Structural Timing-Aware Circuit Partitioning with Feasibility Constraints for Multi-Chiplet Design

Hengyuan Zhang*
Beijing University of Posts and Telecommunications
Beijing, China
zhy679@bupt.edu.cn

Kanglin Tian*
Beijing University of Posts and Telecommunications
Beijing, China
tiankl@bupt.edu.cn

Zirui Li
Beijing University of Posts and Telecommunications
Beijing, China
lzr_official@bupt.edu.cn

Jianwang Zhai†
Beijing University of Posts and Telecommunications
Beijing, China
zhaijw@bupt.edu.cn

Xiuli Fu
Beijing University of Posts and Telecommunications
Beijing, China
xiulifu@bupt.edu.cn

Kang Zhao
Beijing University of Posts and Telecommunications
Beijing, China
zhaokang@bupt.edu.cn

Abstract

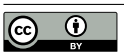
Chiplet-based heterogeneous integration has become a scalable paradigm for modern VLSI systems, where early-stage chiplet partitioning critically affects timing quality after physical design. However, conventional hypergraph partitioning primarily optimizes connectivity, while accurate timing analysis is unavailable or too expensive to obtain at early stages. To address this issue, we present a timing-aware chiplet partitioning framework that integrates structural timing modeling into hypergraph construction while preserving logic module binding and physical feasibility constraints. By embedding timing sensitivity into hyperedge weights, the proposed method guides partitioning to better preserve timing-critical connections during early design stages. A constraint-preserving hypergraph construction strategy is further introduced to maintain implementation consistency across the physical design flow. Experimental results show that the proposed framework significantly improves post-placement timing metrics, and the benefit becomes more pronounced as the chiplet count increases.

Keywords

Chiplet Partitioning, Timing-Aware, Feasibility Constraints

ACM Reference Format:
Hengyuan Zhang, Kanglin Tian, Zirui Li, Jianwang Zhai, Xiuli Fu, and Kang Zhao. 2026. Structural Timing-Aware Circuit Partitioning with Feasibility Constraints for Multi-Chiplet Design . In *Great Lakes Symposium on VLSI 2026 (GLSVLSI '26), June 22–24, 2026, Canandaigua, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3787109.3815270

*Co-first authors with equal contributions.
†Corresponding author.
This work is supported by the National Natural Science Foundation of China (No. 62404021), the Fundamental Research Funds for the BUPT (No. 2025AI4S20), the Research Initiation Project for Introduced Talents of BUPT (No. 510124058, 510224062), and the BUPT Excellent Ph.D. Students Foundation (No. CX20251004).



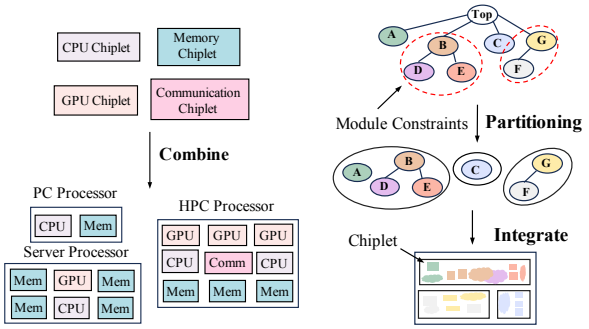
GLSVLSI '26, Canandaigua, NY, USA

ACM ISBN 979-8-4007-2431-2/2026/06
https://doi.org/10.1145/3787109.3815270

1 Introduction

Chiplet-based heterogeneous integration has become an important paradigm for scaling modern VLSI systems beyond the limits of monolithic integration [1, 2]. As illustrated in Fig. 1, two major design paradigms are commonly adopted in multi-chiplet systems. In the *composition-based* paradigm, chiplets from a pre-designed library are assembled to form a complete system. In contrast, the *decomposition-based* paradigm starts from a monolithic design and partitions it into multiple chiplets that are fabricated separately and later integrated through advanced packaging technologies. By decomposing a large monolithic design into multiple smaller chiplets interconnected through advanced packaging technologies, chiplet-based systems enable improved yield, design reuse, and heterogeneous integration across different process nodes. However, this architectural shift also introduces new system-level physical design challenges for EDA tools. In particular, top-level chiplet partitioning determines how the original design is decomposed and which connections remain intra-die versus becoming inter-die links. Since inter-die interconnect typically incurs higher latency and stricter electrical constraints than intra-die nets, early partitioning decisions can significantly affect the achievable timing quality after physical implementation [3]. Consequently, chiplet partitioning becomes a critical early-stage decision that can strongly influence downstream physical design and timing closure.

Recent EDA research has increasingly pushed timing awareness toward earlier stages of the physical design flow, since iterative routing and signoff timing analysis are often too expensive to be embedded in upstream optimization loops. For example, Net2 [4] indicates that timing-relevant signals can be inferred before placement from structural design information, enabling early identification of timing-sensitive nets and paths when detailed physical data is not yet available. Guo *et al.* [5] further show that pre-routing slack can be predicted using timing-engine-inspired graph models, suggesting that endpoint-level timing risk can be approximated without running a full back-end toolchain. These studies suggest that useful timing cues can be extracted from structural features available at the netlist level. However, early timing estimation alone does not improve implementation quality unless it is incorporated into the optimization process. In large-scale netlist decomposition, hypergraph



(a) Composition-based (b) Decomposition-based

Figure 1: Two chiplet design paradigms: (a) composition-based integration from chiplet libraries. (b) decomposition-based partitioning for monolithic designs.

partitioning provides a natural vehicle for this integration: classical multilevel frameworks establish the widely adopted coarsen–initial partition–refine paradigm, and modern open-source solvers enable scalable K-way optimization [6, 7]. Beyond connectivity-only objectives, recent efforts have explored learning-guided hypergraph partitioning to inject task-aligned signals into partition optimization rather than treating all nets uniformly [8]. This trend motivates building on scalable open-source partitioners while introducing timing-aware information through hyperedge reweighting, so that partitioning decisions become more consistent with post-placement timing behavior under chiplet integration.

Timing-aware chiplet partitioning is particularly challenging, mainly due to two practical issues. First, design feasibility must be guaranteed under hard module binding constraints and physical region constraints (e.g., minimum area, utilization, and aspect ratio), where any violation leads to invalid chiplet design solutions; systematically incorporating such constraints into scalable partitioning remains non-trivial [9]. Second, accurate timing guidance is expensive early in the flow, motivating lightweight timing characterization that can be extracted from netlist-accessible features and used to steer upstream optimization without invoking full signoff analysis [4, 5, 10, 11].

To address these challenges, we develop a *structural timing-aware hypergraph partitioning* framework that integrates lightweight timing characterization with hierarchy-aware feasibility preservation. The key idea is to embed timing sensitivity into the partitioning objective through hyperedge weighting derived from structural information available at the netlist level, while enforcing module binding constraints through a constraint-preserving hypergraph transformation guided by the module hierarchy.

The main contributions are summarized as follows:

- **Hierarchy-aware constraint-preserving hypergraph construction.** A tree-based hierarchy representation is used to interpret module binding specifications with hierarchical semantics and to perform constraint-preserving node contraction, ensuring that all module constraints remain satisfied throughout partitioning.
- **Lightweight structural timing modeling for partition guidance.** Timing sensitivity is characterized using netlist-accessible structural factors and embedded into bounded integer hyperedge weights, enabling timing-aware partitioning without relying on full static timing analysis within the partitioning loop.
- **End-to-end placement-based validation.** Partitioning results are propagated to physical implementation and validated by post-placement timing metrics, quantifying timing improvement under realistic physical constraints.

2 Preliminaries

Chiplet-based design emphasizes modular implementation, which naturally requires a circuit to be partitioned into physically separable and functionally meaningful components. Therefore, chiplet partitioning must not only divide the netlist structurally but also respect module semantics, physical feasibility, and timing implications. This section introduces the basic hypergraph representation, feasibility constraints, and the timing-driven problem formulation used in this work.

2.1 Hypergraph Representation

Given a gate-level netlist, the circuit is modeled as a vertex- and hyperedge-weighted hypergraph [6]:

$$H = (V, E), \tag{1}$$

where V is the set of vertices and E is the set of hyperedges. Each vertex $v \in V$ represents a physical instance (e.g., a cell or a macro), and each hyperedge $e \in E$ represents a net connecting a subset of vertices.

Each vertex $v \in V$ is associated with a non-negative area weight $a(v) \in \mathbb{R}_{\geq 0}$, where $a(v)$ denotes the physical area of v. Each hyperedge $e \in E$ is associated with a positive integer weight $w(e) \in \mathbb{Z}^+$, where $w(e)$ denotes the importance of net e in partitioning.

For a target number of chiplets $K \in \mathbb{Z}^+$, a K-way partition is defined by:

$$\pi : V \to \{1, 2, \ldots, K\}, \tag{2}$$

where $\pi(v) = i$ means that vertex v is assigned to chiplet i. This assignment induces K disjoint subsets:

$$V_i = \{v \in V \mid \pi(v) = i\}, \qquad i = 1, 2, \ldots, K. \tag{3}$$

For each hyperedge $e \in E$, its cut indicator is defined as [6]:

$$\delta(e) = \begin{cases} 1, & \exists\, u, v \in e \text{ such that } \pi(u) \neq \pi(v), \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

where $\delta(e) = 1$ indicates that net e is cut across multiple chiplets. Based on this definition, the weighted cut cost of the partition π is:

$$C_{\text{cut}}(\pi) = \sum_{e \in E} w(e)\delta(e), \tag{5}$$

where $C_{\text{cut}}(\pi)$ measures the total weighted cut introduced by the partition. In this work, $C_{\text{cut}}(\pi)$ is used to characterize structural partition quality, but it is not the final optimization objective.

2.2 Feasibility Constraints for Chiplet Partitioning

Practical chiplet partitioning must satisfy both module binding constraints and physical feasibility constraints [2].

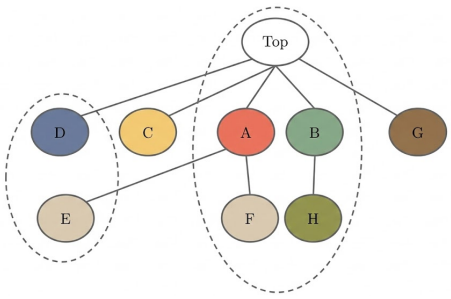


Figure 2: Example of module grouping constraints over a hierarchical tree.

Module Constraints with Hierarchical Semantics. Let $\mathcal{P}$ denote the set of module paths in the hierarchical design tree. The constraint specification is represented as a set of groups:

$$\mathcal{G} = \{G_1, G_2, \ldots, G_M\}, \tag{6}$$

where each group $G_m \subseteq \mathcal{P}$ contains module paths that must be assigned to the same chiplet. For a module path $p \in \mathcal{P}$, let $\text{chip}(p) \in \{1, 2, \ldots, K\}$ denote its chiplet assignment. Then, for every $G_m \in \mathcal{G}$,

$$\forall\, p, q \in G_m, \quad \text{chip}(p) = \text{chip}(q). \tag{7}$$

As illustrated in Figure 2, dashed regions denote module groups that must be co-located on the same chiplet, while tree edges denote hierarchical containment. Each constrained module path represents its subtree; if one of its descendants is explicitly listed in another group, that descendant subtree is excluded from the ancestor scope. This exclusion rule ensures an unambiguous interpretation of overlapping parent/child constraints.

Physical Feasibility Constraints. Let $R_i \in \mathbb{R}_{>0}$ denote the physical region area of chiplet i. The total area assigned to chiplet i is:

$$A_i = \sum_{v \in V_i} a(v), \tag{8}$$

where $A_i \in \mathbb{R}_{\geq 0}$ is the accumulated area of all vertices in chiplet i. Each chiplet must satisfy:

$$A_i \geq A_{\min}, \tag{9}$$

where $A_{\min} \in \mathbb{R}_{>0}$ is the minimum chiplet area, and utilization constraint:

$$u_i^{\min} \leq \frac{A_i}{R_i} \leq u_i^{\max}, \tag{10}$$

where $u_i^{\min}, u_i^{\max} \in (0, 1]$ are the lower and upper utilization bounds, respectively. In addition, chiplet regions must satisfy geometric constraints such as aspect ratio limits and non-overlap requirements [9].

2.3 Problem Formulation

The timing-aware chiplet partitioning problem, considered in terms of design feasibility, is defined as follows.

Input. The input includes a hypergraph $H = (V, E)$ derived from a gate-level netlist, vertex area weights $\{a(v) \mid v \in V\}$, hyperedge weights $\{w(e) \mid e \in E\}$, a target chiplet number K, module-based constraint groups $\mathcal{G}$, and physical feasibility parameters including $A_{\min}$, region areas $\{R_i\}_{i=1}^K$, utilization bounds $\{(u_i^{\min}, u_i^{\max})\}_{i=1}^K$, and geometric constraints.

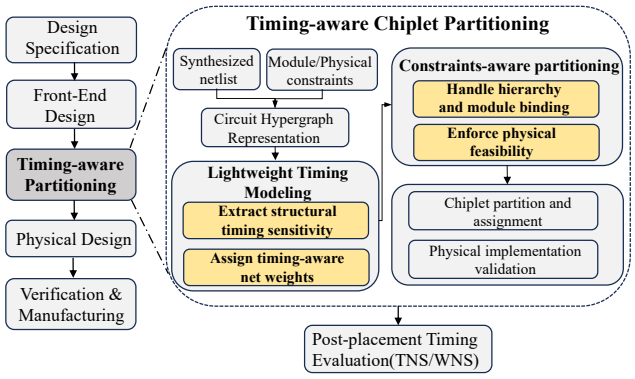


Figure 3: Overview of the timing-aware chiplet partitioning framework.

Optimization Goal. The goal is to find a feasible partition assignment $\pi : V \to \{1, 2, \ldots, K\}$ that improves the final timing quality after physical implementation, where the timing cost is evaluated by post-placement metrics such as Total Negative Slack (TNS) and Worst Negative Slack (WNS). Since accurate timing evaluation is difficult to obtain during early-stage partitioning, we use lightweight timing modeling to estimate relative timing sensitivity and encode it into hyperedge weights $w(e)$. Thus, the partitioning stage minimizes the weighted cut cost $C_{\text{cut}}(\pi)$ as a surrogate objective, while the final optimization target remains the post-placement timing quality.

Constraints. The partition must satisfy: (i) module-based co-location constraints under the hierarchical exclusion semantics; and (ii) physical feasibility constraints, including minimum area, utilization bounds, and geometric implementability.

In summary, the key challenge is to find a feasible partition that preserves timing-sensitive structures as much as possible while satisfying both hierarchical and physical constraints.

3 Methodology

As illustrated in Figure 3, the proposed timing-aware chiplet partitioning framework consists of two tightly coupled stages: *lightweight timing modeling* and *constraints-aware partitioning*. Starting from the synthesized netlist, the library file, and the module/physical constraint file, the circuit is first converted into a hypergraph representation together with the required physical and hierarchical attributes. On top of this representation, the lightweight timing modeling stage extracts structural timing sensitivity and assigns timing-aware net weights. The resulting weighted hypergraph is then processed by the constraints-aware partitioning stage, which handles hierarchy and module binding, enforces physical feasibility, and produces the chiplet partition and assignment. Finally, the partition result is validated by physical implementation, and post-placement timing is evaluated using TNS and WNS.

3.1 Lightweight Structural Timing Modeling

The purpose of lightweight timing modeling is to provide early-stage timing guidance for partitioning when accurate post-placement timing information is not yet available. Since invoking full timing

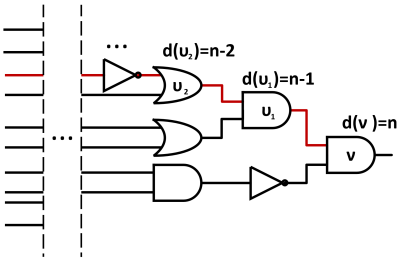


Figure 4: Illustration of logic depth $d(\cdot)$ defined as the longest fan-in path length ending at each cell.

analysis inside the partitioning loop is computationally impractical, the proposed method estimates timing sensitivity from structural information available from the synthesized netlist and the library. Three factors are considered for each hyperedge: logic depth, load scale, and inter-module span. Their combination yields a timing-aware score, which is subsequently mapped to the hyperedge weight used in partitioning.

3.1.1 Logic Depth. Logic depth characterizes delay accumulation along combinational propagation paths. A net embedded in a deeper logic context is therefore more likely to be associated with timing-critical structures and should receive a larger cut penalty.

This factor is derived from circuit topology extracted from the synthesized netlist using Yosys [12]. After parsing the netlist, Yosys recovers the fanin relations between cells, from which a directed dependency graph is constructed. Let $\mathrm{pred}(v)$ denote the set of fan-in predecessors of cell v. The logic depth of cell v is defined as the length of the longest fan-in path ending at v:

$$d(v) = \max_{u \in \mathrm{pred}(v)} \big(d(u) + 1\big), \tag{11}$$

where source-like cells are initialized to 0. Figure 4 illustrates this definition.

Since the cells incident to a hyperedge may lie at different depths, a net-level depth descriptor is introduced by averaging the depths of all incident cells:

$$D_e = \frac{1}{|e|} \sum_{v \in e} d(v), \tag{12}$$

where $|e|$ denotes the number of vertices incident to hyperedge e. In practice, D_e is obtained by first evaluating $d(v)$ for all cells through a topology-consistent traversal of the dependency graph and then aggregating the depths over each hyperedge.

3.1.2 Load Scale. The load scale reflects the receiver-side loading of a net. A connection that drives a larger total sink load is more likely to incur delay growth and is thus assigned a higher timing penalty.

The rationale follows the classical Elmore delay model [13]. For an RC tree rooted at the driver, the delay to a sink can be computed as:

$$t_{\mathrm{Elmore}} = \sum_{i \in \mathcal{E}} R_i \, C_i, \tag{13}$$

where R_i denotes the resistance of segment i and C_i denotes the downstream capacitance seen through that segment. This expression indicates that larger downstream capacitance generally leads to larger delay.

At the partitioning stage, however, detailed pin capacitance is usually unavailable. In standard-cell designs, sink input capacitance is primarily determined by transistor sizing and is typically correlated with the physical size of the sink instance. This motivates the monotone approximation:

$$C_{\mathrm{in}}(v) \;\propto\; \mathrm{size}(v) \;\propto\; a(v), \tag{14}$$

where $a(v)$ denotes the area of instance v.

Based on this approximation, the sink set $\mathrm{sinks}(e)$ of each hyperedge e is first identified from the driver–receiver relations in the parsed netlist, and the load-scale factor is defined as:

$$L_e = \sum_{v \in \mathrm{sinks}(e)} a(v), \tag{15}$$

where the area weight $a(v)$ is obtained by matching the instance type in the synthesized netlist to the corresponding cell entry in the library. Thus, L_e measures the total sink area driven by hyperedge e and serves as a lightweight proxy for its receiver-side capacitive load.

3.1.3 Inter-Module Span. Inter-module span measures how broadly a net spreads across the hierarchy. Nets touching more hierarchical modules are structurally more prone to cross-chiplet boundaries after partitioning and therefore deserve stronger protection.

The required hierarchy information is recovered from the parsed synthesized netlist. Let $\mathrm{mod}(v)$ denote the hierarchical module to which cell v belongs. The inter-module-span factor of hyperedge e is defined as:

$$S_e = \big|\{\mathrm{mod}(v) \mid v \in e\}\big|, \tag{16}$$

which counts the number of distinct modules covered by the cells on the hyperedge e. Computationally, this amounts to collecting the module labels of all incident cells and counting the unique memberships.

3.1.4 Weighted Combination and Net Reweighting. After the three factors are computed, they are normalized to comparable ranges by a monotone normalization operator $\mathcal{N}(\cdot)$. The timing sensitivity score of hyperedge e is then defined as:

$$s(e) = \alpha\, \mathcal{N}(D_e) + \beta\, \mathcal{N}(L_e) + \gamma\, \mathcal{N}(S_e), \tag{17}$$

where α, β, and γ are the coefficients satisfying $\alpha + \beta + \gamma = 1$.

The coefficients are not chosen manually. Instead, several candidate coefficient settings are predefined and evaluated through repeated experiments. The underlying consideration is that the three structural factors do not contribute equally to timing sensitivity; if the coefficient setting is inappropriate, their relative importance cannot be properly reflected, and the resulting hyperedge weights may become unstable across different designs or partitioning runs. We therefore compare multiple candidate settings and select the one that exhibits the most stable timing improvement behavior, i.e., the setting whose post-placement TNS/WNS improvements remain the most consistent over repeated experiments. Once selected, the same coefficient setting is fixed and used for all remaining benchmarks.

The calibrated score $s(e)$ is finally mapped to the integer range supported by the partitioner, producing the timing-aware hyperedge weight. In this way, nets with higher structural timing sensitivity are assigned larger cut penalties before partitioning.

3.2 Constraints-Aware Partitioning

After timing-aware hyperedge weights are assigned, the hypergraph is transformed to incorporate module binding constraints and physical feasibility requirements. This stage converts the original weighted hypergraph into a constrained weighted partitioning instance that can be directly handled by a standard partitioner.

3.2.1 Handling Hierarchy and Module Binding. Module constraints ensure that designated modules are co-located on the same chiplet under the hierarchical semantics defined in Section 2.2. Since these constraints are specified at the module-path level, they must first be translated into leaf-level partitioning entities.

The parsed netlist hierarchy is represented as a rooted tree $\mathcal{T}$ whose nodes correspond to hierarchical paths. Let $\text{Leaves}(p)$ denote the set of leaf instances contained in the subtree rooted at path p:

$$\text{Leaves}(p) = \{\, v \in V \mid v \text{ is a leaf instance and } v \text{ is under } p \,\}. \quad (18)$$

Given a constraint group G_k defined in Section 2.2, the corresponding instance-level binding set is:

$$B_k = \bigcup_{p \in G_k} \text{Leaves}(p), \quad (19)$$

which requires all instances in B_k to share the same chiplet assignment. In practice, this step is realized by traversing the hierarchy tree, expanding each constrained module path to its leaf instances, and forming the union over all module paths in the same constraint group. When both an ancestor module and one of its descendants appear in the constraint specification, the descendant subtree is excluded from the ancestor scope according to the same hierarchical exclusion semantics.

To embed these module-binding constraints directly into the partitioning instance, all vertices belonging to the same binding set are merged through a contraction mapping:

$$\phi : V \to V_c, \quad (20)$$

where V_c is the vertex set of the contracted hypergraph. The area of each contracted vertex $u \in V_c$ is defined as:

$$a_c(u) = \sum_{v \in V : \phi(v) = u} a(v), \quad (21)$$

which aggregates the areas of all original vertices mapped to u. As a result, the module-binding constraints are locked into the graph structure, while the physical size of each merged entity remains visible to the partitioner.

3.2.2 Reconstruction of the Contracted Hypergraph. Once vertex contraction is completed, the hypergraph itself must be rebuilt. Each hyperedge is updated according to its contracted endpoints:

$$\phi(e) = \{\phi(v) \mid v \in e\}. \quad (22)$$

The reconstruction then follows naturally from the updated endpoint sets. A hyperedge whose endpoints all collapse to the same contracted vertex becomes an internal connection and is discarded, since it no longer influences partitioning. If multiple contracted vertices remain incident to the hyperedge, its connectivity is preserved on the contracted graph. In addition, different original hyperedges may collapse onto the same contracted endpoint set after vertex merging. Such hyperedges are merged into a single hyperedge, and their weights are accumulated so that the total timing penalty carried by the original connections is preserved after contraction. The resulting contracted hypergraph is denoted by:

$$H_c = (V_c, E_c). \quad (23)$$

3.2.3 Partitioning and Physical Feasibility Enforcement. The constrained weighted hypergraph H_c is then partitioned using a multilevel hypergraph partitioning algorithm, which can be implemented by open-source partitioners such as KaHyPar[7]. The partitioning process takes the contracted vertex areas $a_c(\cdot)$ as vertex weights and the timing-aware hyperedge weights as cut costs, and produces a partition with K chiplets.

After each partitioning run, the assigned area of each chiplet is collected from the partition result and checked against the physical feasibility conditions defined in Section 2.2, including minimum chiplet area, utilization bounds, and region-related requirements. Operationally, this amounts to summing the contracted vertex areas assigned to each partition and comparing the resulting partition areas with the feasible physical ranges implied by the constraint file. If any chiplet violates these physical constraints, the partitioning procedure is re-invoked with updated balance settings, and the same feasibility check is repeated until a feasible partition is obtained.

3.2.4 Physical Implementation Validation. Once a feasible partition is found, the chiplet assignment on contracted vertices is expanded back to the original instances through ϕ^{-1} and propagated to the physical design representation. Placement is then performed under chiplet constraints using DREAMPlace [14, 15]. Finally, post-placement timing quality is evaluated using TNS and WNS. This validation step verifies whether the proposed timing-aware partitioning decisions translate into post-placement timing improvement under realistic physical implementation.

4 Evaluation

4.1 Experimental Setup

The proposed timing-aware partitioning flow is implemented in C++ and Python on top of open-source hypergraph partitioners[7, 16, 17]. For downstream physical validation, placement is performed using the open-source DREAMPlace [14, 15, 18], and post-placement timing analysis is conducted using OpenSTA [19]. All experiments are conducted on a Linux workstation, equipped with a 2.40 GHz Intel(R) Xeon(R) CPU, NVIDIA RTX3090 GPU, and 128 GB of memory. Each experiment is independently repeated five times, and the reported results correspond to the average.

4.1.1 Benchmarks. We conduct experiments on four benchmark circuits. Table 1 presents statistics for each circuit, adapted from [20]. Each case provides a gate-level netlist along with: (i) module constraints specifying binding requirements and hierarchical semantics; and (ii) physical constraints specifying the core area and chiplet feasibility requirements, including minimum chiplet area, utilization bounds, and aspect ratio bounds. Feasibility is treated

Table 1: Benchmark statistics.

Case	# Std Cells	# Macros	# Nets	# Chiplets
case1	84K	14	101K	3
case2	1221K	96	1385K	3
case3	136K	26	158K	3
case4	1221K	96	1385K	3

*case2 and case4 share identical sizes but different connectivity.

as a hard requirement: any solution that violates the constraints is considered invalid. These benchmarks cover different scales and connectivity structures, enabling the evaluation of whether the proposed framework remains effective across different designs. In particular, **case2** and **case4** have the same design statistics but different netlist connectivity, which helps evaluate the effect of connectivity differences under similar design size.

4.1.2 Validation Flow and Metrics. Because the goal of partitioning is to improve post-placement timing quality rather than only partition-level structural objectives, all partitioning solutions are validated through a unified physical design flow. For each partitioning solution, the partition results are first translated into chiplet-level floorplanning constraints. Chiplet-level floorplanning is performed using FTAFP [21]. To eliminate floorplanning-induced variability, relaxed area constraints are adopted so that all partitioning solutions within each case share the same floorplan configuration. This ensures that timing improvements primarily arise from partitioning decisions rather than from floorplanning artifacts. The floorplan result is then translated into fence-region constraints, under which mixed-size placement, macro legalization, and timing-driven global placement are performed. The final post-placement timing quality is evaluated using **(i)** TNS and **(ii)** WNS.

4.1.3 Chiplet Count. For fair comparison, all methods use the same chiplet count K within each benchmark case. In addition to the default setting of each case, we further vary K to study how the benefit of timing-aware partitioning changes with chiplet granularity. As K increases, more inter-chiplet nets are introduced, making timing-sensitive net cuts more likely. This experiment is therefore used to examine whether the proposed method becomes more beneficial under finer-grained chiplet partitioning.

4.2 Baselines

We use three open-source hypergraph partitioners as baselines: KaHyPar[7], hMetis[16], and PaToH[17]. To isolate the contribution of the proposed timing-aware weighting scheme, a paired comparison strategy is adopted for each partitioner.

(i) Baseline configuration: the partitioner runs in its original mode, where partitioning is driven only by vertex weights and structural connectivity, without introducing any hyperedge weights for timing awareness.

(ii) Our configuration: the same partitioner runs under identical constraint settings—including target chiplet number K, core area coordinates, and minimum chiplet area—but incorporates the timing-aware hyperedge weights derived in Section 3.1

This setup forms a controlled ablation study in which the only independent variable is whether timing-aware hyperedge reweighting is enabled. For fair evaluation, all partitions are propagated through a standardized downstream physical validation flow. This flow enforces identical module and physical constraints (e.g., utilization and aspect ratio ranges) and employs placement. Consequently, any differences in TNS and WNS are attributable to the partitioning objective rather than implementation settings.

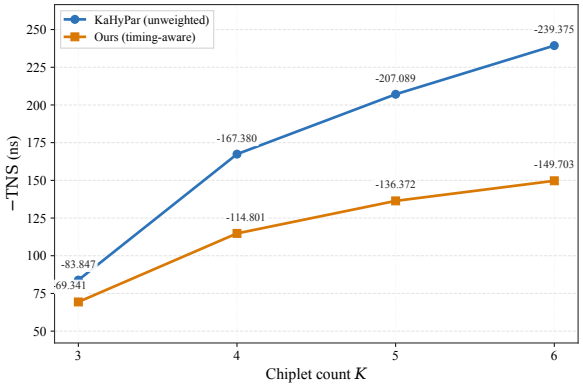


Figure 5: Sensitivity to chiplet count K on case1 (KaHyPar). The figure plots $-TNS$ (ns) for the baseline (unweighted) and the proposed timing-aware reweighting.

4.3 Main Results

This experiment is conducted to verify whether the proposed timing-aware hyperedge weighting improves post-placement timing quality under the same partitioning backend and downstream implementation flow. Table 2 reports post-placement timing results (TNS/WNS) on four benchmark cases. For each engine, the baseline and the proposed configuration differ only in whether timing-aware hyperedge weights are enabled.

Across all cases and solver engines, the proposed weighting consistently improves TNS. The largest improvement is observed on case3 with hMETIS, achieving a 42.36% TNS reduction, while the smallest improvement occurs on case4 with hMETIS, with a 2.85% reduction. These results indicate that although the improvement magnitude depends on circuit characteristics and solver behavior, the overall TNS trend remains consistently positive.

In contrast, WNS improvements are generally smaller, and a few slight regressions in some cases are observed. For example, in case1 with KaHyPar, WNS changes from −2.980 to −2.986. These regressions are minor compared with the corresponding TNS improvements. This is expected because the proposed weighting mainly targets accumulated timing violations reflected by TNS, whereas WNS depends on a single worst path and is more sensitive to local post-placement effects.

The bottom row of Table 2 provides an average-level summary of the overall improvement trend. Since TNS improvements are more stable across cases and better reflect global timing quality, the subsequent chiplet-count study uses TNS as the primary metric.

4.4 Impact of Chiplet Count

To further examine the effect of chiplet granularity, we conduct an additional experiment on case1 using KaHyPar with $K \in \{3, 4, 5, 6\}$.

Table 2: Post-placement timing comparison (TNS/WNS). For each engine, "Proposed (Ours)" applies the same partitioner with timing-aware hyperedge weights. The bottom row reports the *ratio* computed on averages over four cases: ratio = $\overline{m}_{\text{baseline}}/\overline{m}_{\text{ours}}$ for $m \in \{\text{TNS}, \text{WNS}\}$; ratios of "Proposed (Ours)" are set to 1.000.

Case	KaHyPar [7]		Proposed (Ours)		hMETIS [16]		Proposed (Ours)		PaToH [17]		Proposed (Ours)	
	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)	TNS (ns)	WNS (ns)
case1	-83.847	**-2.980**	**-69.341**	-2.986	-145.862	-6.307	**-106.900**	**-4.985**	-107.211	-4.195	**-91.937**	**-2.927**
case2	-310251.380	-14.391	**-261220.510**	**-10.589**	-187346.630	-7.240	**-159209.555**	**-5.312**	-193499.223	-8.128	**-176979.627**	**-6.259**
case3	-1062.246	**-4.975**	**-944.488**	-7.561	-1419.855	**-3.554**	**-818.417**	-3.668	-1177.700	-3.083	**-903.668**	**-2.651**
case4	-210643.020	-11.410	**-199073.670**	**-9.250**	-161082.328	-6.139	**-156495.215**	**-5.932**	-198919.273	-6.960	**-144477.516**	**-5.208**
ratio	1.132	1.111	**1.000**	**1.000**	1.105	1.168	**1.000**	**1.000**	1.221	1.312	**1.000**	**1.000**

The two compared configurations differ only in whether timing-aware hyperedge weights are enabled. Figure 5 plots $-TNS$ as a function of K.

As K increases, more chiplet boundaries are introduced, increasing the chance that timing-sensitive connections are cut across chiplets. In this setting, timing-aware weighting becomes increasingly beneficial because it explicitly discourages cuts on structurally critical nets. The TNS ratio (baseline divided by proposed) increases from 1.209 at $K = 3$ to 1.599 at $K = 6$, showing that the advantage of the proposed method becomes more pronounced under finer-grained chiplet partitioning. This trend is consistent with the expectation that additional chiplet boundaries create more opportunities for timing-sensitive cuts, thereby making timing-aware guidance more valuable.

4.5 Discussion

The above results suggest that the proposed timing-aware weighting is more effective at improving global timing quality, as reflected by TNS, than optimizing the single worst path, as reflected by WNS. Since TNS accumulates violations over many endpoints, it is more directly affected by whether timing-sensitive connections are preserved within the same chiplet, whereas WNS is dominated by one critical path and is more sensitive to local placement variation.

The chiplet count experiment in Section 4.4 further supports this interpretation. As K increases, more chiplet boundaries are introduced, making timing-sensitive connections more likely to be cut under connectivity-only objectives. Thus, the proposed weighting provides stronger guidance for finer-grained chiplet partitioning by assigning larger penalties to structurally critical nets.

The improvement magnitude varies across partitioners and benchmark cases, since the final partition also depends on each solver's coarsening, refinement, and local search behavior, as well as circuit topology. Nevertheless, the consistently positive TNS trend across all engines indicates that the proposed timing-aware weighting is robust to different partitioners.

5 Conclusion

This paper presents a timing-aware chiplet partitioning framework that integrates structural timing sensitivity into hypergraph partitioning under hierarchical module constraints and strict physical feasibility requirements. The proposed method combines lightweight timing-aware hyperedge weighting with constraint-preserving hypergraph transformation, enabling timing-aware partitioning while maintaining validity.

Evaluation demonstrates that the proposed framework consistently improves post-placement timing quality across benchmark cases and partitioners, with improvements of 10.5%∼22.1% for TNS and 11.1%∼31.2% for WNS. The chiplet-count study further shows that the advantage of timing-aware partitioning becomes more significant as more chiplet boundaries are introduced. These results suggest that early-stage chiplet partitioning should explicitly consider timing sensitivity rather than relying only on connectivity-driven objectives. Incorporating timing awareness at the partitioning stage provides an effective path toward better downstream timing closure in chiplet-based physical design.

References

[1] S. Chen, H. Zhang *et al.*, "CHASE: A Chiplet Architecture Simulation and Exploration Framework with Decoupled Multi-Fidelity Optimization," in *Proc. ISPD*, 2026, pp. 153–161.

[2] S. Chen *et al.*, "The survey of 2.5 D integrated architecture: An EDA perspective," in *Proc. ASPDAC*, 2025, pp. 285–293.

[3] Y. Kim *et al.*, "Signal and power integrity analysis in 2.5 D integrated circuits (ICs) with glass, silicon and organic interposer," in *Proc. ECTC*, 2015, pp. 738–743.

[4] Z. Xie *et al.*, "Preplacement net length and timing estimation by customized graph neural network," *IEEE TCAD*, vol. 41, no. 11, pp. 4667–4680, 2022.

[5] Z. Guo *et al.*, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. DAC*, 2022, pp. 1207–1212.

[6] G. Karypis *et al.*, "Multilevel hypergraph partitioning: Application in vlsi domain," in *Proc. DAC*, 1997, pp. 526–529.

[7] S. Schlag *et al.*, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[8] M. H. Khan *et al.*, "VLSI hypergraph partitioning with deep learning," *arXiv preprint arXiv:2409.01387*, 2024.

[9] I. Bustany *et al.*, "An open-source constraints-driven general partitioning multi-tool for VLSI physical design," in *Proc. ICCAD*, 2023, pp. 1–9.

[10] K. Chang and T. Kim, "Pre-route timing prediction and optimization with graph neural network models," *Integration*, vol. 99, p. 102262, 2024.

[11] X. He *et al.*, "Efficient timing prediction and optimization using derivable gradient boosting machine model at placement stage," *ACM TODAES*, 2025.

[12] C. Wolf *et al.*, "Yosys-a free verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics*, 2013, pp. 1–6.

[13] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[14] J. Gu *et al.*, "DREAMPlace 3.0: Multi-electrostatics based robust VLSI placement with region constraints," in *Proc. ICCAD*, 2020, pp. 1–9.

[15] P. Liao *et al.*, "DREAMPlace 4.0: Timing-driven global placement with momentum-based net weighting," in *Proc. DATE*, 2022, pp. 939–944.

[16] G. Karypis *et al.*, "Multilevel k-way hypergraph partitioning," in *Proc. DAC*, 1999, pp. 343–348.

[17] Ü. V. Çatalyürek and C. Aykanat, "PaToH: Partitioning Tool for Hypergraphs." 2011.

[18] Y. Chen *et al.*, "Stronger Mixed-Size Placement Backbone Considering Second-Order Information," in *Proc. ICCAD*, 2023, pp. 1–9.

[19] "OpenROAD-flow-script," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts, 2023.

[20] "China Postgraduate IC Innovation Competition·EDA Elite Challenge Contest," http://edachallenge.cn, 2024.

[21] Z. Li *et al.*, "FTAFP: A Feedthrough-Aware Floorplanner for Hierarchical Design of Large-Scale SoCs," in *Proc. ASPDAC*, 2025, pp. 886–892.