

PACT: A Runtime Contract for Sparse-Topology Updates in GPU-Accelerated Timing ECOs

Haichuan Liu¹, Zizheng Guo^{1,2}, Zuodong Zhang², Haoyi Zhang¹,
Chunyuan Zhao¹, Kairong Guo¹, Shenglu Hua¹, Guannan Guo^{1*}, and Yibo Lin^{1,2,3*}

¹ School of Integrated Circuits, Peking University, Beijing, China ² Institute of EDA, Peking University, Wuxi, China

³ Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China

{haichuanliu, hy.zhang, zhaochunyuan, krguo25, huashenglu26}@stu.pku.edu.cn,
{gzz, guannan.guo, yibolin}@pku.edu.cn, zhangzd@pkueda.org.cn

Abstract

Timing ECO flows repeatedly modify circuit topology, yet GPU-accelerated Static Timing Analysis (STA) engines built on static sparse layouts commonly rebuild and resynchronize the GPU-visible topology after each edit. Existing GPU-accelerated STA work instead co-designs dynamic representations with application-specific kernels, exposing representation assumptions to consumers and limiting reuse with CSR-based analyses. We present **PACT** (Private Atomic Committed Topology), a runtime contract that keeps the mutation representation runtime-private behind a representation-independent committed-traversal contract. GPU traversal consumers observe only a committed CSR-compatible view. PACT maintains this view through a contract-preserving localized commit realization that maps objects created by or incident to each edit batch into affected CSR rows, rematerializes each complete affected CSR row, and exposes the new view after synchronization. Dependency-ordered contract composition applies the same contract to explicit netlist/parasitic edits and timing-state edits derived from their committed frontiers. The selected realization achieves 3.59× the geometric-mean topology-maintenance throughput of a reimplemented Dual-CSR baseline. In an in-house CPU-GPU STA engine, PACT delivers 2.72× and 2.96× end-to-end speedups over full rebuild under two ECO schedules with identical complete timing outputs. Scalability studies further delimit its measured operating range across host thread counts and update-batch sizes.

CCS Concepts

• **Hardware** → **Static timing analysis**.

Keywords

static timing analysis, engineering change orders, GPU acceleration, incremental sparse topology

ACM Reference Format:

Haichuan Liu, Zizheng Guo, Zuodong Zhang, Haoyi Zhang, Chunyuan Zhao, Kairong Guo, Shenglu Hua, Guannan Guo, and Yibo Lin. 2027. PACT: A Runtime Contract for Sparse-Topology Updates in GPU-Accelerated Timing ECOs. In *32nd Asia and South Pacific Design Automation Conference (ASPDAC '27)*, January 25–28, 2027, Tokyo, Japan. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3840404.3848150>

*Corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License. ASPDAC '27, Tokyo, Japan

© 2027 Copyright held by the owner/author(s). ACM ISBN 979-8-4007-2923-2/2027/01

<https://doi.org/10.1145/3840404.3848150>

Localized ECO transaction

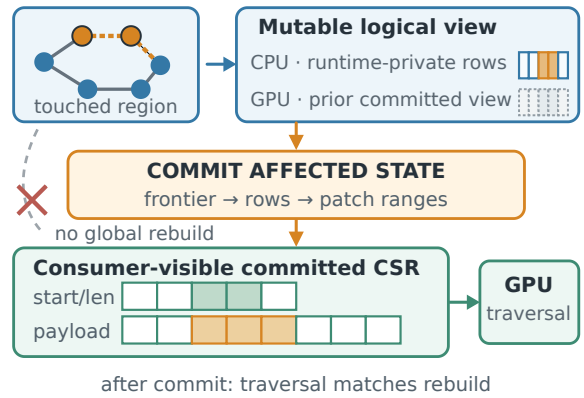


Figure 1: PACT's representation-ownership and visibility boundary. Pending rows belong to the runtime-private mutation representation and never enter the committed-traversal contract. GPU traversal consumers access only the old or new committed view through a CSR-compatible row-range accessor; commit is the sole atomic visibility transition.

'27), January 25–28, 2027, Tokyo, Japan. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3840404.3848150>

1 Introduction

At advanced technology nodes, timing closure proceeds through repeated analyze–fix–validate cycles as interconnect-dominated delay, process variation, and restrictive design rules complicate implementation [15]. Each cycle relies on static timing analysis (STA) not only for final signoff, but also for feedback on every local change [3, 12].

Accelerating this repeated feedback has motivated GPU systems for timing-driven placement [6, 19], routing [30, 31], and timing kernels [9, 11, 23]. These systems reduce full-run cost, but timing ECO flows are governed by feedback latency rather than peak full-chip throughput. This latency can remain high even when a post-placement optimization touches only a small neighborhood, because the next GPU-accelerated analysis may still require global sparse reconstruction and host–device resynchronization [2].

Such global reconstruction is precisely the redundancy that CPU incremental timers avoid by updating only affected timing cones [13, 14, 16, 24]. Carrying the same reuse into topology-changing GPU flows is harder because GPU kernels favor stable compressed sparse

row (CSR)-like layouts, whereas insertions, deletions, and reconnections shift offsets, migrate payloads, and invalidate synchronized state. This representation conflict leaves two unsatisfactory choices: rebuilding after every localized ECO discards reusable state, while exposing mutability directly adds traversal indirection and couples every consumer to a dynamic layout.

Existing EDA systems resolve different parts of this conflict. Value refresh in HeteroExcept leaves topology unchanged, whereas INSTA and HeteroSTA retain rebuild-oriented layouts [10, 11, 23]. For structural updates, a prior dynamic-CSR representation divides adjacency between base and auxiliary storage. IncreGPUSTA co-designs this representation with customized CSR-update, mixed-subgraph levelization, and timing-propagation kernels [22]. These kernels interpret base/auxiliary storage directly; reuse therefore requires adapting traversal, update, and synchronization interfaces. Similar representation coupling appears in workload-specific systems such as iG-kway and iHyperG [17, 18].

This coupling extends to dynamic-graph frameworks with adjacency-list, hash-based, segmented, and mutable-CSR organizations [5, 7, 8, 21, 25, 28, 29, 32]. These organizations store mutations efficiently, but their execution-facing layouts can require representation-specific EDA traversal and synchronization, creating an integration burden. This integration burden exposes a missing timing-ECO boundary between runtime-owned mutation representation and consumer-visible committed traversal.

PACT provides this missing boundary as a runtime contract for GPU-accelerated timing analysis. The runtime contract keeps mutation representation runtime-private, while GPU traversal consumers observe only a committed CSR-compatible view. PACT makes three contributions:

- (1) **A representation-independent committed-traversal contract.** PACT keeps mutation representation runtime-private and exposes only a committed CSR-compatible view. Three invariants specify what GPU traversal consumers observe, when it atomically changes, and what state the runtime refreshes.
- (2) **A contract-preserving localized commit realization.** For each edit batch, PACT identifies graph objects created by or incident to the edits, maps these objects to affected CSR rows, rematerializes each complete row from the mutable logical view, and exposes the staged metadata/payload updates after synchronization.
- (3) **Dependency-ordered contract composition.** PACT applies the same layer-local contract to caller-supplied netlist and parasitic batches and to the timing batch derived from their committed frontiers. This composition lets the GPU timing consumer use the committed timing view without interpreting any upstream mutation representation.

We first evaluate the localized realization, which delivers 3.59× the geometric-mean topology-maintenance throughput of a reimplemented Dual-CSR baseline [22]. We then evaluate dependency-ordered composition in an in-house CPU-GPU STA engine, obtaining 2.72× and 2.96× end-to-end gate-cloning speedups with identical complete timing outputs. Finally, thread and buffer-insertion studies delimit the measured operating range, retaining 4.05× and 1.74× speedups at 128 threads and 10^5 buffers per round.

The rest of this paper is organized as follows. Section 2 reviews sparse traversal and timing ECOs; Section 3 presents PACT; Section 4 reports the experiments; and Section 5 concludes the paper.

2 Background

2.1 CSR Traversal and Dual-CSR Updates

Compressed Sparse Row (CSR) stores adjacency in an offset array and a contiguous payload array. These arrays delimit each vertex’s neighbors as one continuous range. Each range supports sequential GPU access and compact $O(nnz)$ traversal. This compact layout makes structural updates costly: insertion into a full row shifts later payload and offsets, while deletion leaves holes or requires compaction.

This update cost motivates separating stable connectivity from overflow. Dual-CSR realizes this separation through a base CSR, an auxiliary CSR, and a vertex-to-auxiliary-row map [22]. The preallocated base appends edges for new vertices and reuses invalid entries in existing rows. Edges that exceed the base enter a reconstructed auxiliary CSR through the map. The auxiliary CSR consequently makes traversal interpret both base and auxiliary representations.

Dual-CSR provides IncreGPUSTA’s storage layer. The complete pipeline additionally builds on per-operation CPU lists of inserted and removed forward/reverse edges. These lists drive customized kernels for removal, replacement, reverse-edge updates, base appends, and auxiliary transfers. The updated representation then feeds mixed-subgraph levelization and incremental timing propagation kernels.

2.2 Topology-Changing ECOs

Physical-synthesis timing closure applies value-only resizing and topology-changing buffer insertion, gate cloning, pin reconnection, and rewiring [1, 2, 20, 26]. Resizing changes delay, slew, and load but not connectivity. Topology ECOs instead create or reconnect objects and timing arcs: buffer insertion splits a connection, while gate cloning duplicates a driver and repartitions its sinks. Both produce localized connectivity changes. These changes require the corresponding sparse netlist, parasitic, and timing representations to be refreshed before GPU timing traversal resumes.

3 The PACT Runtime Contract

Topology-changing ECOs therefore require mutable CPU-side state and a reusable GPU row-range traversal interface. PACT reconciles these requirements by separating runtime-private mutation representation from a committed CSR-compatible view (Fig. 1). This section defines the representation-independent committed-traversal contract, presents its contract-preserving localized commit realization, and establishes dependency-ordered contract composition across timing-ECO state layers.

3.1 Representation-Independent Committed-Traversal Contract

At committed version t , PACT represents a managed sparse graph with three states:

$$\Sigma_t = (\mathcal{L}_t, \mathcal{C}_t, \mathcal{G}_t), \quad (1)$$

where $\mathcal{L}_t$ is the runtime-private mutable logical view, $\mathcal{C}_t$ is the committed host-side materialized state, and $\mathcal{G}_t$ is the committed CSR-compatible view visible to GPU traversal consumers. Only $\mathcal{G}_t$ is exposed through the committed-traversal contract. Any realization-specific state—including logical buffers, indexing structures, placement decisions, and patch ranges—remains runtime-private. An edit batch updates the mutable logical view before commit:

$$E_t = (e_1, \dots, e_m), \quad (2)$$

$$e_i \in \{\text{create, delete, connect, disconnect, replace}\},$$

where E_t is the complete batch at the public commit boundary. It changes only private logical state before commit and produces

$$(\mathcal{L}_t, E_t) \xrightarrow{\text{TxnApply}} (\mathcal{L}_{t+1}^-, F_t), \quad (3)$$

where $\mathcal{L}_{t+1}^-$ is the batch-final staged logical view and F_t its edit frontier. Superscript $-$ marks a pending commit version: $\mathcal{C}_{t+1}^-$ is the staged host materialization and $\mathcal{G}_{t+1}^-$ the staged GPU traversal view. Commit exposes $\Sigma_{t+1} = (\mathcal{L}_{t+1}, \mathcal{C}_{t+1}, \mathcal{G}_{t+1})$ only after publication and synchronization complete.

The edit frontier F_t contains objects created by or incident to E_t in the active object domain V . Each frontier object v maps through $\text{row}(v)$ to an affected CSR row:

$$F_t = \{v \in V \mid \exists e_i \in E_t, v \text{ is created by or incident to } e_i\}, \quad (4)$$

$$R_t = \{\text{row}(v) \mid v \in F_t\}. \quad (5)$$

Thus, R_t is the affected CSR-row set. The reverse index deduplicates and orders R_t . For each $r \in R_t$, the committed view supplies its payload through a representation-specific accessor $\text{RowRange}_{\mathcal{G}}(r)$. This accessor, rather than a fixed storage layout, defines CSR compatibility. Let $\text{Nbr}_{\mathcal{G}}(u)$ be the neighbor sequence from $\mathcal{G}$, $\text{rebuild}(\mathcal{L})$ a full CSR reconstruction, and $\text{Traverse}(u)$ the consumer-observed sequence. A commit satisfies

$$\text{Nbr}_{\mathcal{G}_{t+1}}(u) = \text{Nbr}_{\text{rebuild}(\mathcal{L}_{t+1})}(u), \quad \forall u, \quad (6)$$

$$\text{Traverse}(u) = \begin{cases} \text{Nbr}_{\mathcal{G}_t}(u), & \text{before expose,} \\ \text{Nbr}_{\mathcal{G}_{t+1}}(u), & \text{after expose,} \end{cases} \quad (7)$$

Together, the obligations define the committed-traversal contract. *Traversal equivalence* specifies what GPU traversal consumers observe, and *commit visibility* specifies an atomic transition: traversal observes either the prior committed view or the new one, never an intermediate state. *Locality-bounded refresh* restricts rematerialization to the affected CSR rows in R_t , although physical publication may be wider.

3.2 Contract-Preserving Localized Commit Realization

Write $\Gamma_{\mathcal{L}}(r)$ for the complete logical payload of row r . Let $\Delta_m = \bigcup_{r \in R_t} \Delta_r^m$ and $\Delta_p = \bigcup_{r \in R_t} \Delta_r^p$ collect the per-row metadata and payload patches, respectively. Algorithm 1 realizes these obligations in one commit path.

A commit applies E_t to produce the staged logical view $\mathcal{L}_{t+1}^-$ and edit frontier F_t . The frontier maps through the reverse index to the affected CSR-row set R_t . This set bounds rematerialization to complete logical row payloads $Y_r = \Gamma_{\mathcal{L}_{t+1}^-}(r)$, satisfying locality-bound refresh. A placement policy π_t writes each payload into the staged committed view and emits metadata/payload patches

Algorithm 1 PACTCOMMITANDSYNC: Contract-Preserving Localized Commit

Require: visible state $\Sigma_t = (\mathcal{L}_t, \mathcal{C}_t, \mathcal{G}_t)$; edit batch E_t

Ensure: committed state Σ_{t+1}

```

1:  $(\mathcal{L}_{t+1}^-, F_t) \leftarrow \text{TxnApply}(\mathcal{L}_t, E_t)$ 
2:  $R_t \leftarrow \{\text{row}(v) \mid v \in F_t\}$  ▷ affected CSR rows
3:  $(\mathcal{C}_{t+1}^-, \mathcal{G}_{t+1}^-) \leftarrow \text{BEGINSTAGEDPUBLICATION}(\mathcal{C}_t, \mathcal{G}_t)$ 
4:  $\pi_t \leftarrow \text{SELECTPLACEMENT}(R_t, \mathcal{C}_{t+1}^-)$  ▷ append or compact
5: for  $r \in R_t$  do
6:    $Y_r \leftarrow \Gamma_{\mathcal{L}_{t+1}^-}(r)$  ▷ logical row payload
7:    $(\mathcal{C}_{t+1}^-, \Delta_r^m, \Delta_r^p) \leftarrow \text{PLACEROW}(r, Y_r, \pi_t, \mathcal{C}_{t+1}^-)$ 
8:    $(D_m, D_p) \leftarrow \text{COALESCE}(\Delta_m, \Delta_p)$ 
9:    $\text{PUBLISHMETADATA}(\mathcal{G}_{t+1}^-, D_m)$  ▷ scatter / bounded bulk
10:   $\text{PUBLISHPAYLOAD}(\mathcal{G}_{t+1}^-, D_p)$  ▷ selective / contiguous bulk
11:  $\text{SYNCHRONIZEPUBLICATION}(\mathcal{G}_{t+1}^-, D_m, D_p)$ 
12:  $\Sigma_{t+1} \leftarrow \text{EXPOSECOMMITTEDVIEW}(\mathcal{L}_{t+1}^-, \mathcal{C}_{t+1}^-, \mathcal{G}_{t+1}^-)$ 
13: return  $\Sigma_{t+1} = (\mathcal{L}_{t+1}, \mathcal{C}_{t+1}, \mathcal{G}_{t+1})$ 

```

Δ_r^m, Δ_r^p . Because every affected CSR row is rematerialized from $\mathcal{L}_{t+1}^-$, the resulting committed view satisfies traversal equivalence.

The metadata and payload patch sets are separately coalesced into global dirty ranges

$$D_m = \{(a_i, \ell_i)\}_{i=1}^{k_m}, \quad D_p = \{(b_j, \ell_j)\}_{j=1}^{k_p}, \quad (8)$$

where each pair gives the start and length of a dirty metadata or payload range. D_m and D_p drive publication into the staged GPU view $\mathcal{G}_{t+1}^-$. The staged view becomes consumer-visible only after synchronization and expose, satisfying atomic commit visibility.

Publication occurs between analysis rounds, after prior GPU traversal completes. `BEGINSTAGEDPUBLICATION` starts this exclusive update phase; `SYNCHRONIZEPUBLICATION` completes metadata/payload transfers and update kernels before `EXPOSECOMMITTEDVIEW` permits the next traversal.

This commit path also defines the locality boundary. Row rematerialization visits only the logical payload of R_t , and coalescing bounds ordinary publication by the resulting metadata and payload ranges. Compaction or bulk fallback may widen physical movement, and the work approaches rebuild scale as $|R_t| \rightarrow |V|$. Within this boundary, placement and publication still depend on the selected internal realization.

3.3 Internal Realizations and Memory Organizations

The placement map π_t acts on an internal committed-side realization. Five candidate realizations appear in Fig. 2; each satisfies the committed-traversal contract but organizes physical storage, indexing, and free space differently.

To compare them under one notation, represent each memory organization as $\mathcal{M} = (\mathcal{B}, \mathcal{J}, \mathcal{S})$, where $\mathcal{B}$ denotes physical storage, $\mathcal{J}$ maps a logical row r to its committed location(s), and $\mathcal{S}$ records reusable slack or overlay layers. Equivalently, each logical row has a realization set $\Pi(r) = \{(b, \omega, \ell) \mid (b, \omega, \ell) \rightsquigarrow r\}$ of slices contributing to row r . The candidates differ primarily in $|\Pi(r)|$: more slices absorb mutations with less data movement, but add traversal-side indirection.

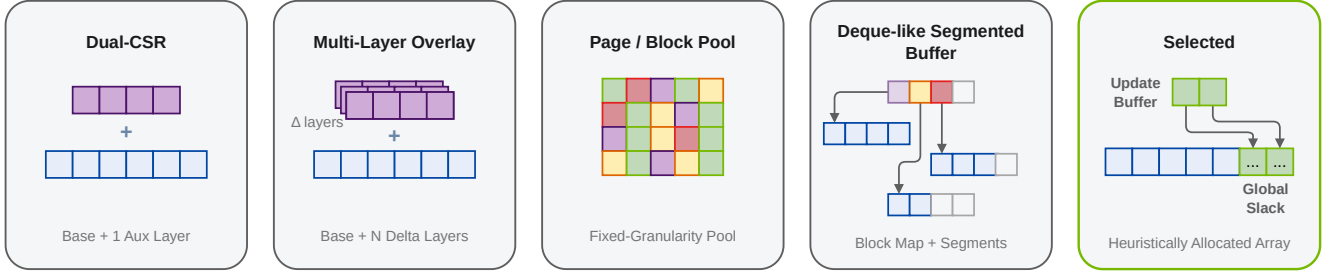


Figure 2: Internal realizations under one committed-traversal contract. Dual-CSR consumers interpret base/auxiliary slices; other organizations add different indirection. The selected flat storage keeps staging private and committed rows usually contiguous under the same contract.

- (1) **Dual-CSR** [22] maintains one base and one auxiliary array with no free-list bookkeeping ($S = \emptyset$), so $|\Pi(r)| \leq 2$. Updates reuse invalid base entries and reconstruct the auxiliary CSR for remaining insertions.
- (2) **Multi-layer overlay** generalizes Dual-CSR by stacking K delta layers atop a base array. $|\Pi(r)|$ grows with K , improving update absorption but increasing the slices consulted per traversal.
- (3) **Page/block pool** divides storage into fixed-granularity pages tracked by a free list $\mathcal{F}_{\text{page}}$. A row that outgrows its page spans multiple pages, trading internal fragmentation and pointer-chasing traversal for simplified free-space management.
- (4) **Deque-like segmented buffer** links variable-sized segments; rows grow by appending new segments. This organization incurs larger $|\Pi(r)|$ than Dual-CSR but smaller fragmentation than page pools.
- (5) **Flat array with global slack** stores each row as one contiguous slice ($|\Pi(r)| = 1$) and absorbs growth through global slack S_{global} rather than per-slab free lists [4, 27]. $\text{RowRange}_g(r)$ reduces to $[\text{rowStart}[r], \text{rowStart}[r] + \text{rowLength}[r]]$, the simplest possible accessor.

PACT selects the flat array because it preserves one contiguous range per committed row, whereas the other candidates require multiple slices or additional indexing. GPU traversal consumers see only RowRange_g , not the internal buffer, placement, or patch mechanics. The committed-traversal contract remains invariant across these choices, so Dual-CSR or another realization can implement the same accessor and obligations. Changing the realization therefore leaves dependency-ordered contract composition unchanged.

3.4 Dependency-Ordered Contract Composition

Applying PACT to a timing-ECO flow composes the layer-local contract across heterogeneous netlist (V_n), parasitic (V_p), and timing-graph (V_t) domains. Each domain maintains independent logical and committed state; a public commit request orchestrates them in dependency order, as Fig. 3 shows.

Explicit netlist and parasitic edit batches E_t^n, E_t^p commit first, producing frontiers F_t^n, F_t^p . These frontiers derive the timing batch $E_t^r = \text{DeriveTimingEdits}(F_t^n, F_t^p)$, which commits through the same path. Only after the timing-layer commit does the GPU-accelerated analysis engine consume the new committed view.

Dependency-ordered contract composition

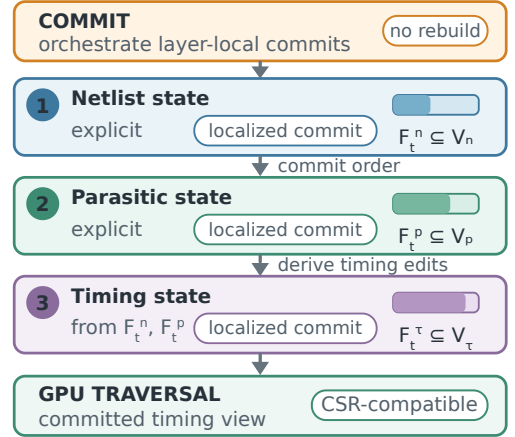


Figure 3: Dependency-ordered composition of the layer-local contract. Netlist/parasitic edits are explicit; timing edits derive from both frontiers. Repeated localized-commit blocks denote contract reuse under one public orchestration request. Longer bars indicate relative frontier extent.

This timing-layer commit uses the same mechanism as the two upstream commits (Algorithm 1); only batch provenance differs. The progressively longer bars in Fig. 3 indicate relative frontier extent rather than absolute size. This layer-local composition remains independent of the realization policies left open by Algorithm 1.

3.5 Optimization Policies of the Selected Realization

Algorithm 1 leaves four realization choices open: when to compact versus append (line 4), how to publish metadata (line 9), whether to reuse freed slots (line 7), and when to prefetch (lines 5–7). Bounded offline exploration selected the following fixed policies; no policy search occurs at runtime.

The selected realization uses four policies:

- (1) **Density-aware compaction** (line 4). `SelectPlacement` chooses between compact and append using a density-gated slack comparison. Let $\rho_t = |R_t|/|V|$, B_{move} be the suffix payload migrated by compaction, B_{new} the unavoidable

Table 1: Statistics of Tau Benchmark Circuits.

Benchmark	#Gates	#Nets	#Pins
aes_core	22,938	23,199	66,221
vga_lcd_iccad	259,067	259,152	662,179
b19_iccad	255,278	255,300	776,320
cordic_ispd	45,359	45,393	127,993
des_perf_ispd	138,878	139,112	371,587
edit_dist_ispd	147,650	150,212	416,609
fft_ispd	38,158	39,184	116,139
leon2_iccad	1,616,369	1,616,984	4,178,874
leon3mp_iccad	1,247,725	1,247,979	3,267,993
netcard_iccad	1,496,719	1,498,555	3,901,343
matrix_mult_ispd	164,040	167,242	475,186
mgc_edit_dist_iccad	161,692	164,254	444,693
mgc_matrix_mult_iccad	171,282	174,484	489,670
pci_bridge32_ispd	40,790	40,950	108,172
usb_phy_ispd	923	938	2,447

new payload, and slack = capacity – size. After a capacity-fragmentation gate, PACT compacts when

$$\text{slack} \cdot s(\rho_t) \geq \max(B_{\text{move}} - B_{\text{new}}, 0), \quad (9)$$

with $s(\rho_t) = 10$ for $\rho_t \geq 10\%$, 14 for $1\% \leq \rho_t < 10\%$, and 20 otherwise.

- (2) **Scatter-patch metadata sync** (line 9). PublishMetadata uses scatter-patch kernels for sparse metadata updates, and falls back to bulk copy only when the full metadata copy is at most 32 MiB and no more than 8× the scatter-upload size.
- (3) **Append-oriented growth without slab reuse** (line 7). PlaceRow defaults to append, avoiding intrusive free-list bookkeeping whose memory benefit disappears under global-slack allocation.
- (4) **Gated CPU prefetching** (lines 5–7). The rematerialization loop enables prefetching at distance 16 only for batches of at least 65,536 entries; smaller batches see no benefit.

These thresholds define the selected realization evaluated as PACT; Base PACT disables them in the Q1 ablation. Both variants preserve the committed-traversal contract of Algorithm 1.

4 Experimental Results

We implemented PACT with GCC 13.3.0, CUDA 13.0, and Rust 1.92.0 on Ubuntu 22.04.5 with two AMD EPYC 7542 32-core CPUs, an RTX 4090, and 251 GiB RAM. We ask whether PACT (Q1) reduces topology-maintenance cost, (Q2) reduces timing-ECO turnaround, and (Q3) remains useful across host parallelism and ECO scale. Except for thread scaling, runs use 16 CPU threads and one GPU.

4.1 Dynamic Graph Update Microbenchmark

Q1 isolates topology maintenance in a common harness. Each configuration runs three iterations of 100 update rounds, with append/replace/remove edit weights of 4/4/2; Table 2 reports the mean runtime. We reimplement the Dual-CSR representation baseline from its paper in the same harness [22], while Base PACT disables the four offline-selected policies.

Across 13 configurations, PACT reaches 829.68 K updates/s versus 230.89 for the Dual-CSR representation baseline and 408.30 for

Table 2: Pure graph update runtime and throughput under controlled parameter sweeps.

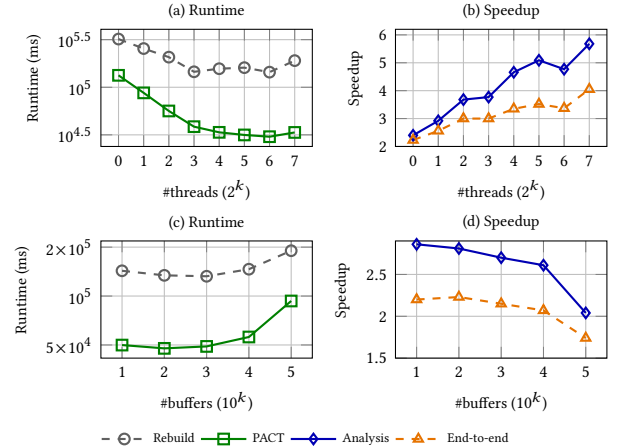
Default configuration: Size=1M, batch=1,024, average degree=8, super-node fraction=2%, and super-node degree=256.

Sweep	Factor	Rebuild RT/Throu.	Dual-CSR RT/Throu.	Base RT/Throu.	PACT RT/Throu.
Default	1×	58.80/17.42	3.96/258.34	2.40/425.80	1.34/763.03
Size+batch	2×	111.61/18.35	7.07/289.82	4.69/436.89	2.35/870.73
	5×	265.04/19.32	19.55/261.86	11.83/432.74	5.27/972.20
	10×	543.00/18.86	48.47/211.29	24.59/416.36	11.15/918.03
Avg. deg.	2×	87.49/11.70	4.13/248.22	2.38/430.90	1.20/854.35
	5×	165.32/6.19	5.02/204.09	2.59/395.99	1.30/787.59
	10×	290.96/3.52	6.50/157.48	2.92/350.13	1.61/635.86
Super frac.	2×	78.33/13.07	4.00/255.77	2.37/431.46	1.09/941.23
	5×	130.49/7.85	4.29/238.82	2.51/408.05	1.23/833.78
	10×	218.85/4.68	5.12/200.03	2.74/374.09	1.38/739.80
Super deg.	2×	80.05/12.79	3.93/260.78	2.35/435.43	1.06/962.29
	5×	134.61/7.61	4.25/240.84	2.52/406.05	1.23/834.24
	10×	218.18/4.69	4.90/209.01	2.72/375.84	1.37/746.51
Geometric mean		150.83/9.68	6.32/230.89	3.57/408.30	1.76/829.68
GM speedup		85.75	3.59	2.03	1.00

RT: mean runtime per update round in ms, averaged over three iterations of 100 rounds each; Throu.: throughput in K updates/s.

Size+batch scales node count and batch size together.

Base disables PACT’s four offline-selected policies. GM speedup is the geometric mean of PACT throughput divided by method throughput.


Figure 4: Thread and buffer-insertion scalability on netcard_iccad through 10^5 buffers per round.

Base PACT, yielding 3.59× and 2.03×. The former compares the implementations in the common harness; the latter measures the combined gain from the four policies.

Dual-CSR’s 23.87× geometric-mean gain over rebuild here covers graph updates only; IncreGPUSTA’s reported 2.87× on netcard_iccad includes levelization and timing propagation in a buffer-insertion flow [22]. These ratios measure different workloads and timing boundaries. Base PACT alone provides 1.77× the Dual-CSR throughput.

Table 3: End-to-end gate cloning in HeteroSTA; PACT and rebuild use shared ECO sequences and produce identical complete timing outputs.

R10-P100 Workload									
Benchmark	Initial WNS	Final WNS	Initial TNS	Final TNS	ECO RT	Rebuild Analysis RT	PACT Analysis RT	E2E Speedup	Analysis Speedup
aes_core	-22.02	-9.65	-7.31e3	-3.79e3	147.81	2,636.02	915.65	2.62×	2.88×
vga_lcd_iccad	-77.25	-34.07	-2.29e5	-1.87e5	2,683.49	22,773.59	5,668.90	3.05×	4.02×
b19_iccad	-37.66	-22.45	-9.64e4	-7.11e4	2,312.17	19,832.44	6,076.86	2.64×	3.26×
cordic_ispd	-290.27	-221.74	-2.22e5	-1.92e5	250.19	4,345.39	1,896.87	2.14×	2.29×
des_perf_ispd	-88.38	-73.23	-1.19e5	-1.01e5	3,248.95	9,858.50	3,965.91	1.82×	2.49×
edit_dist_ispd	-179.41	-173.48	-4.00e5	-3.81e5	2,242.88	12,336.77	4,865.11	2.05×	2.54×
fft_ispd	-49.06	-44.26	-1.70e4	-8.83e3	317.21	4,125.55	1,468.35	2.49×	2.81×
leon2_iccad	-62.81	-32.75	-4.25e6	-3.44e6	22,646.25	197,597.32	41,790.35	3.42×	4.73×
leon3mp_iccad	-70.55	-39.85	-2.78e6	-2.37e6	15,947.84	140,477.04	30,784.43	3.35×	4.56×
netcard_iccad	-94.13	-46.94	-3.05e6	-2.15e6	17,515.07	166,904.68	32,690.32	3.67×	5.11×
matrix_mult_ispd	-165.34	-137.38	-2.60e5	-2.32e5	1,944.83	13,872.35	4,804.45	2.34×	2.89×
mgc_edit_dist_iccad	-29.51	-27.94	-7.51e4	-6.89e4	2,583.11	13,355.80	4,564.59	2.23×	2.93×
mgc_matrix_mult_iccad	-22.33	-21.92	-2.77e4	-2.68e4	2,239.93	13,752.22	4,405.29	2.41×	3.12×
pci_bridge32_ispd	-210.41	-89.75	-2.24e5	-1.49e5	355.40	3,909.11	1,439.77	2.38×	2.72×
usb_phy_ispd	-37.51	-33.77	-1.45e3	-1.20e3	15.25	1,309.11	194.60	6.31×	6.73×
Geometric Mean	—	—	—	—	1,425.82	14,007.44	4,159.42	2.72×	3.37×
R100-P10 Workload									
Benchmark	Initial WNS	Final WNS	Initial TNS	Final TNS	ECO RT	Rebuild Analysis RT	PACT Analysis RT	E2E Speedup	Analysis Speedup
aes_core	-22.02	-8.60	-7.31e3	-3.74e3	1,013.60	25,204.87	7,910.52	2.94×	3.19×
vga_lcd_iccad	-77.25	-12.04	-2.29e5	-9.91e4	23,718.71	210,747.95	51,935.49	3.10×	4.06×
b19_iccad	-37.66	-18.80	-9.64e4	-6.98e4	18,865.81	195,239.25	54,938.49	2.90×	3.55×
cordic_ispd	-290.27	-169.08	-2.22e5	-1.57e5	2,321.86	42,942.43	17,501.70	2.28×	2.45×
des_perf_ispd	-88.38	-61.78	-1.19e5	-9.84e4	8,528.67	86,727.00	32,834.25	2.30×	2.64×
edit_dist_ispd	-179.41	-148.31	-4.00e5	-3.51e5	9,485.99	107,303.88	38,067.32	2.46×	2.82×
fft_ispd	-49.06	-35.74	-1.70e4	-9.25e3	2,072.73	38,358.71	13,071.67	2.67×	2.93×
leon2_iccad	-62.81	-25.46	-4.25e6	-2.80e6	185,635.51	1,694,384.33	343,587.20	3.55×	4.93×
leon3mp_iccad	-70.55	-26.26	-2.78e6	-2.01e6	116,795.26	1,162,952.81	261,606.20	3.38×	4.45×
netcard_iccad	-94.13	-29.44	-3.05e6	-1.58e6	122,952.93	1,361,280.45	272,840.33	3.75×	4.99×
matrix_mult_ispd	-165.34	-123.56	-2.60e5	-2.29e5	12,423.93	129,220.08	42,189.08	2.59×	3.06×
mgc_edit_dist_iccad	-29.51	-24.38	-7.51e4	-6.76e4	9,442.32	111,617.66	36,725.73	2.62×	3.04×
mgc_matrix_mult_iccad	-22.33	-20.38	-2.77e4	-2.69e4	10,688.25	117,988.30	37,155.53	2.69×	3.18×
pci_bridge32_ispd	-210.41	-74.30	-2.24e5	-1.40e5	2,467.76	35,927.91	12,463.43	2.57×	2.88×
usb_phy_ispd	-37.51	-46.65	-1.45e3	-1.76e3	57.82	5,761.14	898.82	6.08×	6.41×
Geometric Mean	—	—	—	—	8,479.32	119,797.18	34,192.35	2.96×	3.50×

WNS/TNS: setup worst/total negative slack in ns; RT: runtime in ms. Runtimes are cumulative over each ECO sequence; E2E includes ECO runtime, while analysis covers parasitic extraction, three state commits, and timing analysis.

4.2 End-to-End Gate-Cloning Case Study

We integrate PACT into HeteroSTA [10] and evaluate it on the TAU circuits [12] (Table 1). Gate cloning duplicates drivers and repartitions sinks, exercising all three state layers. R10-P100 executes 10 rounds of 100 critical-path selections, while R100-P10 executes 100 rounds of 10 selections. Rebuild and PACT use identical ECO sequences; the oracle compares complete timing outputs, with Table 3 reporting their common setup WNS/TNS.

Across 15 circuits, R10-P100 reaches 3.37×/2.72× analysis/E2E speedup, while R100-P10 reaches 3.50×/2.96×. Identical complete timing outputs support contract-preserving composition across all

three domains. The larger R100-P10 gain is consistent with greater state reuse under more frequent, smaller commits.

4.3 Scalability

Figure 4 characterizes host parallelism and ECO scale on the evaluated platform. Thread scaling uses a separate run of the R10-P100 gate-cloning schedule. At one thread, gate-cloning analysis/E2E speedups are 2.40×/2.23×; at 128 threads, they reach 5.68×/4.05×. Beyond 16 threads, the gains are limited and non-monotonic. Across the ten-round buffer insertion sweep, speedup declines from 2.86×/2.20× at 10 buffers per round to 2.04×/1.74× at 10⁵. The largest case inserts 10⁶ buffers cumulatively, comparable to

netcard_iccad's 1.50 million original gates. Thus the measured operating range extends beyond small edits, while work approaches rebuild scale as affected CSR rows approach the full graph.

5 Conclusion

PACT separates runtime-private mutation representation from a representation-independent committed-traversal contract and composes that contract across three dependent timing-ECO state layers. Its selected realization achieves $3.59\times$ the topology-maintenance throughput of a reimplemented Dual-CSR baseline, while gate-cloning integration delivers $2.72\times$ and $2.96\times$ speedups with identical timing results. The measured operating range extends beyond small edits, while the advantage diminishes as affected CSR rows approach the full graph. Future work can investigate how this methodology may extend to other dynamic GPU-accelerated EDA systems with sparse-topology updates and row-range traversal consumers.

Acknowledgments

This work is supported in part by Beijing High Innovation Plan (202604841450) and Jiangsu National Science and Technology Major Project (BG2025014).

References

- [1] Charles J. Alpert, Shirang K. Karandikar, Zhuo Li, Gi-Joon Nam, Stephen T. Quay, Haoxing Ren, Cliff N. Sze, Paul G. Villarrubia, and Mehmet C. Yildiz. 2007. Techniques for fast physical synthesis. *Proc. IEEE* 95, 3 (2007), 573–599.
- [2] Charles J. Alpert, Dinesh P. Mehta, and Sachin S. Sapatnekar (Eds.). 2008. *Handbook of algorithms for physical design automation*. CRC Press, Boca Raton, FL, USA.
- [3] Jayaram Bhasker and Rakesh Chadha. 2009. *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, New York, NY, USA.
- [4] Jeff Bonwick. 1994. The slab allocator: An object-caching kernel memory allocator. In *USENIX Summer 1994 Technical Conference*. USENIX Association, Boston, MA, USA, 87–98.
- [5] Federico Busato, Oded Green, Nicola Bombieri, and David A. Bader. 2018. Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs. In *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Piscataway, NJ, USA, 1–7.
- [6] Bangqi Fu, Lixin Liu, Martin D. F. Wong, and Evangeline F. Y. Young. 2024. Hybrid Modeling and Weighting for Timing-Driven Placement with Efficient Calibration. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Association for Computing Machinery, New York, NY, USA, 1–9.
- [7] Hongru Gao, Xiaofei Liao, Zhiyuan Shao, Kexin Li, Jiajie Chen, and Hai Jin. 2024. A Survey on Dynamic Graph Processing on GPUs: Concepts, Terminologies and Systems. *Frontiers of Computer Science* 18, 4 (2024), 184106.
- [8] Oded Green and David A. Bader. 2016. cuSTINGER: Supporting Dynamic Graph Algorithms for GPUs. In *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Piscataway, NJ, USA, 1–6.
- [9] Zizheng Guo, Tsung-Wei Huang, Zhou Jin, Cheng Zhuo, Yibo Lin, Runsheng Wang, and Ru Huang. 2024. Heterogeneous Static Timing Analysis with Advanced Delay Calculator. In *Design, Automation and Test in Europe (DATE)*. IEEE, Piscataway, NJ, USA, 1–6.
- [10] Zizheng Guo, Haichuan Liu, Xizhe Shi, Shenglu Hua, Zuodong Zhang, Chunyuan Zhao, Runsheng Wang, and Yibo Lin. 2026. HeteroSTA: A CPU-GPU Heterogeneous Static Timing Analysis Engine with Holistic Industrial Design Support. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, Piscataway, NJ, USA, 1090–1096.
- [11] Zizheng Guo, Zuodong Zhang, Wuxi Li, Tsung-Wei Huang, Xizhe Shi, Yufan Du, Yibo Lin, Runsheng Wang, and Ru Huang. 2024. HeteroExcept: A CPU-GPU Heterogeneous Algorithm to Accelerate Exception-Aware Static Timing Analysis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Association for Computing Machinery, New York, NY, USA, 1–9.
- [12] Jin Hu, Greg Schaeffer, and Vibhor Garg. 2015. TAU 2015 contest on incremental timing analysis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Piscataway, NJ, USA, 882–889.
- [13] Tsung-Wei Huang, Guannan Guo, Chun-Xun Lin, and Martin D. F. Wong. 2021. OpenTimer v2: A New Parallel Incremental Timing Analysis Engine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 40, 4 (2021), 776–789.
- [14] Tsung-Wei Huang and Martin D. F. Wong. 2015. OpenTimer: A high-performance timing analysis tool. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Piscataway, NJ, USA, 895–902.
- [15] IEEE International Roadmap for Devices and Systems. 2024. *2024 Update: More Moore*. Technical Report. IEEE.
- [16] Pei-Yu Lee, Iris Hui-Ru Jiang, Cheng-Ruei Li, Wei-Lun Chiu, and Yu-Ming Yang. 2015. iTimerC 2.0: Fast incremental timing and CPPR analysis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Piscataway, NJ, USA, 890–894.
- [17] Wan Luan Lee, Shui Jiang, Dian-Lun Lin, Che Chang, Boyang Zhang, Yi-Hua Chung, Ulf Schlichtmann, Tsung-Yi Ho, and Tsung-Wei Huang. 2025. iG-kway: Incremental k-way Graph Partitioning on GPU. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, Piscataway, NJ, USA, 1–7.
- [18] Wan Luan Lee, Aditya Das Sarma, Che Chang, Chih-Chun Chang, and Tsung-Wei Huang. 2026. iHyperG: Incremental Hypergraph Partitioning on GPU. In *ACM/IEEE Design Automation Conference (DAC)*. Association for Computing Machinery, New York, NY, USA, 1–7.
- [19] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. 2023. DREAMPlace 4.0: Timing-Driven Placement with Momentum-Based Net Weighting and Lagrangian-Based Refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 10 (2023), 3374–3387.
- [20] John Lillis, Chung-Kuan Cheng, and T.-T. Y. Lin. 1996. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Journal of Solid-State Circuits* 31, 3 (1996), 437–447.
- [21] Yinnian Lin, Lei Zou, and Xunbin Su. 2024. Towards Sufficient GPU-accelerated Dynamic Graph Management: Survey and Experiment. *Proceedings of the VLDB Endowment (PVLDB)* 18, 3 (2024), 599–612.
- [22] Haichuan Liu, Zizheng Guo, Runsheng Wang, and Yibo Lin. 2025. IncreGPUSTA: GPU-accelerated incremental static timing analysis for iterative design flows. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Piscataway, NJ, USA, 1–9.
- [23] Yi-Chen Lu, Zizheng Guo, Kishor Kunal, Rongjian Liang, and Haoxing Ren. 2025. INSTA: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, Piscataway, NJ, USA, 1–7.
- [24] C. Peddada, A. Goel, B. Dheeraj, and N. Chandrachoodan. 2015. iitRACE: A memory efficient engine for fast incremental timing analysis and clock pessimism removal. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Piscataway, NJ, USA, 903–909.
- [25] Mo Sha, Yuchen Li, Bingsheng He, and Kian-Lee Tan. 2017. Accelerating Dynamic Graph Analytics on GPUs. *Proceedings of the VLDB Endowment (PVLDB)* 11, 1 (2017), 107–120.
- [26] Peter Suaris, Lungtien Liu, Yuzheng Ding, and Nanchi Chou. 2004. Incremental physical resynthesis for timing optimization. In *ACM Symposium on FPGAs*. Association for Computing Machinery, New York, NY, USA, 99–108.
- [27] Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. 1995. Dynamic storage allocation: A survey and critical review. In *Memory Management (Lecture Notes in Computer Science, Vol. 986)*. Springer, Berlin, Heidelberg, 1–116.
- [28] Martin Winter, Daniel Mlakar, Rhaleb Zayer, Hans-Peter Seidel, and Markus Steinberger. 2018. faimGraph: High Performance Management of Fully-Dynamic Graphs under Tight Memory Constraints on the GPU. In *ACM/IEEE Supercomputing Conference (SC)*. IEEE, Piscataway, NJ, USA, 754–766.
- [29] Martin Winter, Rhaleb Zayer, and Markus Steinberger. 2017. Autonomous, Independent Management of Dynamic Graphs on GPUs. In *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Piscataway, NJ, USA, 1–7.
- [30] Chunyuan Zhao, Zizheng Guo, Rui Wang, Zaiwen Wen, Yun Liang, and Yibo Lin. 2024. HeLEM-GR: Heterogeneous Global Routing with Linearized Exponential Multiplier Method. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Association for Computing Machinery, New York, NY, USA, 1–9.
- [31] Chunyuan Zhao, Zizheng Guo, Zuodong Zhang, and Yibo Lin. 2026. GAP-LA: GPU-Accelerated Performance-Driven Layer Assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 45, 9 (2026), 4100–4109.
- [32] Lei Zou, Fan Zhang, Yinnian Lin, and Yanpeng Yu. 2023. An Efficient Data Structure for Dynamic Graph on GPUs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 11 (2023), 11051–11066.