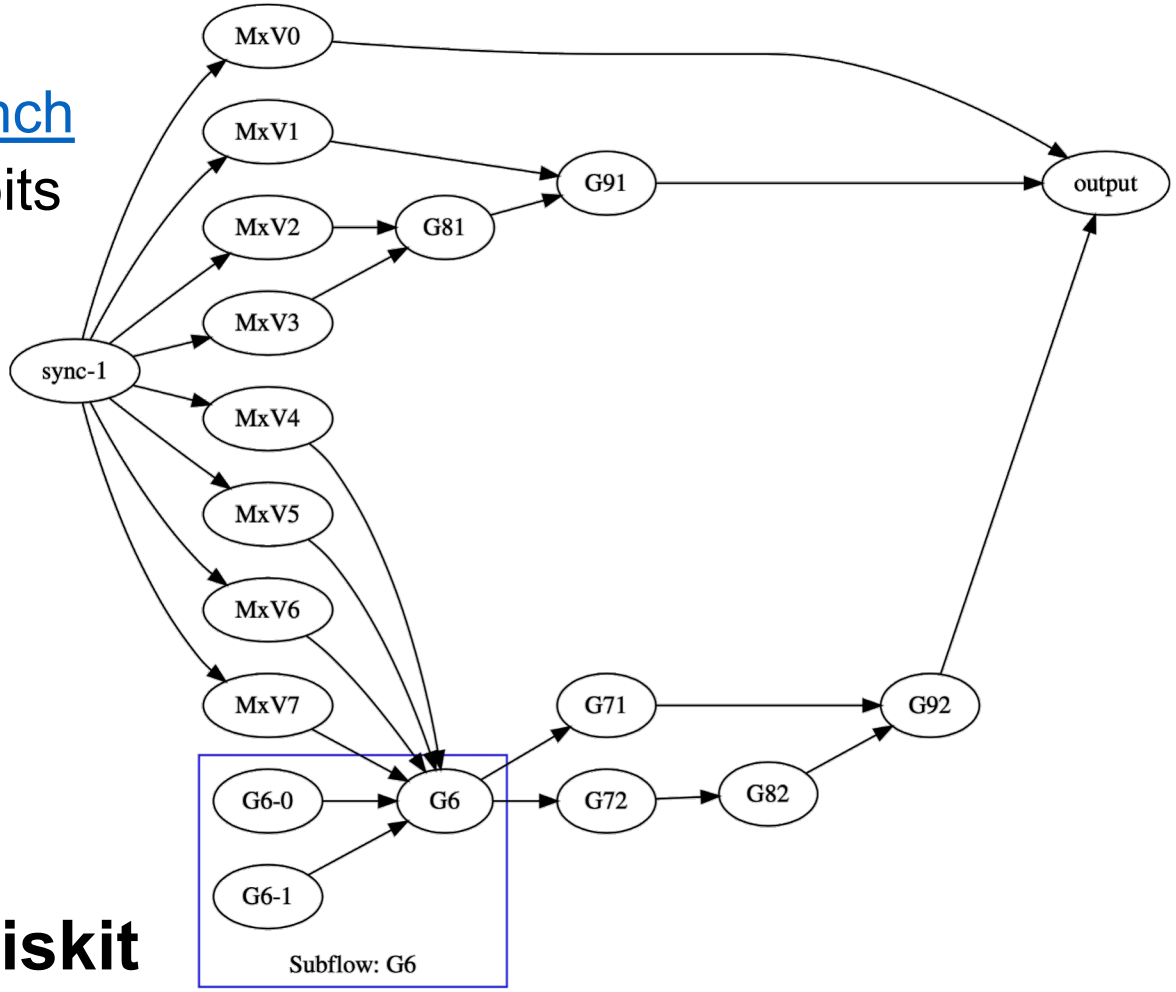


	Partitions
L1	{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}
L2	{0, 1, 2, 3, 4, 5, 6}
L3	{2, 3, 4, 5}
L4	{0, 1}, {2}, {3}, {4}, {5}, {6}, {7}

	Partitions	Overlap →	Uncovered Partitions
L1	{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}		
L2	{0, 1, 2, 3, 4, 5, 6}	{0, 1, 2, 3, 4, 5, 6}	{}
L5	{0, 1, 2, 3}		{0, 1, 2, 3}
L3	{2, 3, 4, 5}	{2, 3, 4, 5}	{0, 1}
L4	{0, 1}, {2}, {3}, {4}, {5}, {6}, {7}	{0, 1}	{}

Experiment Set-up

- **Evaluated on QASMBench**
  - https://github.com/pnnl/QASMBench
  - Medium-size circuits up to 26 qubits
- **Implemented using Taskflow**
  - https://taskflow.github.io/
  - An example taskflow on the right
- **Tested on a CentOS server**
  - 16 Intel i7 cores at 2.50 GHz
  - 128 GB RAM
  - Compiled with clang++ v12
  - Enabled –std=c++17 and –O3
- **Compared with Qulacs and Qiskit**



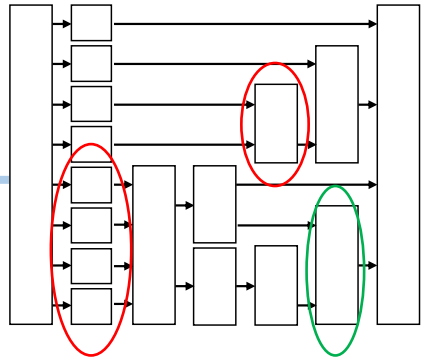
Experimental Results

- **6x** faster than Qulacs and **10x** faster than Qiskit with incrementality

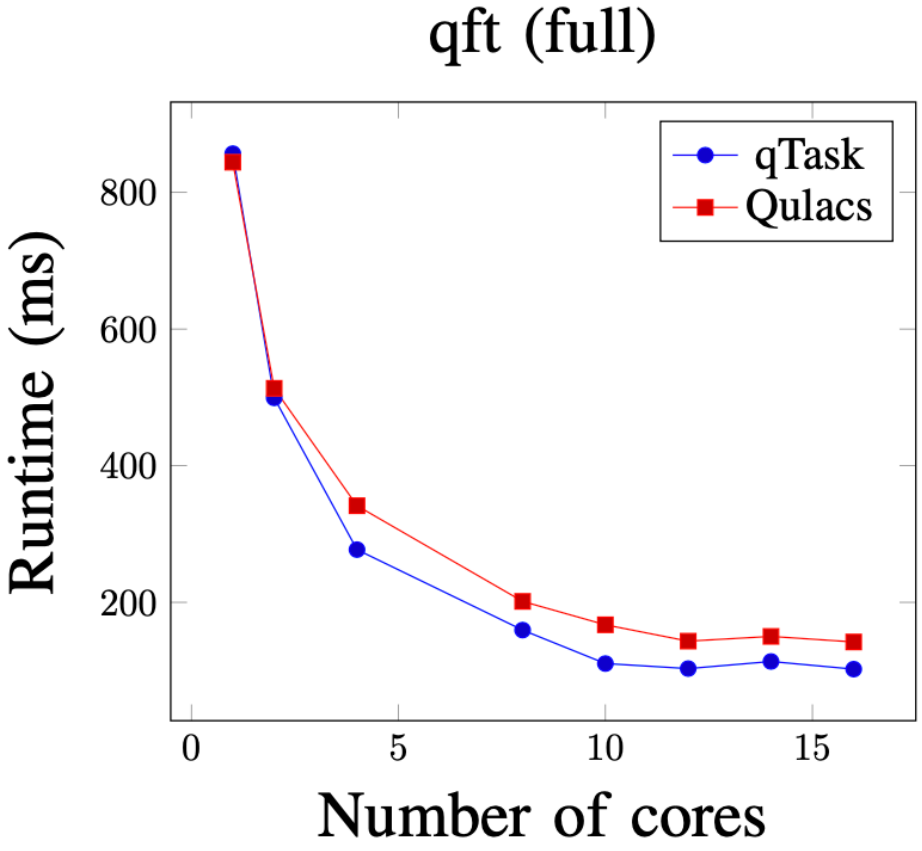
Circuit	Description	Qubits	Gates	CNOT	Qulacs			Qiskit			qTask		
					full (ms)	inc (ms)	mem (GB)	full (ms)	inc (ms)	mem (GB)	full (ms)	inc (ms)	mem (GB)
dnn	Quantum deep neural network	8	1200	384	21.8	2167.8	0.07	51.4	5114.3	0.07	22.4	529.3	0.09
adder	Quantum ripple adder	10	142	65	17.2	186.4	0.05	29.5	320.1	0.04	11.79	57.9	0.06
bb84	Quantum key distribution	8	27	0	1.1	2.3	0.03	1.1	2.4	0.03	1.5	1.9	0.04
bv	Berstein-Vazirani algorithm	14	41	13	9.0	21.7	0.11	16.7	40.6	0.12	6.7	14.3	0.13
ising	Ising model simulation	10	480	90	49.6	1438.1	0.08	81.4	2360.1	0.09	41.7	550.14	0.10
multiplier	Quantum multiplication	15	574	246	150.9	4199.0	1.98	283.7	7896.3	2.86	101.62	1052.6	3.46
multiplier_35	3×5 matrix multiplication	13	98	40	22.4	130.1	0.10	47.1	273.54	0.15	16.01	92.7	0.18
qaoa	Approximation optimization	6	270	54	5.4	148.5	0.01	13.4	368.5	0.01	6.1	37.65	0.02
qf21	Quantum factorization of 21	15	311	115	79.8	1173.1	1.59	191.5	2815.1	1.66	58.3	480.7	1.91
qft	Quantum Fourier transform	15	540	210	142.0	3621.0	2.75	281.2	7170.1	3.11	102.2	949.4	3.17
qpe	Quantum phase estimation	9	123	43	10.3	100.42	0.02	27.8	270.4	0.04	7.65	80.44	0.05
sat	Boolean satisfiability solver	11	679	252	85.5	3660.7	0.11	196.7	8422.1	0.21	62.3	786.5	0.28
seca	Shor's algorithm	11	216	84	28.4	401.0	0.06	59.64	843.0	0.09	21.42	128.5	0.11
simons	Simon's algorithm	6	44	14	0.83	3.9	0.03	1.44	6.71	0.03	0.81	2.44	0.04
vqe_uccsd	Variational quantum eigensolver	8	10808	5488	244.4	249084.2	0.36	435.1	443367.1	0.56	259.4	44251.1	0.76
big_adder	Quantum ripple adder	18	284	130	200.1	2401.3	7.98	360.4	4300.8	11.4	137.9	602.5	13.9
big_bv	Bernstein-Vazirani algorithm	19	56	18	125.0	305.9	2.6	234.5	573.9	3.9	95.4	126.6	4.9
big_cc	Counterfeit coin finding	18	34	17	24.9	47.8	0.98	42.3	63.3	1.5	16.6	24.5	1.7
big_ising	Ising model simulation	26	280	50	1939.1	3345.5	89.4	1745.3	2866.2	91.4	991.4	2000.3	114.3
big_qft	Quantum Fourier transform	20	970	380	2936.3	100567.0	67.3	3012.6	144453.4	77.6	2209.7	12912.8	91.2
					1.46	**5.77**	**0.74**	**1.71**	**9.76**	**0.82**	**1.00**	**1.00**	**1.00**

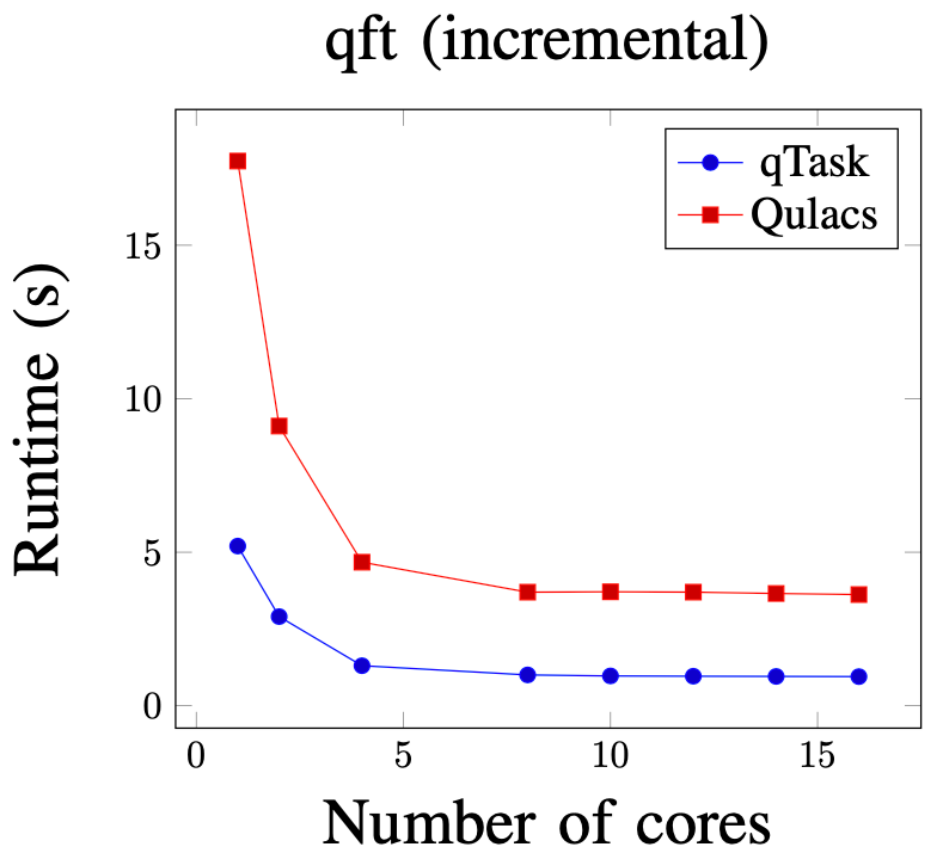
Qubits: number of qubits **Gates**: number of standard gates **CNOT**: number of CNOT gates to entangle and disentangle states
full: runtime of full simulation **inc**: runtime of incremental simulation **mem**: maximum resident set size (RSS)

Full and Incremental Simulations



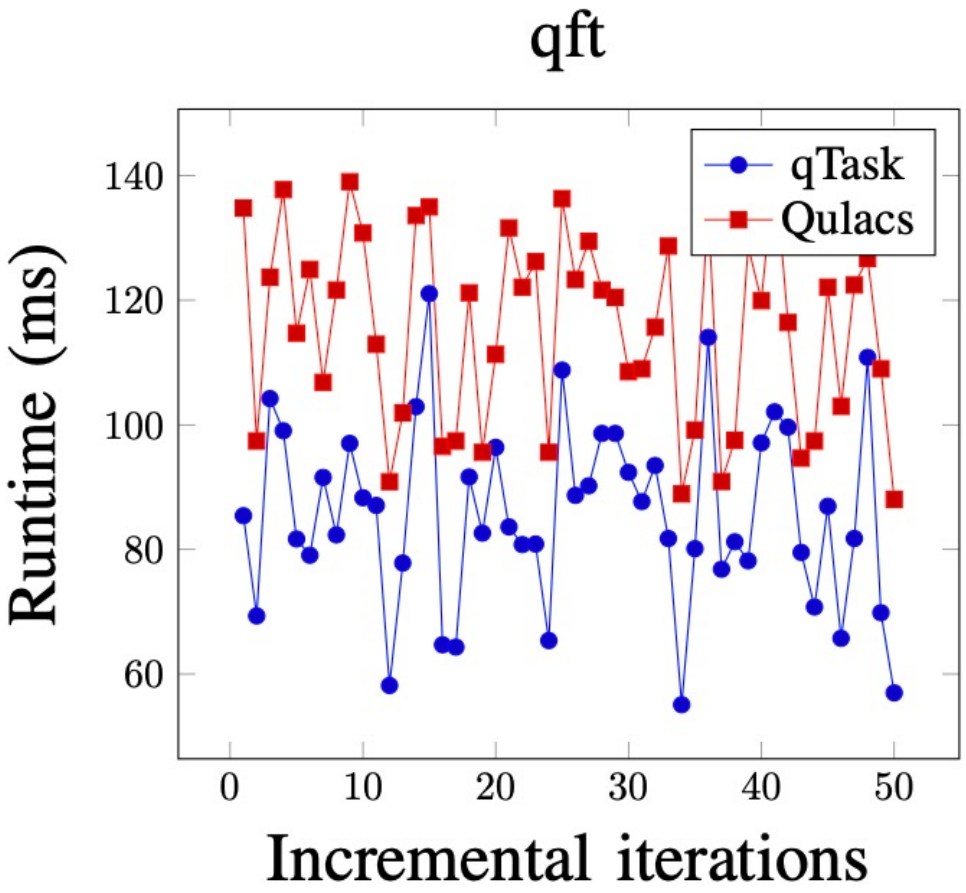
- qTask explores both **inter-** and **intra-**gate parallelism

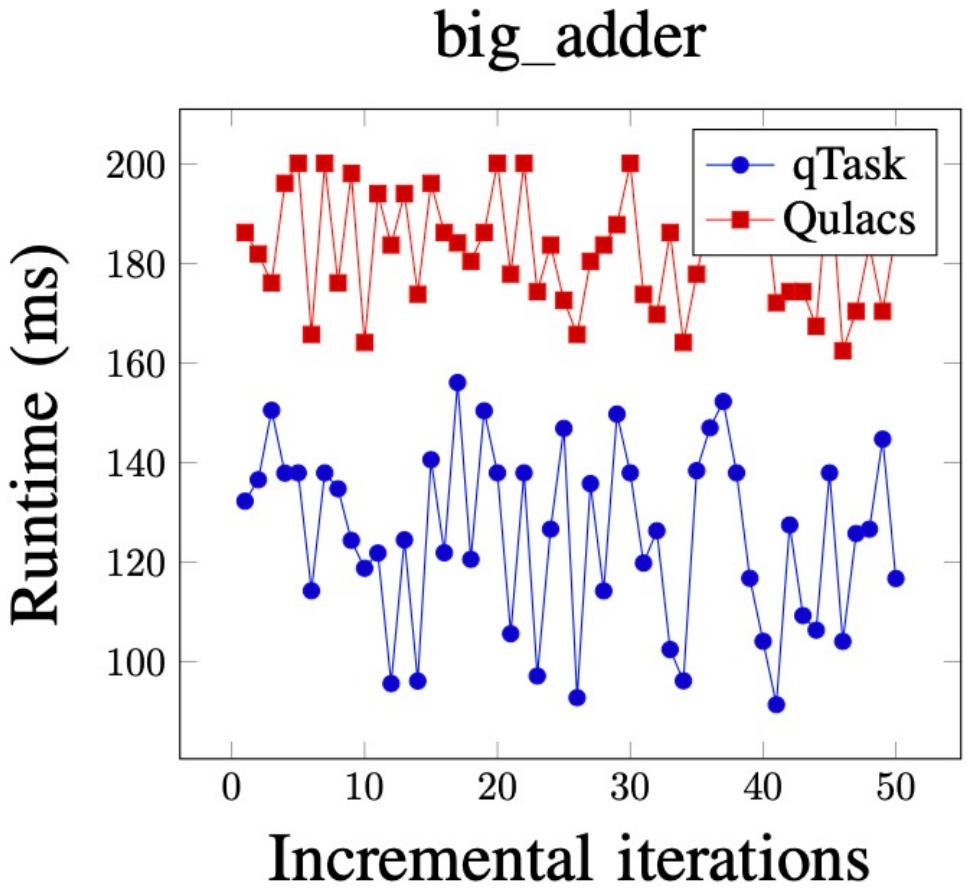




Runtime vs Incremental Iterations

- Random gate insertion and removal





Conclusion

- **Identified the need for incremental quantum circuit simulator**
  - When a quantum circuit starts to change, how we we quickly re-simulate the state results without starting from scratch?
- **Introduced a new incremental simulator called qTask**
  - A new C++-based programming model
  - A new task-parallel partitioning strategy
  - A new incremental update method for partitions and state vectors
- **Presented the simulation performance of qTask**
  - 6—10x faster than the state-of-the-art simulators