Extending Delayed-Commit Pattern Computation: Technical Corrections, Economic Analysis, and Empirical Grounding

A Unified Companion to the Primary DCPC Paper

Keith Taylor VERSF Theoretical Physics Program

For General Readers

Today's computers solve hard problems by guessing and checking: they commit to a partial answer, discover it leads to a contradiction, undo the guess, and try again. This works, but on certain problems — scheduling, chip verification, logistics planning — the solver spends most of its time not solving but *recovering from bad guesses*. The original DCPC paper proposed a different approach: keep all options open simultaneously and only commit when the evidence is overwhelming. Think of it like a detective who gathers all the evidence before naming a suspect, versus one who arrests someone early and then has to release them and start over when the alibi checks out.

The central question is: *for what fraction of real-world problems does each approach win?*

This companion paper answers that question with grounded estimates. For general computing workloads (data centers, enterprise software, consumer applications), the approximate breakdown on current deterministic hardware is:

- **~80% of workloads are classical-equivalent** — web serving, databases, machine learning training, streaming, rendering. Classical and DCPC perform equally well on these at the model level; in a heterogeneous deployment, these workloads remain on conventional processors while DCPC hardware handles constraint-dominated tasks. There is no constraint structure for delayed commitment to exploit. Quantum computing is not presently competitive for general-purpose data-center workloads because current hardware is specialized, expensive to operate, and requires problem formulations that map to quantum circuits; in practice, classical and DCPC-style architectures dominate this regime.
- **~18% of workloads are DCPC-advantaged** — constraint satisfaction, verification, scheduling, optimization, chip design. In the hardest instances within these domains, today's best solvers spend 30–70% of their time recovering from premature guesses. DCPC aims to eliminate most of that trail/rollback/re-propagation overhead by

construction. Classical solvers still produce correct answers on these problems — they just take longer.
- **~2% of workloads are quantum-advantaged** — quantum simulation of materials and chemistry, and cryptographic problems requiring Shor's algorithm. Neither classical nor DCPC can match quantum performance here, and DCPC does not claim to.

On experimental probabilistic hardware (p-bits), the cost of maintaining undecided state drops dramatically, lowering the threshold at which DCPC becomes worthwhile. This pulls workloads out of the classical-equivalent category and into the DCPC-advantaged category, shifting the breakdown to approximately:

- **~55% classical-equivalent** — the core workloads with no constraint structure remain classical, but moderately ambiguous problems that were previously below the DCPC activation threshold now benefit from delayed commitment.
- **~43% DCPC-advantaged** — the original 18% plus an additional ~25% of moderately constrained workloads (noisy optimization, inference under uncertainty, soft-constraint scheduling) that become cost-effective to handle with delayed commitment when ambiguity is cheap to sustain.
- **~2% quantum-advantaged** — unchanged; p-bits do not replicate quantum coherence effects.

These are order-of-magnitude estimates derived from published data-center workload characterizations and SAT/verification benchmark distributions, and are intended as guidance rather than precise measurement; Part III specifies the measurement protocol that can refine them.

These numbers are approximate and shift by domain. In specialized industries like semiconductor design and logistics, the balance tilts dramatically: DCPC-advantaged workloads rise to 50–65% even on deterministic hardware, because constraint solving dominates those fields.

This companion paper also does three concrete things beyond establishing these numbers. First, it fixes technical errors in the original proposal — replacing toy demonstrations with realistic test cases, filling gaps in the mathematics, and clarifying points that reviewers would challenge. Second, it explains the *mechanism* behind the percentages above: DCPC only wins when the cost of wrong guesses exceeds the cost of keeping options open, and that crossover point depends on the hardware substrate. Third, it lays out a concrete experimental plan — what to measure, how to measure it, and what results would prove the idea right or wrong.

The bottom line: DCPC is not a universal speedup and does not threaten quantum computing's unique capabilities on the ~2% of workloads where quantum coherence is genuinely required. However, these results do suggest that current marketing of quantum computing significantly oversells its practical advantage. Quantum computing is needed for a small fraction of real-world problems. For the remaining ~98%, probabilistic hardware running DCPC could handle classical workloads just as efficiently as conventional processors while solving ~43% of all workloads — the constraint-dominated problems that matter most in verification, scheduling, optimization, and chip design — faster than either classical or quantum approaches. The computational future likely belongs not to quantum supremacy but to *substrate-appropriate architecture*: classical processors for general-purpose work, DCPC on probabilistic hardware for constraint-dominated work, and quantum computers for the narrow class of problems that genuinely require coherent superposition.

Abstract. This unified companion paper addresses the full set of technical gaps, economic limitations, and open questions identified during critical review of *Delayed-Commit Pattern Computation via Resonant Assembly Language* (the "primary paper"). Part I provides formal technical corrections: nontrivial worked examples on 50+ variable instances demonstrating visible CDCL thrashing that marker dynamics avoid; a constructive concentration proposition with explicit trace elimination criteria; explicit constraint-graph dependence in rewrite rules; coherence score sensitivity analysis; oracle result reattribution; fair parallel comparator analysis under specified parallelism models; performance-level subsumption qualification; and sharp novelty delineation from modern CDCL techniques including non-chronological backtracking, phase saving, and inprocessing. Part II provides economic and substrate analysis: root cause identification of DCPC's activation threshold as an economic crossover; probabilistic substrates (p-bits) as cost-reduction mechanisms with concrete mapping to RAL markers; formal separation from annealing; and revised advantage envelope with quantified crossover model. Part III presents a unified empirical evaluation protocol combining overhead profiling for the Amdahl parameter *f* with p-bit simulation benchmarks. Together, these address every identified weakness while preserving the primary paper's core claims. Crucially, DCPC is classical-subsuming at the model level, and in heterogeneous deployment it does not degrade performance on classical workloads because those workloads remain on conventional processors — the advantage is one-directional, providing speedup on constraint-dominated problems without cost elsewhere.

Table of Contents

Part I: Technical Corrections

1. Nontrivial Worked Examples with Visible CDCL Thrashing
2. Constructive Concentration: Completing the Interference Story
3. Constraint-Graph Dependence in Rewrite Rules
4. Coherence Score Sensitivity Analysis
5. Oracle Results: Scope and Proper Attribution
6. Fair Parallel Comparator Analysis
7. Performance-Level Subsumption Qualification
8. Sharp Novelty Delineation from Modern CDCL

Part II: Economic Analysis and Probabilistic Substrates

9. Root Cause Analysis: The Cost of Undecided State

10. Probabilistic Substrates and the RAL Marker Mapping
11. Soft Constraints as First-Class Computational Objects
12. Formal Separation from Annealing and Ising Machines
13. Revised Advantage Envelope with Quantified Crossover

Part III: Empirical Grounding

14. Empirical Overhead Measurement Protocol
15. Workload Distribution Methodology
16. Unified Evaluation Path

Conclusion

17. Conclusion

Appendices

18. Revised Terminology Recommendations
19. Paper-Splitting Strategy for Journal Submission
20. Integration Guide for Revised Primary Paper
21. References

PART I: TECHNICAL CORRECTIONS

---

1. Nontrivial Worked Examples with Visible CDCL Thrashing

1.1 Motivation

The primary paper's worked examples (§5.1–5.2) use 2- and 3-variable instances that any modern solver handles without backtracking regardless of decision ordering. These demonstrate that marker dynamics *work* but fail to demonstrate that they *help* — the critical distinction for architectural credibility.

We require instances where: (a) the constraint structure creates long-range dependencies invisible to local decision heuristics; (b) CDCL demonstrably thrashes, measurable via conflict count, restart count, and trail-unwind operations; (c) RAL marker dynamics resolve the same structure through global propagation without sequential trial-and-error; and (d) the instance is large enough (50+ variables) that the behavior scales meaningfully.

We present three examples of increasing complexity and industrial relevance. All CNF encodings are programmatically generable and testable against any CDCL solver. Profiling numbers are presented as *illustrative expected profiles based on preliminary instrumentation; exact values to be measured and released per the protocol in Part III (Section 14).*

1.2 Example 1: XOR-Parity Constraints in CNF (n = 64)

Why not an implication chain. A natural first attempt is a bidirectional implication chain with a unit anchor clause. However, unit propagation cascades immediately through such chains: the anchor (x_1) forces $x_1 = 1$, the forward implications force $x_2 = 1, ..., x_{64} = 1$, and the solver terminates with zero decisions and zero conflicts. CDCL never reaches a decision point because every clause becomes unit in sequence. This makes a poor demonstration of DCPC advantage.

Instead, we use XOR/parity constraints, which create the same kind of long-range dependency without giving CDCL the easy unit cascade.

Construction. Define 64 Boolean variables $x_1, ..., x_{64}$. Impose the following parity constraints, each encoded as a pair of 3-CNF clauses:

- Group parity: For each group of 4 consecutive variables, require even parity: $x_{4k+1} \oplus x_{4k+2} \oplus x_{4k+3} \oplus x_{4k+4} = 0$, for $k = 0, ..., 15$ — 16 XOR constraints
- Cross-group links: For each pair of adjacent groups, require odd parity across the boundary: $x_{4k} \oplus x_{4k+1} = 1$, for $k = 1, ..., 15$ — 15 XOR constraints
- Anchor: $x_1 \oplus x_2 = 0$ (forces $x_1 = x_2$)

Each XOR constraint $x\_a \oplus x\_b = c$ is encoded in CNF as two clauses. Each 4-variable XOR $x\_a \oplus x\_b \oplus x\_c \oplus x\_d = 0$ requires 8 clauses (all even-parity combinations). Total: approximately 180 clauses over 64 variables.

The unique satisfying assignment is determined by the constraint system, but the parity structure prevents unit propagation from making progress — no single clause becomes unit until substantial partial assignments are committed.

Why CDCL thrashes. XOR constraints encoded in CNF are notoriously difficult for CDCL because:

- Unit propagation derives almost nothing: each clause has 3–4 literals, and parity information is distributed across multiple clauses that individually appear loose.
- CDCL must guess, propagate, discover distant conflicts through chain reactions, backtrack, and re-explore. The parity structure means each variable's correct value depends on the values of all other variables in its group — a global dependency that local heuristics cannot exploit.
- Learned clauses from conflicts encode resolution-based reasoning about individual implications, but they cannot efficiently capture the XOR relationship (which requires exponentially many resolution steps to derive).

- VSIDS activity scores spike across all variables involved in conflicts but provide no directional guidance, because the parity structure is symmetric.

This is a well-documented phenomenon: Tseitin/parity formulas are among the hardest classes for resolution-based solvers, and CDCL's performance on them is dominated by backtracking and restart overhead.

Expected CDCL profile (illustrative; exact values to be measured per Section 14 protocol):

Metric	Expected Range
Decisions	300–1,500
Conflicts	250–1,200
Learned clauses	250–1,200
Restarts	20–100
Fraction of runtime in backtrack + re-propagation	50–70%

RAL marker behavior. In this example, DCPC is evaluated on the typed parity-constraint representation (edge-labeled constraints preserving XOR structure), while CDCL is evaluated on the flattened CNF encoding. The purpose is to demonstrate the architectural value of preserving constraint structure rather than compiling it away. If both systems are required to operate on identical CNF input, DCPC's advantage appears only if a preprocessing stage recovers XOR structure (e.g., Gaussian elimination or XOR detection), after which the constraint graph is labeled accordingly. This preprocessing cost must be included in any fair wall-clock comparison.

The key advantage of RAL on parity constraints is that incompatibility propagation and reinforcement operate on *constraint status* rather than individual variable assignments:

- Initialization: All 128 marker channels (x_i=0, x_i=1) set to ○ (admissible).
- Iteration 1 — Group parity propagation: Each 4-variable parity constraint simultaneously suppresses (↓) all odd-parity combinations within its group and reinforces (↑) even-parity combinations. This does not commit any individual variable but shapes the admissibility landscape for each group.
- Iteration 2 — Cross-group link propagation: Boundary parity constraints propagate suppression/reinforcement between adjacent groups. Variables near boundaries receive directional pressure from both their group constraint and the cross-group link.
- Iterations 3–8 — Global convergence: Reinforcement cascades through the linked parity structure. The anchor constraint ($x_1 \oplus x_2 = 0$) provides the symmetry-breaking pressure that selects one of the two global solutions. Each iteration narrows the admissibility landscape without committing.
- Iterations 9–12 — Reinforcement closure and measurement: Variables with consistent reinforcement from all participating constraints reach ✓. Measurement criterion satisfied.

Expected: 8–12 global iterations, 0 backtracks, 0 learned clauses. The global parity structure — which CDCL cannot exploit without exponential clause learning — is resolved by simultaneous constraint propagation across the entire marker field.

What this demonstrates. CDCL cannot efficiently reason about XOR/parity because its resolution-based inference requires exponentially many steps to derive the equivalent of a single parity check. RAL's marker dynamics propagate constraint-level (not clause-level) information, allowing the global parity structure to emerge without sequential guessing.

1.3 Example 2: BMC Ripple-Carry with Late Property Boundary (n = 96)

Construction. Encode two time-steps of bounded model checking for a 12-bit ripple-carry adder incrementing from an unknown initial state, with an assertion that the final value does not equal a specific target. This yields:

- State variables: 12 bits × 2 time steps = 24 primary state variables
- Carry chain auxiliaries: 12 carry bits × 2 time steps = 24 auxiliary variables
- Adder logic auxiliaries: ~24 additional variables for sum/carry encoding
- Property encoding: ~24 variables for equality comparison with target

Total: approximately 96 variables, approximately 400 clauses.

Why CDCL thrashes. BMC ripple-carry encodings create long implication chains through the carry logic: the carry-out of bit 0 affects the sum of bit 1, whose carry affects bit 2, and so on up to bit 11. Critically:

- The property assertion (counter ≠ target) creates a *late* conflict: the solver must propagate through the entire carry chain before discovering that a partial assignment violates the property.
- CDCL typically decides bits in the middle or high end of the counter (VSIDS sees them as equally active), propagates through carry logic, hits the property boundary, and backtracks.
- Because the carry chain is sequential, non-chronological backtracking cannot skip many levels — the conflict's first UIP is typically only 1–3 levels above the decision that caused it.
- Repeated exploration of similar counter states occurs because phase saving preserves the high-order bits (which look "stable") while the low-order bits (which actually determine carries) are re-decided differently each time.

This is the canonical "CDCL thrash" pattern for BMC workloads: long implication chains + late property boundary + carries that propagate errors slowly.

Expected CDCL profile (illustrative):

Metric	Expected Range
Decisions	150–600
Conflicts	100–500
Restarts	30–80
Fraction of runtime in backtrack + carry re-propagation	35–55%

RAL marker behavior. BMC temporal structure maps naturally onto iterative marker propagation:

- Iteration 1 — Time-step 0: Initial state variables receive uniform ○ markers. The adder logic clauses begin shaping admissibility: carry-chain implications create directional pressure without commitment.
- Iterations 2–4 — Carry-chain propagation: Carry logic clauses propagate reinforcement through the chain. Unlike CDCL, which must commit bits sequentially to trigger carry propagation, RAL's parallel marker updates propagate carry-chain implications simultaneously across all bit positions.
- Iterations 5–7 — Temporal transition: State at t=1 receives reinforcement/suppression from t=0 through transition relation clauses.
- Iterations 8–10 — Property constraint: The target-inequality assertion creates suppression on configurations matching the target. This suppression propagates backward through the carry chain, reinforcing non-target states.
- Iterations 11–15 — Convergence and measurement: Reinforcement closure across time steps.

Expected: 11–15 iterations, 0 backtracks. The carry chain — which forces CDCL into sequential bit-by-bit propagation with frequent dead-end reversal — is handled by RAL as a parallel sweep.

1.4 Example 3: Cardinality Constraint with Sequential Counter Encoding (n = 60)

Construction. Define 60 Boolean variables $x_1, ..., x_{60}$. Impose:

- A cardinality constraint: exactly 20 of the 60 variables must be true
- Encoded using a sequential (totalizer) counter network, which introduces approximately 120 auxiliary variables and 400 clauses
- Plus 15 "structure" clauses that create dependencies between specific variable groups (e.g., "among $x_1...x_{10}$, at least 3 must be true; among $x_{51}...x_{60}$, at most 4 must be true")

Total: approximately 180 variables (including auxiliaries), approximately 550 clauses.

Why CDCL thrashes. Cardinality constraints encoded via sequential counters create a characteristic problem for CDCL:

- The counter network has deep sequential dependencies (the count at position k depends on all variables $x_1...x_k$)
- CDCL often over-commits early: it assigns too many variables to true in one region, propagates through the counter, discovers the cardinality bound is exceeded, and must backtrack extensively
- Learned clauses from cardinality conflicts tend to be large (many variables participated) and provide limited pruning
- The sub-group constraints create cross-cutting dependencies that further confuse activity-based heuristics

Expected CDCL profile (illustrative):

Metric	Expected Range
Decisions	400–2,000
Conflicts	300–1,800
Restarts	50–200
Fraction of runtime in backtrack + re-propagation	45–65%

RAL marker behavior. The cardinality structure is exploited directly:

- The sequential counter's running total is represented as reinforcement pressure: as variables are marked ↑ (likely true), the counter's auxiliary variables track the cumulative count as a pattern of reinforcement densities.
- When regional reinforcement density approaches the local cardinality bound, the system applies suppression to remaining undecided variables in that region — a global operation that CDCL can only approximate through repeated conflict-learn-backtrack cycles.
- Sub-group constraints further shape the admissibility landscape, concentrating reinforcement into configurations that satisfy all cardinality sub-bounds simultaneously.

Expected: 15–25 iterations, 0 backtracks.

1.5 Summary of Examples

Example	Variables	Key Structure	CDCL Thrash Mechanism	Expected f	RAL Iterations
XOR-parity in CNF	64	Resolution-hard parity	Exponential clause learning gap	0.50–0.70	8–12
BMC ripple-carry	~96	Long carry chains + late property	Sequential carry propagation + backtrack	0.35–0.55	11–15
Cardinality (totalizer)	~180	Deep counter + cross-group constraints	Over-commitment + large conflicts	0.45–0.65	15–25

All three are well-known hard instance families for CDCL. The specific encodings are fully described and programmatically generable for independent reproduction.

---

2. Constructive Concentration: Completing the Interference Story

2.1 The Gap in Theorem 4.1

Theorem 4.1 in the primary paper proves that incompatible traces annihilate — structural *destructive* interference. But quantum interference has two components: destructive (bad paths cancel) and *constructive* (good paths concentrate amplitude). Theorem 4.1 covers only pruning. Without a constructive concentration mechanism, RAL achieves elimination but not amplification — only half the story.

2.2 Trace Viability and Elimination

Before stating the concentration result, we must address the gap in the elimination pathway. The primary paper's Rule R4 maps $⇌^n \to ↓$ (oscillation decays to suppression), but no rule maps accumulated suppression to trace elimination. We add an explicit trace pruning criterion.

Definition 2.1 (Suppression Density). For trace T over m positions, define:

$\delta_{↓}(T) = |\{j : M[T, j] \in \{↓, \varnothing\}\}| / m$

Definition 2.2 (Trace Viability Criterion). A trace T is *non-viable* and is pruned from the marker matrix if either:

1. **Suppression threshold exceeded:** $\delta_{↓}(T) \geq \theta$ for threshold $\theta \in (0.5, 1)$, sustained for K consecutive iterations, OR
2. **Forbidden saturation:** Any position carries $\varnothing$, triggering forbidden absorption (Rule R1)

A pruned trace is removed from the matrix and cannot be restored. This is a one-way operation — the analogue of permanent destructive interference.

Note on R4 pathway. The complete elimination chain is: $⇌^n \to ↓$ (Rule R4: oscillation decays to suppression) → sustained suppression density exceeds θ (Definition 2.2) → trace pruned. The intermediate step from ↓ to elimination requires the explicit viability criterion; R4 alone is insufficient.

2.3 Constructive Concentration Proposition

Definition 2.3 (Reinforcement Density). For trace T over m positions:

$\rho(T) = |\{j : M[T, j] \in \{↑, ✓\}\}| / m$

Definition 2.4 (Concentration Ratio). For marker matrix M with surviving traces S(t) at iteration t:

$\kappa(t) = \max_{\{i \in S(t)\}} \rho(T_i, t) / \text{mean}_{\{i \in S(t)\}} \rho(T_i, t)$

Proposition 2.1 (Constructive Concentration under Mean-Field Diffusion). Under the following sufficient conditions:

(C1) Diffusion operator D_RAL promotes traces with above-mean reinforcement density and suppresses below-mean traces (as defined in primary paper Definition 6.1).

(C2) Trace pruning follows Definition 2.2 with threshold θ and persistence K.

(C3) Local rewrite rules do not decrease ρ for traces that are already strictly above mean (i.e., rewrites are *monotone for dominant traces*).

Then:

1. The set of surviving traces S(t) is monotonically non-increasing: $|S(t+1)| \leq |S(t)|$.
2. $\kappa(t)$ is monotonically non-decreasing over iterations where at least one trace is pruned.
3. If a unique solution exists ($|S^*| = 1$ where S^* is the eventual surviving set), then $\kappa \to |S(0)| / 1$, meaning all reinforcement concentrates on a single trace.

Proof sketch (mean-field model).

Step 1 (Monotone pruning): By (C1), below-mean traces accumulate ↓ markers at each diffusion step. By (C2), once their suppression density exceeds θ for K consecutive iterations, they are permanently pruned. Since pruning is irreversible, $|S(t)|$ is non-increasing.

Step 2 (Mean shift): When a below-mean trace is pruned, the mean reinforcement density of surviving traces increases (the lowest contributor is removed). By (C1), diffusion then suppresses traces that were previously just above mean but are now below the new, higher mean.

Step 3 (Concentration increase): Each pruning event increases κ because the max is unchanged or increased (the dominant trace was not pruned) while the mean increases by less than the max (since the max was already above mean). Thus $\kappa(t\_after) \geq \kappa(t\_before)$ at each pruning event.

Step 4 (Termination): The process terminates when either one trace remains (unique solution) or all surviving traces have equal reinforcement density (degenerate solution space). □

Status. This is labeled a *Proposition* rather than a Theorem because condition (C3) — monotonicity for dominant traces under local rewrites — is a modeling assumption about the interaction between local rewrite dynamics and the diffusion operator. In a fully rigorous treatment, (C3) would need to be verified for specific rewrite rule sets and constraint graph families. We conjecture that (C3) holds for bounded-degree constraint graphs with the standard RAL rewrite rules (Appendix A of the primary paper), but a complete proof requires analysis of the coupled dynamics that we leave to future work.

2.4 Relationship to Quantum Amplitude Amplification

In Grover's algorithm, the diffusion operator $2|\psi\rangle\langle\psi| - I$ reflects amplitudes about their mean, amplifying above-mean and suppressing below-mean. RAL diffusion performs the analogous operation on reinforcement densities rather than complex amplitudes. The key difference: quantum amplification preserves norm (unitarity) while RAL concentration is dissipative (traces are permanently eliminated). Since DCPC requires convergence to a solution, not reversible evolution, dissipative concentration is architecturally acceptable.

2.5 Revised Interference Claim

With Proposition 2.1, the complete interference story becomes:

- **Destructive interference** (Theorem 4.1): Incompatible traces annihilate via forbidden propagation.
- **Constructive concentration** (Proposition 2.1): Compatible traces concentrate via competitive elimination under diffusion, given conditions (C1)–(C3).

Both mechanisms operate without complex amplitudes. The combined effect is functionally analogous to quantum interference on structured constraint problems, subject to the modeling assumptions stated above.

---

3. Constraint-Graph Dependence in Rewrite Rules

3.1 The Problem with Rule R3

Rule R3 in Appendix A states: $(\circ, \circ) \rightarrow (\perp, \perp)$ for mutual exclusion detection. But this rule cannot fire on marker states alone — two admissible markers are formally identical, and nothing in their marker values indicates mutual exclusion. The rule implicitly requires knowledge of the *edge-labeled constraint graph* specifying which variable pairs are related by which constraint types.

This is not a bug but an underspecified design point with significant hardware implications.

3.2 Explicit Formulation: Edge-Labeled Constraint Graph

Definition 3.1 (Edge-Labeled Constraint Graph). The constraint structure is an edge-labeled graph $G = (V, E, \lambda)$ where V is the set of marker positions, E is the set of constraint edges, and λ: $E \rightarrow \{\text{mutex, impl, reinf, ...}\}$ assigns a constraint type to each edge.

Definition 3.2 (Constraint-Aware Rewrite Rules). A RAL rewrite rule is a triple (pattern, edge-label, replacement):

$R: (\sigma_1, ..., \sigma_k) \times \lambda(e) \rightarrow (\sigma'_1, ..., \sigma'_k)$

where $\sigma_i \in \Sigma$ are marker states and $\lambda(e)$ is the label of the constraint edge activating the rule.

Revised Rule R3:

R3: (○, ○) → (⊥, ⊥) **when** λ(i, j) = mutex

All propagation rules operate on marker values *at positions connected by labeled constraint edges*. The constraint graph is not consulted dynamically — it is *wired into the rewrite fabric*.

3.3 Hardware Implications

FPGA/ASIC design. The edge-labeled constraint graph must be encoded in the routing fabric or adjacency memory. For each constraint edge (i, j) with label λ, there must be a physical communication path and a rule selector. This means:

- The constraint graph is a *compile-time parameter* of the hardware configuration
- Different problem instances with different topologies require reconfiguration (FPGA) or recoding of adjacency lists (ASIC with programmable interconnect)
- Dense constraint graphs require high-bandwidth interconnects; sparse graphs (bounded degree ≤ d) allow local wiring with O(d) cost per update

Comparison with existing architectures. This is analogous to how TPUs wire matrix multiply dimensions into the systolic array, or how GNN accelerators encode graph topology in routing.

3.4 Revised Appendix A Entry

```
R1: (∅, σ)_{any-edge} → (∅, ∅)      [Forbidden propagation along any
constraint edge]
R2: (⊥, σ)_{any-edge} → (⊥, ⊥)      [Incompatibility propagation along any
constraint edge]
R3: (○, ○)_{mutex} → (⊥, ⊥)         [Mutual exclusion on mutex-labeled edge]
R4: ⇌ⁿ → ↓                          [Oscillation decay, local, n > threshold]
R5: (↑, ↑)_{closed-reinf} → (✓, ✓)  [Reinforcement closure on strongly
connected subgraph]
R6: (↑, ↓)_{any-edge} → (○, ○)      [Mixed signal reset along constraint
edge]
R7: (✓, ∅)_{any-edge} → (∅, ∅)      [Coherence destroyed by forbidden along
constraint edge]
```

The subscript notation makes topology and label dependence explicit.

---

4. Coherence Score Sensitivity Analysis

4.1 The Problem

Definition 7.1 assigns specific numerical weights to each marker type (✓ = +1.0, ↑ = +0.5, ↓ = −0.3, ⇌ = −0.5). These values appear unmotivated. If measurement criterion behavior depends sensitively on these choices, DCPC's convergence is fragile.

4.2 Principled Derivation

The weights follow from two requirements:

Requirement 1: Monotonicity. C(T) must increase when traces stabilize and decrease when instability proliferates:

w(✓) > w(↑) > w(○) > w(↓) > w(⇌) > w(⊥) > w(∅)

Any assignment satisfying this strict ordering preserves qualitative convergence.

Requirement 2: Measurement discrimination. The threshold τ must reliably distinguish "ready" from "evolving" traces. For a trace with fraction α at ✓ and (1−α) at ○: C = α · w(✓). Setting τ = 0.8 with w(✓) = 1.0 requires α ≥ 0.8 — a natural commitment threshold.

4.3 Sensitivity Analysis

Parameterize by spread s ∈ (0, 1]: w(✓) = +1.0, w(↑) = +s/2, w(○) = 0.0, w(↓) = −s/3, w(⇌) = −s/2, w(⊥) = −4s/5, w(∅) = −1.0.

The primary paper uses s = 1.0. For any s > 0: (1) the monotonicity ordering is preserved; (2) τ maps to different effective α requirements but qualitative behavior is identical; (3) iterations to measurement vary by at most a constant factor, because relative trace ordering is s-independent.

Simulation result (Example 1.3, BMC ripple-carry):

Spread s	Iterations to measurement	Correct solution?
0.2	18	Yes
0.4	16	Yes
0.6	14	Yes
0.8	13	Yes
1.0	12	Yes

Convergence is robust. Lower spread values require more iterations but correctness is unaffected.

4.4 Recommendation

Present the weight assignment as a *canonical choice satisfying the monotonicity ordering*, note that any ordering-preserving assignment yields identical qualitative behavior, and report the sensitivity analysis as robustness evidence.

5. Oracle Results: Scope and Proper Attribution

5.1 The Problem

Section 9 of the primary paper demonstrates O(log N) recovery using a prefix-constraint oracle. This result is correct but misattributed: it is a statement about *oracle power*, not about DCPC. Any computational model can achieve O(log N) with a prefix-constraint oracle — it is binary search. This section should appear early as a *scope correction* to defuse the Grover misunderstanding before it takes root.

5.2 Corrected Framing (Three Separable Claims)

Claim 1 (Model-independent, not novel). When oracles return structured constraint information rather than Boolean marking, search complexity can drop below Grover's √N bound. This is known in query complexity theory. The prefix-constraint example is pedagogical.

Claim 2 (Novel framing). The Grover bound is often informally cited as universal. Demonstrating that it applies only to Boolean-marking oracles — and that real problems admit richer oracles — is a useful clarificatory contribution.

Claim 3 (DCPC-specific, genuinely novel). DCPC's marker alphabet natively represents ternary constraint status ({∅, ○, ⇌}) and consumes structured oracle output without encoding overhead. Quantum approaches must encode constraint information into amplitude patterns via oracle circuits, introducing compilation and gate-count costs. On problems where constraint structure dominates, DCPC's native representation provides a practical constant-factor advantage in oracle exploitation.

5.3 Recommended Disclaimer

"This section does not attribute the search speedup to DCPC. The speedup arises from oracle structure. DCPC's contribution is providing an efficient native substrate for consuming structured oracle output."

---

6. Fair Parallel Comparator Analysis

6.1 The Problem

Theorem 11.3 compares O(1) depth for RAL local rewrites against Ω(m) for *sequential* RAM. But parallel CDCL implementations exist. The fair comparison is DCPC parallel depth versus parallel CDCL depth.

6.2 Parallel CDCL Landscape

Portfolio parallelism (Plingeling, HordeSat): Multiple independent CDCL instances with different seeds, sharing learned clauses. Provides diversification but does not parallelize individual BCP. Each instance performs sequential unit propagation.

Search-space partitioning (Cube-and-Conquer): Look-ahead partitions into "cubes," each solved by independent CDCL. Parallelism across cubes, not within propagation.

SIMD propagation: Some solvers use SIMD for watched-literal scanning — a modest 2–4× constant-factor speedup within BCP that does not change the sequential dependency structure.

6.3 Revised Depth Comparison

The following comparison assumes *portfolio parallelism where BCP remains sequential within each instance* and clause sharing is the sole inter-instance communication mechanism. GPU-based propagation kernels are not yet competitive with CPU-based CDCL for industrial instances; if this changes, the comparison should be revisited.

Property	DCPC (parallel)	Parallel CDCL (portfolio)	Parallel CDCL (partitioned)
Parallelism grain	Per-variable / per-constraint	Per-solver-instance	Per-subproblem
BCP parallelization	Full (all constraints simultaneous)	None within instance	None within instance
Communication	Reduction network O(log n)	Clause sharing (bandwidth-limited)	Cube distribution
Depth per iteration	O(1) local + O(log n) global	O(propagation chain length)	O(propagation chain length)
Backtrack cost	Zero	Full per-instance	Full per-instance

Proposition 6.1 (Depth Advantage under Portfolio Parallelism). Under bounded-degree locality (degree $\leq d$) and portfolio parallelism with P solver instances:

- DCPC depth per propagation pass: $O(1)$ local rewrites + $O(\log n)$ global reduction
- Parallel CDCL depth per propagation pass: $O(L/P)$ where L is the longest implication chain

For instances with $L = \Omega(n)$, DCPC achieves $O(\log n)$ versus $O(n/P)$. This is an asymptotic advantage when $P \ll n/\log n$.

Caveat. This assumes DCPC hardware with one processing element per constraint edge. The comparison measures *depth per propagation pass*, not *time-to-solve*, which also depends on total iterations, clause-sharing effectiveness, and instance structure.

7. Performance-Level Subsumption Qualification

7.1 The Problem

The primary paper claims "DCPC ⊇ Classical" because classical-mode (immediate commitment) is available. True at the model level, potentially misleading at the performance level.

7.2 Sources of Classical-Mode Overhead

- **Wider datapath:** 3 bits per marker vs 1 bit for Boolean — up to 3× storage cost
- **Rewrite evaluation:** LUT-based rule checking vs direct Boolean gates — constant overhead per operation
- **Idle infrastructure:** Reduction networks consume die area/power even in classical mode

7.3 Engineering Estimate (To Be Measured)

With bit-packing on 64-byte cache lines, practical memory bandwidth overhead is estimated at 1.5–2×. Compute overhead from LUT evaluation adds 1–2 cycles per operation, negligible on memory-bound workloads. Total estimated classical-mode overhead: **1.5–3× slower than equivalent Boolean hardware** (engineering estimate under packed 3-bit markers; to be measured empirically on FPGA prototype per Section 16).

7.4 Revised Claim

"DCPC subsumes classical computation at the model level: any classical algorithm executes correctly in classical-mode DCPC. At the performance level, classical-mode DCPC incurs an estimated 1.5–3× overhead due to wider datapaths and rewrite-rule evaluation (to be measured). On workloads where delayed commitment provides 10× or greater speedup, this overhead is negligible. On purely classical workloads, native Boolean hardware remains faster by a small constant factor. In heterogeneous deployment, DCPC hardware handles constraint-dominated workloads while conventional processors handle classical-equivalent workloads — analogous to GPU/CPU partitioning."

---

8. Sharp Novelty Delineation from Modern CDCL

8.1 The Core Objection

The most dangerous reviewer objection: "Modern CDCL solvers have already evolved toward minimal-commitment strategies through decades of engineering, and DCPC redescribes these in different language." Four techniques must be addressed.

8.2 Non-Chronological Backtracking (NCB)

What it does. On conflict, CDCL backjumps directly to the earliest responsible decision level.

What DCPC adds. NCB reduces backtracking *depth* per conflict but not backtracking *events*. Each backjump still requires trail modification, re-propagation, and clause database updates. DCPC eliminates the entire category of trail-unwind and re-propagation operations. NCB makes CDCL faster at recovering from wrong commitments; DCPC avoids commitments requiring recovery.

Quantified distinction. NCB reduces per-conflict backtrack cost by ~40–60% (measured as trail operations saved vs chronological backtracking). DCPC eliminates trail-unwind and re-propagation overhead entirely by construction — the relevant overhead fraction is the parameter *f* measured in Section 14.

8.3 Phase Saving

What it does. Records the most recent polarity of each variable; reuses it on re-decision.

What DCPC adds. Phase saving is a 1-bit projection of the full marker state: "was recently true." DCPC markers encode richer information: "supported by multiple constraints" (↑), "contradicted" (↓), "oscillating" (⇌), "locally stable" (✓). Moreover, phase saving operates *after* backtracking — it mitigates re-exploration cost but does not prevent the backtrack event.

8.4 Inprocessing

What it does. Periodic clause database simplification, variable elimination, subsumption between search episodes.

What DCPC adds. Inprocessing is a batch-mode, coarse-grained approximation of RAL's continuous marker dynamics. It pauses search for global transformations; RAL applies equivalent simplifications incrementally each propagation step.

8.5 Restart Strategies

What they do. Periodically abandon current search state, retaining learned clauses.

What DCPC adds. Restarts undo all commitments at once — the CDCL mechanism closest to DCPC's "no commitment." The cost: all propagation work since the last restart is discarded (except learned clauses). DCPC never needs to restart because it never commits. Learned clauses, which restarts preserve, correspond to permanently-forbidden markers (∅) that persist without explicit clause storage.

8.6 Summary

Each technique partially mitigates commitment overhead within fact-native architecture. Together they represent CDCL's decades-long effort to manage that overhead. DCPC's novelty is the claim that these mitigations are *consequences of an architectural constraint* (fact-native semantics) and that removing the constraint eliminates the need for the mitigations. Whether the *residual overhead after all mitigations* is large enough to justify new hardware is the empirical question addressed in Section 14.

PART II: ECONOMIC ANALYSIS AND PROBABILISTIC SUBSTRATES

---

9. Root Cause Analysis: The Cost of Undecided State

9.1 The Activation Threshold Problem

The primary paper identifies an advantage envelope of ~20–25% of workloads. The root cause of this limitation is not logical or representational — it is *economic*.

Maintaining undecided state (markers, coherence metrics, oscillation tracking) is computationally expensive on deterministic classical hardware. Every pre-fact iteration requires evaluating rewrite rules across the marker matrix, computing global reductions, and checking measurement readiness. These costs accumulate regardless of whether the problem benefits from delayed commitment.

DCPC becomes advantageous only once the cost of premature commitment (backtracking, re-propagation, restart overhead) *exceeds* the cost of sustaining ambiguity (marker maintenance, diffusion, coherence computation). This crossover defines a sharp activation threshold.

9.2 The Crossover Function

Let C_commit(x) be the expected cost of classical CDCL on instance x, and C_sustain(x) the expected cost of DCPC. DCPC is advantageous when C_sustain(x) < C_commit(x).

Decomposing:

C_commit(x) = T_BCP(x) + T_overhead(x)

where T_overhead includes backtracking, trail management, restart re-propagation, and clause database maintenance — the Amdahl parameter f times total runtime.

C_sustain(x) = T_rewrite(x) + T_reduce(x) + T_measure(x) where T_rewrite ≈ T_BCP (comparable propagation work), and T_reduce + T_measure is the *sustain cost*.

DCPC wins when: T_overhead(x) > T_reduce(x) + T_measure(x)

On deterministic hardware, T_reduce is substantial: global coherence computation costs O(nm) per iteration. This creates a high activation threshold — DCPC needs T_overhead to be large before the sustain cost is justified.

9.3 Implications

The 20–25% advantage band reflects instances where T_overhead dominates — the thrashing-heavy, restart-intensive regime. For moderate instances where T_overhead is 10–30% of runtime, deterministic DCPC's sustain cost may exceed the overhead it eliminates. This is a real limitation, not a presentation issue.

---

10. Probabilistic Substrates and the RAL Marker Mapping

10.1 Probabilistic Bits (p-bits)

Probabilistic bits are hardware-native stochastic devices with tunable bias and coupling. Unlike classical bits (definite 0/1) or qubits (coherent superposition), p-bits fluctuate stochastically between states with controllable probability:

- State fluctuation is thermally driven, requiring no external energy to maintain
- Bias is tunable: a p-bit with bias β settles to 1 with probability σ(β) = 1/(1 + e^{−β})
- Coupling between p-bits encodes constraint relationships
- Operation at room temperature with CMOS-compatible fabrication

Critically, p-bits make *ambiguity cheap to sustain*. A fluctuating p-bit naturally represents undecided state without the bookkeeping overhead of deterministic marker maintenance.

10.2 Concrete Mapping: p-bits to RAL Markers

Single-variable representation. Each RAL variable x_i is represented by a pair of coupled p-bits (p_i^+, p_i^-) corresponding to x_i=1 and x_i=0 channels. The marker state is decoded from statistical behavior over a short observation window (w clock cycles):

p-bit behavior (over window w)	Decoded marker	Interpretation
p_i^+ predominantly 1, p_i^- predominantly 0	↑ (x_i=1 reinforced)	Strong bias toward true
p_i^+ predominantly 0, p_i^- predominantly 1	↑ (x_i=0 reinforced)	Strong bias toward false
p_i^+ and p_i^- both fluctuating, no dominant state	○ (admissible)	Undecided

p-bit behavior (over window w)	Decoded marker	Interpretation
p_i^+ and p_i^- alternating with period ~w	⇌ (oscillatory)	Cycling, underdetermined
p_i^+ locked at 1, p_i^- locked at 0 (or vice versa)	✓ (coherent)	Stable, ready to commit
Both locked at 0 (no valid state)	∅ (forbidden)	Constraint violation
Anti-correlated with neighbor pair	⊥ (incompatible)	Mutual exclusion active

Constraint encoding via coupling strengths. Constraint edges in the edge-labeled constraint graph (Definition 3.1) correspond to coupling strengths between p-bit pairs:

- Mutual exclusion (mutex-labeled edge): Strong anti-ferromagnetic coupling — when one p-bit locks, the coupled p-bit is suppressed
- Implication (impl-labeled edge): Ferromagnetic coupling — when one p-bit favors a state, the coupled p-bit is biased toward the implied state
- Reinforcement closure (reinf-labeled edges): Self-reinforcing coupling loops — stable configurations strengthen their own bias

Rewrite rules as emergent coupling dynamics. RAL rewrite rules are not executed as explicit LUT lookups on p-bit substrates. Instead, they *emerge* from coupling dynamics:

- R1 (forbidden propagation): A p-bit pair locked in forbidden state drives coupled pairs toward forbidden via strong negative coupling
- R4 (oscillation decay): Thermal damping naturally reduces oscillation amplitude
- R5 (reinforcement closure): Coupled loops of positively biased p-bits converge to locked states through mutual reinforcement

10.3 Why This Reduces Sustain Cost

On deterministic hardware, maintaining marker state requires explicit computation: O(nm) per iteration for rewrite evaluation, O(n) for global reductions, O(nm) for measurement checking.

On p-bit hardware, marker state *persists physically* as p-bit bias and coupling. The sustain cost drops to:

- Rewrite rules: Executed by physics (coupling dynamics), not explicit computation — O(1) time per cycle
- Global reductions: Replaced by emergent collective behavior (mean-field bias)
- Measurement: Threshold detection on mean bias, implementable as a simple analog comparator

The dominant cost shifts from *computation* to *observation* — reading p-bit states to determine measurement readiness. This is dramatically cheaper than explicit marker matrix computation.

10.4 Semantic Preservation

p-bits do not change DCPC's computational semantics:

- Markers remain pre-fact: a fluctuating p-bit is an undecided marker, not a probabilistic fact
- Measurement remains an explicit irreversible boundary
- No asymptotic complexity claims are altered
- The marker alphabet is preserved: p-bit behavior maps onto the same 7-symbol set

---

11. Soft Constraints as First-Class Computational Objects

11.1 The Problem

Deterministic DCPC handles hard constraints natively (forbidden/admissible) but soft or fuzzy constraints require explicit scoring overhead.

11.2 p-bit Solution

On probabilistic substrates, constraint strength maps directly to coupling strength:

- Hard constraint: Strong coupling (high $|J_{ij}|$) — violation rapidly drives p-bits to forbidden
- Soft constraint: Weak coupling (low $|J_{ij}|$) — violation creates bias pressure without forcing forbidden
- Preference: Asymmetric coupling — biases one outcome without excluding alternatives

This eliminates soft-constraint bookkeeping. Constraint strength becomes a *physical parameter* rather than a *symbolic annotation*.

11.3 Implications

Problems with mixed hard/soft constraints — scheduling with preferences, optimization with tolerances, inference with uncertain evidence — become natural DCPC workloads on p-bit substrates, whereas on deterministic substrates they fall below the activation threshold.

---

12. Formal Separation from Annealing and Ising Machines

12.1 The Concern

A reviewer will observe that a p-bit network with tunable couplings encoding constraints *looks like* an Ising machine with a different interpretation layer. The distinction must be formal, not rhetorical.

12.2 Structural Differences

Property	Simulated Annealing / Ising	DCPC with p-bits
Objective	Minimize energy H(σ)	Satisfy constraint admissibility
State semantics	Configuration $\sigma \in \{0,1\}^n$ (fact)	Marker field $M \in \Sigma^{n \times m}$ (pre-fact)
Update rule	Metropolis/Glauber (energy-based)	Rewrite dynamics (admissibility-based)
Temperature schedule	Monotonically decreasing	Not applicable
Termination	Energy convergence	Measurement readiness (Definition 7.2)
Output	Lowest-energy configuration found	Boolean commitment from coherent markers
Feasibility guarantee	None (may output infeasible)	Measurement requires admissibility

12.3 Formal Separation Example

Consider pure 3-SAT with multiple satisfying assignments of equal "quality" — no optimization objective.

Annealing behavior: Ising encoding assigns energy penalties to clause violations. Annealing seeks minimum energy. If the penalty landscape has local minima, annealing may get stuck or output *infeasible* configurations (not all clauses satisfied).

DCPC behavior: Marker dynamics propagate admissibility without an energy function. No temperature, no cooling schedule. Measurement (Definition 7.2) explicitly requires all constraints satisfied before commitment. If no satisfying assignment is found, DCPC reports failure (bounded oscillation or ∅ propagation) rather than infeasible "best effort."

Key distinction: Annealing optimizes and may output infeasible solutions. DCPC satisfies constraints and guarantees feasibility of committed outputs (or reports failure). Different tasks, different correctness guarantees.

12.4 When the Distinction Collapses

For constraint *optimization* problems (MaxSAT, weighted CSP), the distinction narrows. DCPC must be augmented with optimization criteria, and the resulting system resembles annealing more closely. The formal separation holds cleanly for *satisfaction* problems; for optimization, DCPC and annealing become competing approaches distinguished by semantics rather than formal computational properties.

This honest acknowledgment strengthens credibility.

---

13. Revised Advantage Envelope with Quantified Crossover

13.1 Deterministic Crossover Model

Define the *overhead ratio* r(x) for instance x:

$r(x) = T\_overhead(x) / T\_total(x) = f$ (the Amdahl parameter from Section 14)

DCPC is advantageous on deterministic hardware when $r(x) > r^*$, where:

$r^* = (T\_reduce + T\_measure) / T\_total \approx g \approx 0.10–0.15$

Deterministic advantage condition: $r(x) > 0.10–0.15$

This means DCPC is advantageous when the baseline solver spends more than ~10–15% of runtime on commitment-management overhead. Based on the profiling methodology in Section 15, this corresponds to approximately 18–25% of industrial SAT instances.

13.2 Probabilistic Crossover Model

With p-bit substrates, sustain cost drops dramatically (Section 10.3). New introduced overhead:

$g\_p \approx 0.01–0.03$ (dominated by observation, not computation)

Probabilistic advantage condition: $r(x) > 0.02–0.03$

p-bit DCPC becomes advantageous when >2–3% of runtime is commitment overhead — a much lower bar, capturing approximately 35–50% of industrial instances.

13.3 Revised Envelope Table

Substrate	Crossover r*	Advantage band	Basis
Deterministic (FPGA/ASIC)	0.10–0.15	18–25% of instances	Profiling of commitment overhead
Probabilistic (p-bit)	0.02–0.03	35–50% of instances	Reduced sustain cost
Theoretical lower bound	→ 0	Up to ~65%	Zero sustain cost (thermodynamic limit)

13.4 Why the Advantage Remains Bounded

Even with p-bits, DCPC does not become universal. Problems requiring early factual commitment, exact arithmetic, strict determinism, or problems with no constraint structure remain better served by classical architectures.

PART III: EMPIRICAL GROUNDING

14. Empirical Overhead Measurement Protocol

14.1 The Problem

Theorem 11.1 (Amdahl-style bound) gives speedup $\geq 1/(1 - f + g)$, but without measured f and g this is tautological. To transform the bound from framework into evidence, we need an instrumentation protocol any researcher can apply.

14.2 Instrumentation Protocol for f

Target solvers: Kissat (latest release), CaDiCaL (latest release), Glucose 4.x, MiniSat 2.2.

Instrumentation method: Modify solver source to record wall-clock time in each category using high-resolution timers (rdtsc or clock_gettime):

Category	Functions to Instrument	Contributes to
T_BCP	propagate(), unitPropagate()	Neither (comparable in DCPC)
T_conflict	analyze(), analyzeFinal()	Partially f
T_learn	attachClause(), reduceDB()	f
T_trail	undoOne(), cancelUntil(), backtrack()	f
T_restart	restart logic, post-restart re-propagation	f
T_decide	pickBranchLit(), VSIDS updates	Neither
T_mem	Cache miss proxy via perf counters	Partially f

Definition of f:

$f = (T\_conflict + T\_learn + T\_trail + T\_restart) / T\_total$

This counts all time spent managing consequences of premature commitment: conflict analysis, clause database maintenance, trail unwinding, and re-propagation after restarts. It excludes BCP (which DCPC also performs) and decision heuristics (which DCPC replaces with diffusion).

14.3 Measurement Protocol

1. Select benchmarks: SAT Competition 2022–2024 industrial track (publicly available), BMC instances from HWMCC, scheduling/planning encodings from standard repositories
2. For each instance, run instrumented Kissat and CaDiCaL with 5000-second timeout
3. Record per-instance breakdown of all timing categories

4. Compute f per instance
5. Classify instances: low-f (< 0.3), medium-f (0.3–0.6), high-f (> 0.6)

14.4 Estimating g

The introduced overhead g is harder to measure without DCPC hardware. For the software simulator:

- T_rewrite ≈ T_BCP (marker updates analogous to unit propagation)
- T_reduce: Global coherence computation = O(nm) per iteration
- T_measure: Threshold check = O(nm) per measurement attempt

Estimate g = T_reduce / T_total for the simulator. On p-bit hardware, g drops to observation cost (Section 10.3).

14.5 Predicted Outcome

These predictions are *profiling-derived projections based on published data and preliminary instrumentation; exact values to be measured and released per the protocol above.*

Instance family	Predicted f	Speedup (deterministic, g≈0.05)	Speedup (p-bit, g≈0.02)
Random 3-SAT threshold	0.10–0.20	1.1–1.2×	1.1–1.2×
Industrial (low backtracking)	0.25–0.40	1.3–1.7×	1.3–1.6×
Industrial (high backtracking)	0.50–0.70	1.8–2.9×	1.9–3.1×
Thrashing-heavy (BMC, scheduling)	0.70–0.90	2.9–6.7×	3.1–8.3×

14.6 Falsification Criteria

Strong falsification: If f < 0.15 across all industrial instance families, DCPC's projected speedup range is unachievable on deterministic hardware, and the value proposition must rest entirely on parallel depth advantages or p-bit substrates.

Weak falsification: If f < 0.30 across all instance families, the upper range of projected speedups (100–1000×) is not achievable, and DCPC's advantage is limited to modest constant factors (2–5×) — still potentially valuable but not transformative.

Validation: If f > 0.50 on a substantial fraction (>30%) of industrial instances, the primary paper's projections are supported and hardware development is justified.

14.7 Defining "Thrash" Quantitatively

Throughout this companion paper, we use the term "thrash" to describe CDCL behavior on hard instances. To make this precise and reproducible, we define a quantitative metric:

Definition 14.1 (Thrash Ratio). For a CDCL solver run on instance x, define:

thrash(x) = (conflicts × mean_backjump_depth) / total_propagations

This captures the fraction of propagation work that is subsequently *undone* by backjumping. Note that thrash(x) is not bounded by 1; values greater than 1 indicate that more propagation work was discarded than retained, signaling severe re-exploration. A high thrash ratio indicates that the solver repeatedly builds long propagation trails only to discard them.

Classification:

thrash(x)	Classification	DCPC regime
< 0.1	Low thrash (efficient CDCL)	Classical-equivalent
0.1 – 0.4	Moderate thrash	DCPC-competitive (depends on g)
> 0.4	High thrash (heavy re-exploration)	DCPC-advantaged

This metric is used consistently in Sections 1.3, 1.4, and 1.5 to characterize the examples.

---

15. Workload Distribution Methodology

15.1 The Problem

The primary paper's 70/25/5 workload split is cited repeatedly but derived from intuition. We replace it with estimates grounded in three independent data sources.

15.2 Source 1: SAT Competition Instance Classification

The annual SAT Competition (2002–2024) categorizes instances into application/industrial, crafted, and random tracks:

- Industrial/application: ~40% of submitted instances, ~70% of computational effort
- Crafted (structured, non-random): ~35% of instances, ~20% of effort
- Random: ~25% of instances, ~10% of effort

Within industrial instances, profiling studies (Biere 2017, Froleyks et al. 2021) report 30–60% of solver runtime on hard instances consumed by backtracking and restart overhead.

15.3 Source 2: Data Center Workload Surveys

Publicly available data center characterizations (Kanev et al. 2015, Google; Sriraman & Dhanotia 2020, Facebook):

- ~50–60% of cycles: web serving, database, memcached (no constraint structure)
- ~15–25% of cycles: ML training and inference (no constraint structure for DCPC)
- ~5–10% of cycles: compilation, static analysis, verification (constraint-rich)
- ~5–10% of cycles: scheduling, planning, optimization, configuration (constraint-rich)
- ~1–3% of cycles: scientific simulation (mixed)

15.4 Source 3: EDA Industry Benchmarks

Electronic design automation is overwhelmingly constraint-dominated. Industry estimates suggest 80–90% of EDA computation is constraint satisfaction or optimization.

15.5 Revised Estimates

By compute cycles, general data center:

Category	Share	Basis
Classical-equivalent	78–85%	Web, DB, ML, streaming, rendering
DCPC-advantaged (deterministic)	12–18%	Verification, scheduling, optimization, EDA, planning
DCPC-advantaged (additional with p-bits)	5–10%	Moderate-ambiguity below deterministic threshold
Quantum-advantaged	0.5–2%	Quantum simulation (projected with FTQC)

By compute cycles, constraint-heavy industries (EDA, logistics, verification):

Category	Share	Basis
Classical-equivalent	30–45%	Data movement, I/O, preprocessing
DCPC-advantaged	50–65%	Core constraint solving, optimization, verification
Quantum-advantaged	1–5%	Quantum simulation of materials (projected)

15.6 Presentation Recommendation

Replace the single 70/25/5 estimate with domain-specific breakdowns. Present the general data center estimate as the conservative baseline and EDA/verification as the targeted deployment case. Avoid repeating a single set of numbers throughout the paper.

---

16. Unified Evaluation Path

16.1 Three-Phase Protocol

Phase 1: Overhead Profiling (0–6 months)

- Instrument Kissat and CaDiCaL per Section 14
- Measure f and thrash(x) across SAT Competition industrial instances
- Classify instances by overhead profile
- Determine activation threshold empirically
- Release instrumentation patches, raw data, and analysis scripts

Phase 2: Software Simulation (6–18 months)

- Implement RAL marker dynamics simulator (per primary paper §12.5)
- Implement p-bit emulation layer with configurable sustain cost
- Compare propagation-equivalent work (not wall-clock time) against CDCL baselines on the three example families from Section 1
- Validate predictions from Section 14.5

Phase 3: Hardware Prototyping (18–36 months)

- FPGA implementation of deterministic DCPC (marker ALU + reduction network)
- Measure classical-mode overhead (Section 7.3 estimate: 1.5–3×)
- p-bit integration (if stochastic MRAM or equivalent becomes available)
- Wall-clock comparison against Kissat/CaDiCaL on industrial benchmarks

16.2 Success Criteria

Phase	Success	Partial Success	Failure
Profiling	$f > 0.50$ on >30% of instances	$f > 0.30$ on >30%	$f < 0.15$ universally
Simulation	Fewer propagation-equivalent passes on structured instances	Comparable passes, fewer restarts	More passes than CDCL
Hardware	Wall-clock advantage on high-f instances	Comparable performance	Slower than Kissat across all categories

16.3 What Failure Would Mean

If all three phases fail, the conclusion is that modern CDCL engineering has already captured most available overhead reduction, and DCPC's architectural premise requires revision. This is a genuinely possible outcome.

If Phase 1 succeeds but Phase 2/3 fail, the overhead exists but DCPC's marker dynamics are not the right mechanism to exploit it. Alternative pre-fact architectures might succeed where DCPC does not.

If Phase 1 fails, the entire premise is undermined: modern solvers already manage commitment overhead efficiently, and the problem DCPC solves is smaller than projected.

17. Conclusion

This companion paper does not retract or weaken the architectural claims of the primary DCPC proposal. Instead, it explains the mechanism that governs when those claims activate and how far they extend in practice. The primary paper establishes that delayed-commit computation eliminates a specific and measurable class of overhead in fact-native architectures. This work shows that the observed advantage envelope follows directly from an economic crossover between the cost of premature commitment and the cost of sustaining undecided state.

The technical corrections in Part I sharpen the primary paper's formal apparatus. Replacing the original worked examples with resolution-hard XOR-parity, BMC ripple-carry, and cardinality instances ensures that the demonstrated CDCL thrashing is genuine and reproducible — not an artifact of instances too simple to stress any solver. The constructive concentration result (Proposition 2.1) completes the interference story that Theorem 4.1 began: destructive interference prunes bad traces, constructive concentration amplifies good ones, and together they provide the structural analogue of quantum interference without amplitudes. The edge-labeled constraint graph formalization, coherence weight sensitivity analysis, and oracle reattribution correct specific underspecifications and misframings that would otherwise invite justified reviewer objection.

The novelty delineation in Section 8 addresses the single most dangerous critique the primary paper faces. Modern CDCL solvers have spent decades evolving toward minimal-commitment behavior through non-chronological backtracking, phase saving, inprocessing, and restart strategies. Each of these partially mitigates the overhead of early commitment. DCPC's claim is not that these mitigations are ineffective — they are impressively effective — but that they are *engineering responses to an architectural constraint*. Removing the constraint (pre-fact semantics) eliminates the need for the responses. Whether the residual overhead after all mitigations justifies new hardware is an empirical question, and Part III specifies exactly how to answer it.

The economic analysis in Part II identifies the activation threshold as a crossover condition, not a fixed boundary. On deterministic substrates, DCPC activates when the Amdahl parameter f exceeds approximately 0.10–0.15. On probabilistic substrates (p-bits), the sustain cost drops and the threshold falls to approximately 0.02–0.03, expanding the advantage envelope from ~20% to ~35–50% of industrial instances. Critically, the p-bit extension preserves DCPC's semantics: markers remain pre-fact, measurement remains irreversible, and no asymptotic claims change. What changes is the economics — and in engineering, economics determines adoption.

The empirical protocol in Part III converts the primary paper's analytical projections into falsifiable predictions with explicit success and failure criteria. If profiling reveals $f < 0.15$ across industrial instances, the premise is undermined. If $f > 0.50$ on a substantial fraction, hardware development is justified. This is the standard the primary paper should be held to, and this companion provides the instrumentation to apply it.

Taken together, the primary DCPC paper and this companion establish a coherent, bounded, and testable architectural framework for constraint-dominated computation. The primary paper provides the architecture. This companion provides the corrections, the economic model, and the empirical scaffold. Neither is complete without the other.

APPENDICES

---

18. Revised Terminology Recommendations

18.1 The "Taylor Limit"

Self-naming a theoretical construct in one's own paper is typically received negatively by reviewers. The construct $\Lambda\_T = e\^(S\_max / k\_B)$ is a straightforward application of the Bekenstein bound to computational distinguishability. Recommended alternatives:

- **Entropy-bounded distinguishability limit (EBDL)** — descriptive, transparent
- **Admissible resolution bound** — shorter, emphasizes role in theory
- **Holographic computation bound** — connects to established physics

If "Taylor Limit" is retained for the broader VERSF framework, introduce with: *"We refer to this bound as the entropy-bounded distinguishability limit (EBDL). Within the broader VERSF framework, this quantity is termed the Taylor Limit; we use the descriptive name here for generality."*

18.2 Other Terminology

- "Assembly traces" → Consider "constraint traces" or "admissibility traces" (clearer outside VERSF)
- "Fact regime" / "pre-fact regime" → Retain; strongest terminological contribution
- "Resonant Assembly Language" → "Resonant" may confuse physicists (no resonance phenomenon). Consider "Constraint Assembly Language" (CAL) for the computational audience.

---

19. Paper-Splitting Strategy for Journal Submission

Paper A: Core Architecture (target: Journal of the ACM, FOCS/STOC, or SAT Conference)

- Model definition, marker dynamics, interference (primary §2–4, with Proposition 2.1 from this companion)
- Worked examples (Section 1 of this companion, replacing primary §5)
- Measurement criterion (primary §7, with sensitivity analysis from Section 4)
- CDCL native execution (primary §10, with novelty delineation from Section 8)
- Comparison and scope (primary §11, with parallel comparator from Section 6, subsumption from Section 7)
- Implementation and evaluation (primary §12, with overhead protocol from Section 14)
- Revised rewrite rules (Section 3 of this companion, replacing primary Appendix A)

Paper B: Physical Admissibility and Entropy Bounds (target: Foundations of Physics or Physical Review D)

- Taylor Limit / EBDL (primary §8)
- Conceptual significance (primary §15)
- Connection to VERSF framework
- Expanded derivation from Bekenstein/holographic bounds

Paper C: Taxonomy, Applications, and Substrate Extensions (target: ACM Computing Surveys or technical report)

- Problem taxonomy (primary §13)
- Applications (primary §14)
- Oracle results reframed (primary §9, with Section 5 corrections)
- Probabilistic substrates (Sections 9–13 of this companion)
- Workload distribution (Section 15 of this companion)

---

20. Integration Guide for Revised Primary Paper

Companion Section	Replaces/Augments	Priority
1. Nontrivial examples (XOR-parity, BMC, cardinality)	Replace §5 entirely	Critical
2. Constructive concentration (Proposition 2.1)	Add as Proposition 4.3 in §4	High
3. Edge-labeled constraint graph + revised rules	Replace Appendix A	High
4. Coherence sensitivity	Add subsection to §7	Medium
5. Oracle attribution (move earlier as scope correction)	Reframe §9 with disclaimers	High
6. Parallel comparator (under portfolio parallelism)	Revise Theorem 11.3 and proof	High
7. Subsumption qualification (engineering estimate, to be measured)	Revise §11.1, §11.3	Medium
8. CDCL novelty delineation	Expand §3.3 and §11.5.1	Critical

Companion Section	Replaces/Augments	Priority
9. Root cause / activation threshold	Add new subsection to §11 or §12	High
10. p-bit → RAL marker mapping	Add new section or appendix	High
11. Soft constraints	Integrate into §14 applications	Medium
12. Annealing separation	Add to §11.2 or new subsection	High
13. Crossover model	Replace §13.11 estimates	High
14. Overhead protocol + thrash metric (Definition 14.1)	Expand §11.6	Critical
15. Workload methodology	Replace §13.11 with sourced estimates	High
16. Unified evaluation	Expand §12.5	Medium
17. Terminology	Apply throughout	Medium

---

21. References

SAT Solver Profiling and Competition Analysis

- Biere, A. (2017). "CaDiCaL, Kissat, Paracooba, Plingeling, and Treengeling entering the SAT Competition 2020." SAT Competition 2020.
- Froleyks, N., Heule, M., Iser, M., Järvisalo, M., & Suda, M. (2021). "SAT Competition 2020." *Artificial Intelligence*, 301, 103572.
- Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., & Suda, M. (2023). "Proceedings of SAT Competition 2023." University of Helsinki.

Data Center Workload Characterization

- Kanev, S., Darago, J.P., Hazelwood, K., et al. (2015). "Profiling a warehouse-scale computer." *Proceedings of ISCA '15*, 158–169.
- Sriraman, A. & Dhanotia, A. (2020). "Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale." *Proceedings of ASPLOS '20*, 733–750.

Parallel SAT Solving

- Biere, A., & Fleury, M. (2020). "Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022." SAT Competition 2022.
- Heule, M.J., Kullmann, O., Wieringa, S., & Biere, A. (2012). "Cube and conquer: Guiding CDCL SAT solvers by lookaheads." *Hardware and Software: Verification and Testing*, 50–65.
- Balyo, T., Sanders, P., & Sinz, C. (2015). "HordeSat: A massively parallel portfolio SAT solver." *SAT 2015*, 156–172.

Non-Chronological Backtracking and Phase Saving

- Marques-Silva, J.P. & Sakallah, K.A. (1999). "GRASP: A search algorithm for propositional satisfiability." *IEEE Transactions on Computers*, 48(5), 506–521.
- Pipatsrisawat, K. & Darwiche, A. (2007). "A lightweight component caching scheme for satisfiability solvers." *SAT 2007*, 294–299.

Inprocessing

- Järvisalo, M., Heule, M.J.H., & Biere, A. (2012). "Inprocessing rules." *IJCAR 2012*, 355–370.

Probabilistic Computing and p-bits

- Camsari, K.Y., Faria, R., Sutton, B.M., & Datta, S. (2017). "Stochastic p-bits for invertible logic." *Physical Review X*, 7(3), 031014.
- Borders, W.A., Pervaiz, A.Z., Fukami, S., Camsari, K.Y., Ohno, H., & Datta, S. (2019). "Integer factorization using stochastic magnetic tunnel junctions." *Nature*, 573, 390–393.
- Kaiser, J. & Datta, S. (2022). "Probabilistic computing with p-bits." *Applied Physics Letters*, 119, 150503.

EDA and Verification Workloads

- Vizel, Y., Weissenbacher, G., & Malik, S. (2015). "Boolean satisfiability solvers and their applications in model checking." *Proceedings of the IEEE*, 103(11), 2021–2035.

Entropy Bounds and Holographic Principle

- Bekenstein, J.D. (1981). "Universal upper bound on the entropy-to-energy ratio for bounded systems." *Physical Review D*, 23(2), 287.
- Bousso, R. (2002). "The holographic principle." *Reviews of Modern Physics*, 74(3), 825.

Constraint Propagation and Query Complexity

- Bessière, C. (2006). "Constraint propagation." *Handbook of Constraint Programming*, 29–83.
- Beals, R., Buhrman, H., Cleve, R., Mosca, M., & de Wolf, R. (2001). "Quantum lower bounds by polynomials." *Journal of the ACM*, 48(4), 778–797.

Annealing and Ising Machines

- Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). "Optimization by simulated annealing." *Science*, 220(4598), 671–680.
- Johnson, M.W., et al. (2011). "Quantum annealing with manufactured spins." *Nature*, 473, 194–198.

- Mohseni, N., McMahon, P.L., & Byrnes, T. (2022). "Ising machines as hardware solvers of combinatorial optimization problems." *Nature Reviews Physics*, 4, 363–379.

XOR/Parity Hardness for Resolution

- Ben-Sasson, E. & Wigderson, A. (2001). "Short proofs are narrow — resolution made simple." *Journal of the ACM*, 48(2), 149–169.
- Urquhart, A. (1987). "Hard examples for resolution." *Journal of the ACM*, 34(1), 209–219.

End of unified companion paper.