Invited Paper: 2025 ICCAD CAD Contest Problem C: Incremental Placement Optimization Beyond Detailed Placement: Simultaneous Gate Sizing, Buffering, and Cell Relocation

Yi-Chen Lu, Rongjian Liang, Wen-Hao Liu, and Haoxing Ren
NVIDIA Research
{yilu, haoxingr}@nvidia.com

***Abstract*—Late-stage placement optimization is where real PPA trade-offs surface, and where conventional heuristic passes tend to get trapped in small, local neighborhoods. We frame an invited "Problem C" contest that treats this stage as a global, multi-operator search over gate sizing, buffer/inverter-pair insertion, and legal cell relocation, with strict reproducibility and legality. Our core belief grounded in production experience is that GPU batching and differentiable guidance expand the tractable search space: you can score and steer thousands of coordinated moves per iteration, not just a handful, and do so under tight runtime budgets. Submissions must produce a replayable ECO changelist and a final legal DEF; a standardized evaluation flow computes timing, power, and wirelength and combines them with displacement and runtime into the contest score. The specification is designed to encourage pragmatic use of gradient signals and tensorized batching without mandating any single method, enabling participants to leverage novel GPU tools to deliver industrially deployable PPA gains.**

I. Introduction

Incremental optimization after global placement is where hard Power, Performance, and Area (PPA) trade-offs become realistic. At this stage, every transform including resizing a gate, inserting an inverter pair, relocating cells through timing, wirelength, congestion, and power optimization needs to be performed in a coherent manner. Existing industrial Electronic Design Automation (EDA) flows manage this complexity with carefully engineered heuristics and repeated STA loops. These flows are robust, but they are also inherently myopic that they only explore a narrow neighborhood around the current solution under the traditional detailed placement scheme, because broad, global moves are computationally expensive and risky late in the flow. The result is familiar to practitioners—significant engineering effort spent negotiating small, local improvements while leaving better global solutions on the table.

Our thesis is simple that with modern GPU-acceleration tools [1], [2] and differentiable techniques [3], [4], we can search through a much larger portion of the feasible space much faster, with clearer guidance about which directions are promising. With modern million-gate industrial designs, a single A100-class and above GPU can evaluate or approximate tens of thousands of candidate moves in parallel, amortizing the cost of model inference, sensitivity analysis, and legality checks. Furthermore, when paired with Machine Learning (ML) driven or differentiable surrogates (e.g., differentiable timing estimators), this approach shifts the traditional exploration regime from "try-and-hope" to "measure-and-steer". Note that we believe PPA gradients need not be perfect to be useful, since coarse yet well-calibrated sensitivities already prune vast swaths of unproductive actions and expose non-obvious combined moves (size+buffer+relocate) that conventional greedy passes rarely uncover.

From an industrial view, three realities motivate the problem we pose. First, late-stage placement objectives are multi-criteria and coupled: improving TNS/WNS must preserve legality, respect Multi-Cornere-Multi-Mode (MCMM) constraints, and avoid power regressions, where the right solution is almost always a set of coherent moves. Second, the move space is discrete but structured where sizing options are function-compatible, buffer insertion comes with topological and considering physical constraints (e.g., density), and cell displacements must interact with local routing resources and timing corners. Third, runtime budgets are non-negotiable: an approach that wins on quality but loses an engineer-day in runtime will not ship. The contest we define encodes these practicalities directly in the inputs/outputs, scoring, and runtime rules.

Why GPUs and differentiability now? Recently, two key enablers have matured in the ralm of Physical Design (PD). (i) Batched evaluation on GPUs. With proper problem packaging including vectorized feature extraction, candidate generation, and batched legality/timing proxy evaluation, we can score orders of magnitude more "what-if analysis" moves per unit wall time than CPU-first scripts. This changes is not just about the speed, but a fundamental paradigm shift in optimization strategy: once we can examine thousands of joint actions per iteration, global coordination beats local greed. (ii) Differentiable analysis. Although exact signoff GPU-STA remains an open-problem needs to be solved, but differentiable timing proxies at early design flows (e.g., placeopt) and placement-aware surrogates provide directional signals that are fast with good correlation to end-of-flow PPA. These signals support

gradient-informed search (e.g., prioritize moves aligned with negative timing gradients), warm-start discrete solvers, and guide LLM priors that propose physically plausible edits rather than free-form guesses.

This Problem C consequently invites the community to treat post-global-placement optimization as a global, multi-operator search under strict legality, with two design choices that reflect production constraints. First, we permit three families of transforms—gate sizing, buffer/inverter-pair insertion, and legal cell relocation—because, in practice, the best fixes are composites. Second, we require both a replayable ECO changelist and a final legal DEF, because reproducibility and auditability matter as much as headline PPA numbers in industrial signoff. Evaluation scripts verify functional consistency, replay ECOs to confirm determinism, and compute timing/power/wirelength using a standardized tool flow; runtime is explicitly scored to reflect total cost-of-optimization.

We emphasize that "differentiable" here is pragmatic rather than doctrinaire. Participants may use exact tools such as OpenROAD [5], learned surrogates, or hybrids; gradients can be analytic, automatic, or approximate. What matters is using signal to steer exploration—for example, ranking candidate buffer sites by estimated path-sensitivity, or jointly sizing a fanout cone based on aggregated slack gradients. Likewise, "GPU-accelerated" covers a spectrum: from fully learned models trained on synthetic/real ECO data, to classical heuristics refactored to exploit tensorized batching. In all cases, the payoff is the same: broader search, better coordination, tighter runtime.

II. Problem Formulation

Scope and goal. We study incremental physical optimization after detailed placement using three operator families: *gate sizing*, *buffer or inverter pair insertion*, and *legal cell relocation*. The objective is to raise timing quality and reduce power while keeping wirelength and displacement under control, with strict legality and full reproducibility.

Inputs. Each testcase provides a self contained package with gate level Verilog, Liberty timing and power files, LEF technology and cell abstracts, a legal seed DEF, and organizer scripts and constraints for evaluation. All fixed objects such as IOs, macros, keep outs, and blockages are immutable and must be honored.

Allowed transformations. *Gate sizing* replaces an instance with a function compatible library variant that is legal in the technology. *Buffer or inverter pair insertion* augments a net with new instances and nets while preserving functional equivalence. *Legal relocation* moves standard cells on the placement site grid with correct row orientation and without overlap. Logic restructuring beyond buffering or inverter pairs, use of cells that do not exist in the library, edits that change combinational equivalence, or changes to clock tree topology are not allowed unless explicitly enabled by the organizers.

ECO command set. Candidates must express edits with the following commands.

- `size_cell <cellName> <libCellName>`
- `insert_buffer {<load pins>} <buffer_lib_cell> <new_buf_inst> <new_net>`
- `insert_buffer -inverter_pairs {<load pins>} <inv_lib_cell> {<inv_instances>} {<new_nets>}`

Names for new instances and nets must be unique and deterministic and must appear in the submitted DEF. Commands are applied in order from top to bottom. Invalid commands may be skipped by the evaluator.

File and format clarifications. All placement edits must snap to valid sites and respect row orientation and power rail alignment as defined by LEF and DEF. Fixed IOs and macros cannot move, and all keep outs and blockages must be respected. If Bookshelf views are provided, they are for convenience only and must remain consistent with the DEF and with the libraries. The ECO changelist defines the netlist that results from the proposed edits and must preserve functional equivalence except for allowed buffering and inverter pairs. Replaying the ECO on the seed must reconstruct the submitted DEF exactly or within the stated coordinate tolerance.

Outputs. Two artifacts are required for every testcase:

- *Final legal DEF* `<design>.sol.def` that reflects all edits and remains legal and consistent.
- *Replayable ECO changelist* `<design>.sol.changelist` that reproduces the submitted DEF when applied to the seed with the organizer tools.

Determinism and reproducibility. Submissions must be deterministic in the reference environment given the organizer seed. Any use of randomness must be controlled so that repeated runs produce the same ECO and the same DEF. The ECO to DEF replay is the canonical reproducibility check.

Evaluation metrics and scoring. The official flow computes the following metrics in a consistent and reproducible manner:

- *Timing* with multi path measures such as TNS and WNS on organizer specified corners and constraints.
- *Power* from Liberty with organizer switching assumptions.
- *Wirelength* using a routed or estimator consistent measure defined by the organizers.
- *Displacement* as average and selected percentile Manhattan distance from the seed for moved instances.
- *Runtime* as the wall clock time of the timed solve phase on the reference machine.

We compute a composite score S from PPA improvement P, displacement penalty D, and runtime efficiency R:

$$P = \alpha\,\mathrm{TNS}_{\mathrm{imp}} + \beta\,\mathrm{Power}_{\mathrm{red}} + \gamma\,\mathrm{WL}_{\mathrm{red}},$$
$$S = 1000\,P - 50\,D - 300\,R.$$

Here $\mathrm{TNS}_{\mathrm{imp}}$, $\mathrm{Power}_{\mathrm{red}}$, and $\mathrm{WL}_{\mathrm{red}}$ are normalized improvements over the seed. D is the average Manhattan displacement per cell in site units, normalized over the seed. R is runtime normalized to a per testcase reference, with time limits applied. Weights α, β, and γ are testcase dependent. If the ECO replay does not match the submitted DEF the run is invalid and no score is reported.

III. Evaluation Environment and Submission

Reference environment. All runs execute inside a supplied Docker image on Ubuntu 20.04 with eight virtual CPUs, thirty two gigabytes of memory, one NVIDIA A100 with forty gigabytes of memory, CUDA 11.8 runtime, driver series 525, and a Mambaforge Python 3.9 environment as defined by `lagrange_env.yaml`. Use of GPU based batching and differentiable guidance is encouraged and not required; measured runtime contributes to the score.

Submission package. Submit a single archive `solution.tar.gz` that unpacks into a top level directory named `solution`. At minimum it must include:

- `setup_environment.sh` which installs any dependencies.
- `run.sh` which performs the solve phase when invoked as `./run.sh <design> <WL_w` α`> <power_w` β`> <timing_w` γ`>`.
- Your code and any binaries or models needed to run.
- The organizer provided directory `testcases/<design_name>/`.

Required outputs. When `run.sh` completes on a testcase it must emit

- `<design_name>.sol.def`
- `<design_name>.sol.changelist`

The evaluation flow first replays the ECO and verifies equality from ECO to DEF, then computes metrics and the contest score. If replay does not reconstruct the submitted DEF the run is invalid.

IV. Benchmark Suite

We adopt the ASAP7 [6] library with multiple threshold voltage (VT) flavors as the technology base, where each standard cell is available in four threshold flavors with representative drive strength ladders. Liberty and LEF define timing, power, and geometry, and each testcase ships with a legal seed DEF that is consistent with the libraries and ready for incremental optimization under our rules.

Design sources and construction. Designs are derived from IWLS [7] and TILOS [8] suites. We synthesize the RTL under varied constraint sets that exercise different PPA configurations and a range of target frequencies and cell densities. For each netlist we generate multiple placed instances by sweeping place and route configurations such as target density, clock constraints, congestion effort, legalization settings, and routing effort. The result is a controlled set of testcases that vary in path depth, fanout profile, and congestion character. The four VT choices introduce realistic timing and leakage trade offs so that methods must coordinate sizing, buffering, and relocation rather than rely on a single class of edits.

Scale, splits, and reproducibility. Instance counts span from small control style blocks to larger data path oriented designs. The suite provides a public split for development and ablation and a hidden split for final scoring. Normalization for scoring is per design relative to its seed to keep results comparable across the diversity of circuits. Every testcase runs inside the reference container with deterministic seeds and organizer scripts, which ensures that experiments are repeatable and that scores reflect method quality rather than environment variance.

TABLE I: The open benchmarks and their attributes in *ASAP 7nm*.

Name	# Nets	# FFs	# Cells	# arcs
ac97_top	7,836	2,191	7,750	62,807
aes	4,660	670	4,393	53,653
aes_cicpher_top	11,891	530	56,194	331,320
ariane	10,883	19,894	105,730	1,328,984
des	2,450	190	2,317	15,015
pci_bridge	12,264	3,313	12,091	111,424

Finally, Table I shows the characteristics of the designs we used for open evaluation. More designs will be leveraged for hidden evaluation which will be announced after the contest.

References

[1] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[2] Y.-C. Lu, Z. Guo, K. Kunal, R. Liang, and H. Ren, "Insta: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications," in *2025 62th ACM/IEEE Design Automation Conference (DAC)*, 2025.

[3] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1315–1320, 2022.

[4] Y.-C. Lu, K. Kunal, G. Pradipta, R. Liang, R. Gandikota, and H. Ren, "Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes," in *Proceedings of the 2025 International Symposium on Physical Design*, pp. 152–162, 2025.

[5] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–4, 2019.

[6] V. Vashishtha, M. Vangala, and L. T. Clark, "Asap7 predictive design kit development and cell design technology co-optimization," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 992–998, IEEE, 2017.

[7] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS)*, vol. 9, 2005.

[8] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, "Assessment of reinforcement learning for macro placement," in *Proceedings of the 2023 International Symposium on Physical Design*.