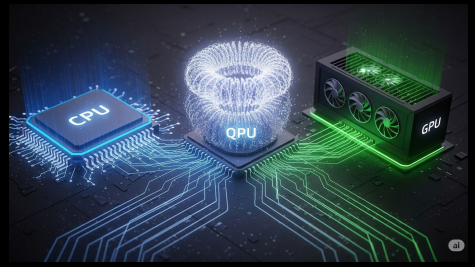


NVIDIA iQuHACK 2026 - LABS Challenge

Quantum-Enhanced Optimization with GPU Acceleration

Name: Ayush

Track: NVIDIA LABS

Hardware: NVIDIA L4 GPU (Brev)

Stack: CUDA-Q, CuPy, NumPy

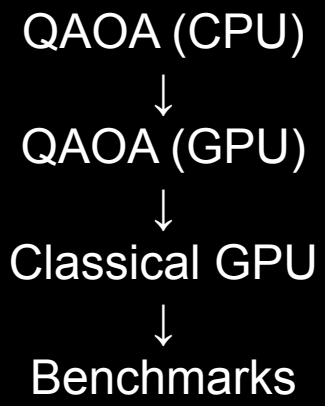
THE PROBLEM (LABS)

- LABS (Low Autocorrelation Binary Sequences) is a hard combinatorial optimization problem
- Goal: minimize autocorrelation energy of a binary sequence
- Brute force scales exponentially with problem size N
- Classical heuristics exist but become slow for large N

THE PLAN (ORIGINAL SOLUTION)

Original Plan

- Use QAOA to explore LABS energy landscape
- Validate correctness on CPU backend
- Migrate to GPU for acceleration
- Benchmark CPU vs GPU performance



CPU VALIDATION & VERIFICATION (STEP A)

- Implemented QAOA using CUDA-Q on CPU backend
- Validated results for small N (N = 3-10)
- Ensured correctness before scaling
- Built a rigorous test suite to catch logic and physics errors

Tests Included

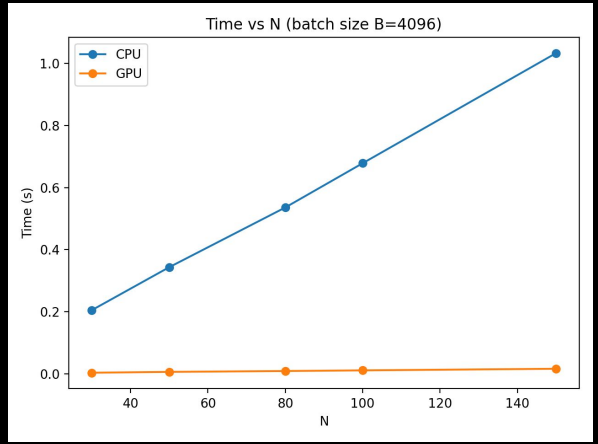
- Energy matches reference
- Spin-flip symmetry
- Energy is non-negative integer
- Brute-force consistency
- QAOA smoke test

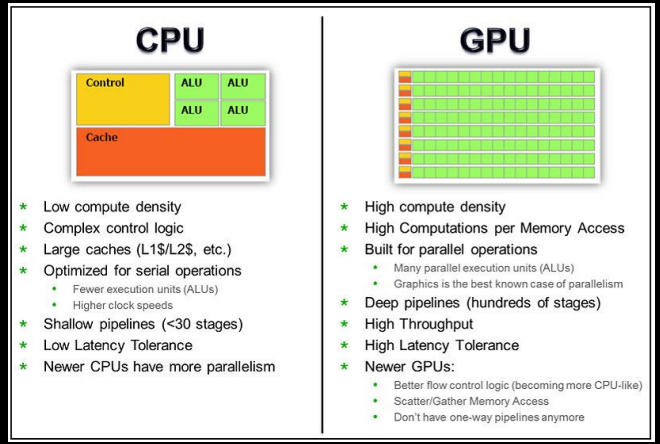
```
cudaq@33df3a216635:~/2026-NVIDIA/team-submissions$ python tests.py
[PASS] LABS energy matches reference
[PASS] Spin-flip symmetry E(s)=E(-s)
[PASS] Energy is nonnegative integer
[PASS] Brute force best is consistent (small N)
[PASS] QAOA smoke test (small N)

Summary: 5/5 tests passed
```

THE PIVOT (WHY GPU WAS NEEDED)

- CPU simulation becomes infeasible as N increases
- QAOA circuit simulation is expensive
- Classical neighbor evaluation dominates runtime
- Needed GPU acceleration for both:
  - Quantum simulation
  - Classical optimization loop





GPU QUANTUM ACCELERATION (STEP B)

- Migrated QAOA to NVIDIA GPU backend using CUDA-Q
- Used cuStateVec-based simulation
- Verified GPU results match CPU energies
- Successfully scaled to larger N (N = 8-12)

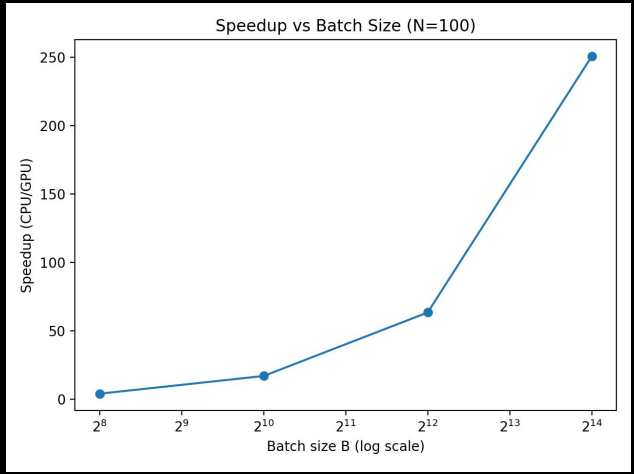
```
N=3  | QAOA E=1  | Brute E=1  | params=(0.3, 0.3)
N=4  | QAOA E=2  | Brute E=2  | params=(0.3, 0.3)
N=5  | QAOA E=2  | Brute E=2  | params=(0.3, 0.3)
N=6  | QAOA E=7  | Brute E=7  | params=(0.3, 0.3)
N=8  | best found E=8  | params=(0.2, 0.2)
N=10 | best found E=13 | params=(0.2, 0.2)
N=12 | best found E=10 | params=(0.2, 0.2)
```

```python
if __name__ == "__main__":
    print(f"CUDA-Q version: {getattr(cudaq, '__version__', 'unknown')}")

    # On Brev GPU images this should work
    try_set_target("nvidia")
```

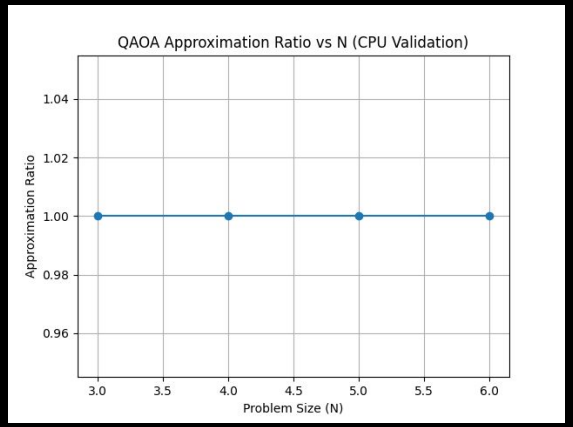
GPU CLASSICAL ACCELERATION (STEP C)

- Classical MTS algorithm rewritten using CuPy
- Batched neighbor energy evaluations on GPU
- Implemented GPU-accelerated local search
- Compared CPU vs GPU runtime directly



Up to **~250× speedup**

RESULTS SUMMARY (METRICS)



Key Results

- Approximation ratio ≈ 1.0 for small N
- GPU drastically reduces time-to-solution
- Classical GPU acceleration provides largest speedup
- Fully GPU-accelerated workflow achieved

N	B	CPU (s)	GPU (s)	speedup
30	256	0.012288	0.003139	3.91
30	1024	0.050604	0.003211	15.76
30	4096	0.204785	0.003138	65.25
30	16384	0.784969	0.004023	195.12
50	256	0.031048	0.005176	6.00
50	1024	0.082054	0.005378	15.26
50	4096	0.343243	0.005855	58.62
50	16384	1.332245	0.005257	253.43
80	256	0.033014	0.008266	3.99
80	1024	0.135455	0.008577	15.79
80	4096	0.535749	0.008589	62.37
80	16384	2.147916	0.008509	252.44
100	256	0.041340	0.010404	3.97
100	1024	0.179551	0.010617	16.91
100	4096	0.678414	0.010709	63.35
100	16384	2.715843	0.010838	250.59
150	256	0.068495	0.016394	4.18
150	1024	0.255862	0.015764	16.23
150	4096	1.033439	0.015881	65.07
150	16384	4.170138	0.016361	254.89

AI STRATEGY & VERIFICATION (MILESTONE 4)

Used AI for:

- CUDA-Q API exploration
- GPU optimization ideas
- Benchmarking scaffolding

AI-generated code was never trusted blindly

All AI output validated using:

- Unit tests
- Brute-force baselines
- CPU vs GPU cross-checks

```
tests.py

Run:
  python tests.py

What this does:
- Verifies LABS energy matches a small reference implementation
- Checks physical invariances (spin-flip symmetry)
- Confirms brute-force best is consistent
- Smoke-tests QAOA sampling for small N (fast)
- Prints a clear PASS/FAIL summary
```

```
cudaq@33df3a216635:~/2026-NVIDIA/team-submissions$ python tests.py
[PASS] LABS energy matches reference
[PASS] Spin-flip symmetry E(s)=E(-s)
[PASS] Energy is nonnegative integer
[PASS] Brute force best is consistent (small N)
[PASS] QAOA smoke test (small N)

Summary: 5/5 tests passed
```

RETROSPECTIVE & TAKEAWAYS

Key Takeaways:

- GPU acceleration matters more for classical heuristics than expected
- Data transfer costs dominate for small batch sizes
- Verification is essential when using AI-generated code
- CUDA-Q + CuPy enables powerful hybrid workflows

This project demonstrates a verified, scalable, GPU-accelerated quantum-classical optimization pipeline.

LIMITATIONS & FUTURE WORK

Limitations

- QAOA depth and problem size limited by classical quantum simulation cost
- Experiments constrained to simulated QPU rather than real quantum hardware
- Classical local search still scales superlinearly for very large N

Future Work

- Run on real QPU hardware as it becomes available
- Explore deeper QAOA layers ($p > 1$) and adaptive parameter tuning
- Extend classical acceleration to multi-GPU or distributed execution
- Combine QAOA with stronger classical heuristics for hybrid workflows

Key Insight

- Near-term quantum advantage is unlocked by pairing quantum algorithms with GPU-accelerated classical optimization, not by quantum circuits alone