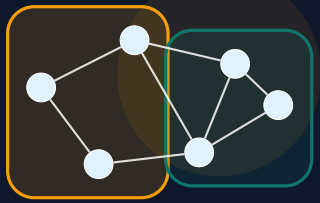
Search-Driven Quantum Circuit Decomposition

All-partition enumeration for routing and OSR-guided search for small-block synthesis

Gregory Morse
Wigner RCP GPU Day 2026

April 2026

- Result 1: all partitions via convex-set enumeration + exact set cover ILP
- Result 2: OSR-guided optimistic ansatz-extension search
- Performance lens: prune hard, batch candidates, parallelize where it matters



partition search
$\Rightarrow$ local synthesis + global routing

Compilation as a search problem

- The bottleneck is combinatorial search: how to partition, cover, and synthesize under hardware constraints.
- The winning pattern is performance-friendly: enumerate feasible work, prune aggressively, offload ranking to optimized kernels/solvers.
- Even before using GPUs directly, the workload is naturally batchable across:
  - candidate partitions,
  - circuit windows,
  - topology instances,
  - graph-search branches.
- So the message is simple: better decomposition comes from better search engineering.

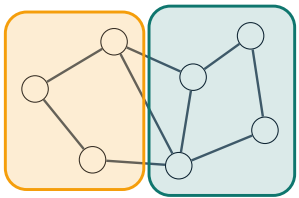
Compiler view
hard search problem + exact optimization + reusable small-k kernels

Performance view
irregular but massively parallel task graph, solver-friendly, accelerator-ready cost evaluation

Practical payoff
less routing pain, better topology awareness, stronger small-block decompositions

The decomposition problem we keep hitting

- Start with a circuit whose interactions do *not* match the target device graph.
- We want blocks that are small enough to synthesize well, but large enough to preserve structure.
- Bad partitioning spills work into routing and SWAP insertion.
- Good partitioning turns one global mess into many local problems.



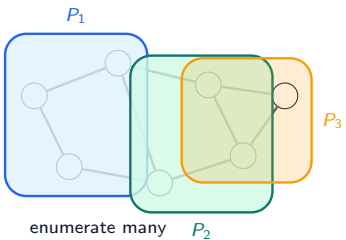
choose blocks well

and the rest of the compiler gets easier

compile = partition+local synthesize+map to topology.

Result 1: all partitions via convex-set enumeration + ILP

- Model the circuit as a DAG; a local block must be a closed convex set with no dependency path leaving and re-entering.
- Enumerate all convex candidates whose qubit footprint satisfies $|Q(P)| \leq Q_{\max}$.
- Select blocks globally with an exact-cover ILP, then enforce an acyclic quotient graph.



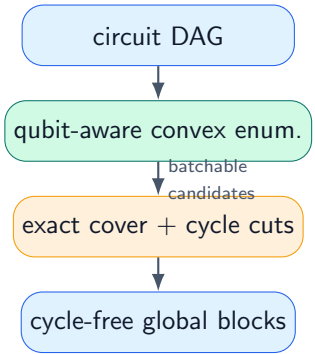
$$\min_{x_P \in \{0,1\}} \sum_{P \in \mathcal{P}} w_P x_P \quad \text{s.t.} \quad \sum_{P \ni g} x_P = 1 \ \forall g, \quad H(x) \text{ acyclic.}$$

$\mathcal{P}$ is the qubit-feasible convex candidate pool; w_P can encode block count, synthesis cost, or routing pressure.

Why this scales better than it sounds

What makes it practical?

- Precompute reachability in the circuit DAG.
- Maintain the current qubit footprint during enumeration.
- Use qubit-aware BFS to prune branches exceeding $Q_{\max}$.
- Keep $Q_{\max}$ small, typically 3–5 qubits.

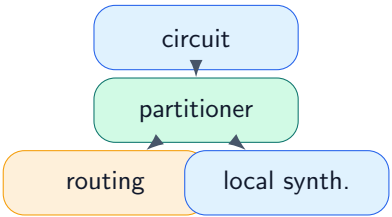


Performance angle
worst-case exponential, but small qubit footprints kill most branches before the ILP sees them.

Global validity: cover once, stay acyclic

- Exact cover ensures every gate belongs to exactly one selected partition.
- The selected partitions induce a quotient dependency graph H.
- If H has a directed cycle, add a cycle-elimination cut and re-solve.
- In experiments, this becomes the stronger partitioning stage before BQSKit/SeqPAM routing.

$$\sum_{i \in C} x_i \leq |C| - 1 \qquad C \text{ directed cycle in } H.$$

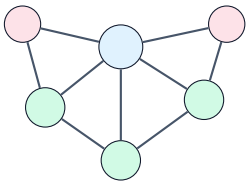


Result 2: OSR-guided ordered-CNOT search

- A node is an ordered topology-respecting CNOT structure $P = (e_1, \ldots, e_\ell)$ with surrounding U3 gates.
- BFGS optimizes the U3s for one active cut C_0 and one dyadic rank target at a time.
- After BFGS, all cuts are evaluated to produce a certificate profile, not the next CNOT insertion.

$$\Gamma_{C_0}(P) = \{(b_C, \kappa_C) : C \in \mathcal{C}\},$$
$$b_C = \lceil \log_2 r_C^{\mathrm{num}} \rceil.$$

Key idea
BFGS sharpens one cut; the all-cut OSR profile guides how the discrete tree ranks and expands nodes.



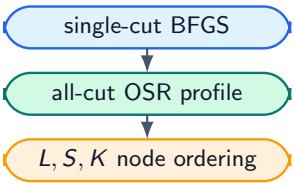
ordered CNOT structures
ranked by OSR certificates

mint = accepted

pink = skipped

How a search node is judged

1. Start from candidate ordered structure P on the hardware topology.
2. Run single-cut BFGS, then post-evaluate all cuts to get (b_C, κ_C).
3. Brute-force edge counts x_e to estimate remaining CNOT deficit $L_{C_0}(P)$.
4. Among minimum-L edge counts, minimize the surplus score $S_{C_0}(P)$.
5. Store $x^{\star}_{C_0}(P)$; it ranks topology edges for later expansion.
6. Order nodes lexicographically by $K(P) = \min_{C_0}(L, S, \Phi)$.



Message
Brute force estimates and judges the node; it does not itself insert the next CNOTs.

How κ affects expansion

- The primary key is L: the optimistic residual CNOT count still required by OSR certificates.
- κ enters only after L is fixed, through a surplus-weighted tie-breaker.
- The resulting $x^\star$ is an edge-demand vector, used to rank which topology edges to try next.
- Φ is the final deterministic cut-profile tie-breaker after L and S.

$$S_{C_0}(P) = \min_{x:\sum_e x_e = L} \sum_{C \in \mathcal{C}} \kappa_C(P; C_0)\, \mathrm{sur}_x(C)$$

$\Phi_{C_0}(P) = \sum_C (|\mathcal{C}| b_C + \kappa_C)$ is only the final profile tie-breaker; $K(P) = \min_{C_0}(L, S, \Phi)$.

Important
the surplus problem judges a candidate node; it is not the mechanism that directly places more CNOTs.

Expansion rule
sort edges by decreasing $x_e^\star$, generate coherent insertions, and accept an improving child.

Experimental results: CX count and runtime (1/2)

Comparison: all-to-all decomposition + SeqPAM routing; QuickPartitioner in BQSKit versus ILP partitioning in the OSR variant, followed by topology-constrained decomposition via BQSKit or OSR graph search.

Circuit	Qubits	Start CX	BQSKit	Time [s]	OSR	Time [s]
bv_n14	14	13	51	24	24	34
multiplier_n15	15	246	451	885	340	9233
dnn_n16	16	384	288	4006	219	22361
heisenberg_n16	16	360	196	315	137	5571
grover_19	19	192	213	5452	184	1869
cat_state_n22	20	21	21	20	21	6
ghz_state_n23	23	22	22	10	22	6
grover_23	23	240	272	805	224	1027
swap_test_n25	20	96	108	5668	106	1230
grover_33	33	360	394	1148	338	1048

Only rows with complete values in the selected columns are shown.

Experimental results: CX count and runtime (2/2)

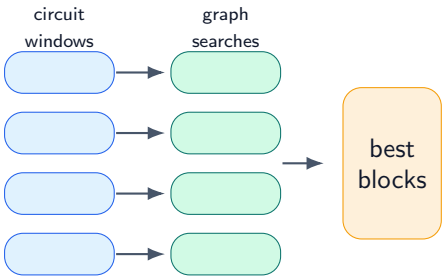
Comparison uses the same pipeline summary as the previous slide; only the selected columns are shown here for readability.

Circuit	Qubits	Start CX	BQSKit	Time [s]	OSR	Time [s]
quantum_volume_n40	40	1200	11771	20566	10436	152483
grover_45	45	504	549	1905	467	1725
grover_61	61	696	760	6479	655	3689
adder_n64	64	455	1595	698	987	5972
grover_81	81	936	1068	2641	890	6059
ising_n98	98	194	194	299	194	298
bv_n140	140	72	550	29	334	660
adder_n4	4	10	20	156	11	216
hhl_n7	7	196	212	1670	158	3549
ising_n10	10	90	77	45	44	1212

Omitted for incomplete selected fields: qft_n20, multiplier_n75, multiplier_n400, tfim, and dnn_n8.

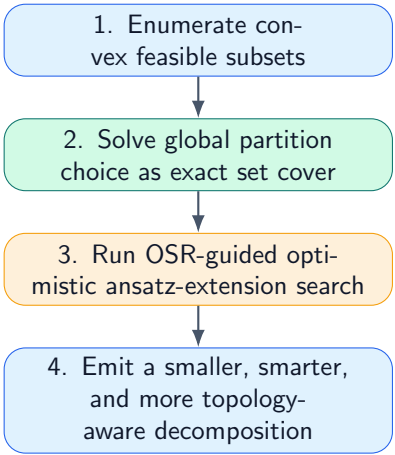
Parallelism opportunity: where concurrency shows up

- Across blocks: thousands of local decompositions are independent.
- Across candidates: partition enumeration and candidate scoring are embarrassingly parallel.
- Across targets: the same circuit can be screened against multiple hardware graphs.
- Inside kernels: OSR updates, weighted-tail scoring, and branch scoring are natural accelerator targets.
- So even if today's implementation is CPU-first, the algorithmic structure is already parallelism-ready.



Translation for GPU Day
irregular compiler search rewards batching, pruning, and careful cost-kernel design.

Put together, the two results form one compiler story



Take-home message

- Exact global partitioning is practical.
- Small-block local search is worth it.
- Both are compiler searches with natural parallel structure.

Future direction
parallel candidate generation, batched cost evaluation, and hardware-aware scheduling are natural next steps.

If you remember only three things

1. Partition quality dominates downstream compilation quality.
2. Exact set cover + solver technology makes all partitions surprisingly practical at large sizes.
3. OSR-guided optimistic ansatz-extension search turns small-block decomposition into a reusable optimization kernel.

$$\text{Better search} \Rightarrow \text{better decomposition} \Rightarrow \text{less routing pain}$$

Questions?