HeteroSTA: A CPU-GPU Heterogeneous Static Timing Analysis Engine with Holistic Industrial Design Support

Zizheng Guo[1,2], Haichuan Liu[1], Xizhe Shi[1], Shenglu Hua[1], Zuodong Zhang[2], Chunyuan Zhao[1], Runsheng Wang[1,2,3], Yibo Lin[1,2,3,*]
[1]School of Integrated Circuits, *Peking University*, [2]Institute of EDA, *Peking University*
[3]Beijing Advanced Innovation Center for Integrated Circuits, *Corresponding author: yibolin@pku.edu.cn

***Abstract*—We introduce in this paper, HeteroSTA, the first CPU-GPU heterogeneous timing analysis engine that efficiently supports: (1) a set of delay calculation models providing versatile accuracy-speed choices without relying on an external golden tool, (2) robust support for industry formats, including especially the .sdc constraints containing all common timing exceptions, clock domains, and case analysis modes, and (3) end-to-end GPU-acceleration for both graph-based and path-based timing queries, all exposed as a zero-overhead flattened heterogeneous application programming interface (API). HeteroSTA is publicly available with both a standalone binary executable and an embeddable shared library targeting ubiquitous academic and industry applications. Example use cases as a standalone tool, a timing-driven DREAMPlace 4.0 integration, and a timing-driven global routing integration have all demonstrated remarkable runtime speed-up and comparable quality.**

I. Introduction

With the sheer increase in the VLSI design volume as well as the growing demand for fast turnaround time and rapid design iteration, heterogeneous CPU/GPU-accelerated EDA has garnered wide attention as an extremely important aspect of next-generation EDA systems. With years of development, GPU acceleration techniques have covered most of the major EDA stages, including RTL simulation [1]–[4], logic synthesis [5]–[10], partitioning [11]–[16], placement [17]–[27], routing [28]–[36], design rule checking [37], [38], timing signoff [39]–[49], etc. Among these efforts, static timing analysis (STA) is one of the central tasks because all major stages embed STA in their timing optimization inner loop, where STA is invoked thousands of times.

The advancement in heterogeneous CPU/GPU STA algorithms has brought orders-of-magnitude speedup to timing analysis, covering all major steps in a typical and complete STA engine (delay calculation [39]–[41], graph propagation [42]–[44], exception handling [45], and path search [46]–[49]). Since then, we have witnessed the early-stage development of timing-driven EDA flows that are end-to-end heterogeneous [19], [24]–[26]. However, existing research all focuses on the acceleration of specific STA steps without integrating all algorithms into a rich-featured, readily-available heterogeneous STA library. Prior timing-driven EDA flows were thus all forced to implement their own small proof-of-concept heterogeneous STA modules that have poor design compatibility (e.g., industrial Verilog, SDC, and SPEF), low analysis accuracy, and inferior runtime efficiency.

The gap between research and application in the GPU-

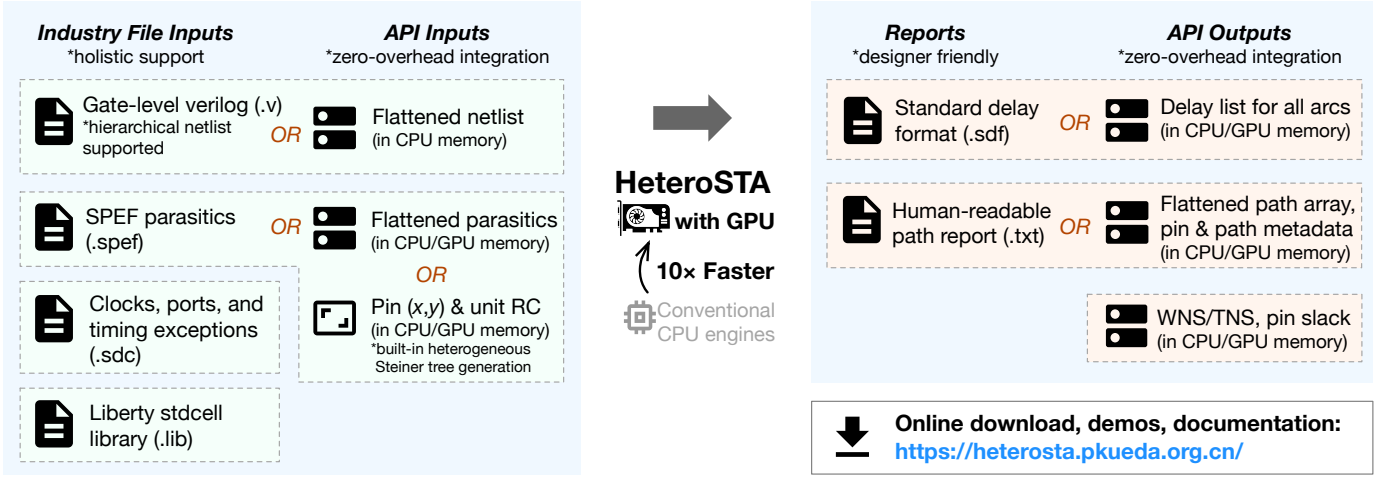

Fig. 1: HeteroSTA supports various input and output combinations featuring holistic industry file format support and a zero-cost abstraction interface. It delivers an order of magnitude STA speed-up for heterogeneous EDA applications.

accelerated STA field has motivated us to develop a holistic heterogeneous STA library, **HeteroSTA**, targeting widespread academic and industry adoption. With such a library available to the public, both researchers and engineers can bootstrap their research and development of EDA algorithms and flows that benefit greatly from the much faster encapsulated STA function, without having to deal with the enormous engineering details in implementing a full STA engine.

Compared to STA libraries available prior to this work, e.g., OpenTimer [50], OpenSTA [51], GCS-Timer [41], and INSTA [43], HeteroSTA is the first heterogeneous STA engine with holistic industrial design support. It is *self-contained*: it does not require an external golden tool to perform delay calculation or exception preprocessing. It is *industry-compatible*: it accepts industry-standard file formats like gate-level Verilog (with hierarchical support), SPEF parasitics, SDC constraints (covering clocks, ports, and all common timing exception definitions), and the Liberty cell library. It is *integration-ready*: as a library and a set of C/C++ API (Figure 1), it is ready to be embedded into different applications, with a novel heterogeneous API design ensuring zero-overhead in data communication between tools and the STA engine.

HeteroSTA is made available from today and can be downloaded at this link: [1], with online documentation and discussion available. Together with the release of the library, we also release two demos integrating HeteroSTA into the open-source timing-driven DREAMPlace 4.0 [52] and Efficient-TDP [53] placers, both available on GitHub[2]. We also integrated HeteroSTA into a GPU-accelerated global routing flow. After replacing the original timers in these representative flows with HeteroSTA, we achieve significant end-to-end speedup without quality degradation.

The rest of this paper is organized as follows. Section II discusses the challenges of a heterogeneous STA library with a review of current STA tools available. Section III demonstrates design details of HeteroSTA. Section IV presents the experimental results including the runtime performance and accuracy of HeteroSTA as both a standalone timer and an integrated STA engine in timing-driven placement and global routing workloads. Finally, Section V concludes the paper.

II. Challenges of a Heterogeneous STA Library

STA determines circuit performance by providing graph-based and path-based timing criticality reports given a circuit netlist, a cell library, a list of clocks and exceptions, and a parasitics annotation [54]. In a typical EDA flow, STA is itself an essential stage managing timing signoff post physical design. More importantly, STA is integrated as a subprocess in many other stages to guide circuit performance optimization – such optimizations are usually formulated as an "analyze-then-optimize" loop, where STA plays a central role in locating timing bottlenecks (Figure 2).

[1]https://heterosta.pkueda.org.cn/

[2]https://github.com/limbo018/DREAMPlace, and https://github.com/PKU-IDEA/Efficient-TDP-HeteroSTA

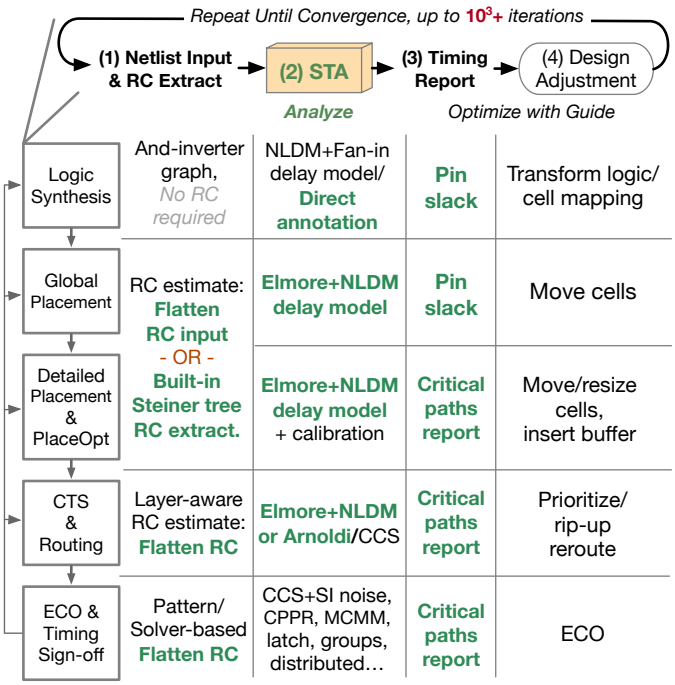

Fig. 2: STA is heavily used in design flows: a fast timer is critical in speeding up design iterations. Different stages may require customized STA inputs, outputs, and functions.

Due to their frequent reuse, core STA engines are packaged as libraries that can be embedded into other flows via an application programming interface (API). Such APIs are often kept for internal use in commercial EDA companies, and their STA engines (e.g., PrimeTime [55] and Tempus [56]) are only released as standalone tool executables with which interactions are mostly text-based (files or Tcl scripts). Instead, academic EDA flows often embed open-source STA libraries like OpenSTA [51] and OpenTimer [50] instead when commercial tools are unavailable and to avoid inefficiencies in text-based interfaces.

The timing reports mismatch between the golden commercial STA engines and the currently available, embeddable STA libraries is one of the major challenges that HeteroSTA tries to resolve. We identify two core issues that are accountable for the mismatch: *delay model accuracy* and *timing exception compatibility*. Accurately compute the cell and net delays at sub-micron nodes requires advanced circuit models beyond the current widely-used Elmore delay model. Timing exceptions such as false paths, multi-cycle paths, case analysis, and cross clock region paths (usually defined in `.sdc` files) can significantly complicate the data structures and states in timing propagation implementation and open-source STA engines often have limited or buggy support.

Currently, none of the publicly-available STA libraries have resolved both issues. The recent work INSTA [43] has presented remarkable correlation with commercial tools.

However, we note that it does *not* intend to address these issues by itself – instead, INSTA relies on a golden commercial STA engine to provide delay annotations and then performs timing propagation and slack calculation based on these annotations. These calculations are only part of the late steps of timing analysis that are closer to the generation of optimization guides. Furthermore, in our experiments, we will show incorrect results from INSTA when handling practical timing exceptions in timing propagation.

Heterogeneous CPU/GPU-accelerated EDA flows have demonstrated orders-of-magnitude runtime speedup and unprecedented scalability, enabling rapid design iteration, faster time-to-market, and more efficient design space exploration. Therefore, they have been regarded as promising directions for future EDA systems. A heterogeneous EDA flow requires a heterogeneous STA library to achieve its peak performance. STA is itself known as a runtime bottleneck in the optimization loop. Moreover, without a GPU-accelerated STA engine, the data interface between the GPU-accelerated optimization loop and the CPU-based STA engine would incur large back-and-forth CPU-GPU data movement cost, as we observe in such flows in reality [52], [53].

The demand on a STA library that is *natively heterogeneous* introduces new major challenges on top of existing ones. STA consists of many distinct graph-based algorithms that are hard to parallelize with GPU's single-instruction-multiple-threads (SIMT) model. Fortunately, these years have witnessed novel algorithms and data structures that help in bridging the compute architecture gap and making STA massive parallel, but how to wrap these individual contributions into a *comprehensive STA software* remains uncertain. In addition to the speed of STA itself, the overall efficiency of the heterogeneous-STA-powered EDA flow is heavily determined by the API design of the STA library. OpenTimer and OpenSTA expose object-oriented programming (OOP) APIs that work on individual cells, nets, or pins. Such OOP-based APIs are no longer feasible for heterogeneous data communications due to the high translation overhead of data structures [39]. Ideally, the communication between heterogeneous STA libraries and EDA flows should be based on a set of *zero-overhead heterogeneous APIs*, which is highly nontrivial without existing practices to follow.

III. HeteroSTA

As presented in Figure 1, HeteroSTA features both a versatile support for industrial file formats and a zero-cost data API designed for heterogeneous integration. To achieve this, HeteroSTA applies a modularized software design consisting of parsers, netlist database, graph-based STA, and path-based STA. Design constraints are parsed with a built-in Tcl interpreter and dispatched to different levels of abstraction. Figure 3 presents a detailed view of the architecture of HeteroSTA.

Industry file format parsers. We implement a set of high-performance parsers for industry-standard formats, including gate-level Verilog (`.v`), Liberty cell library (`.lib`), delay annotation (`.sdf`), and standard parasitics (`.spef`). Our

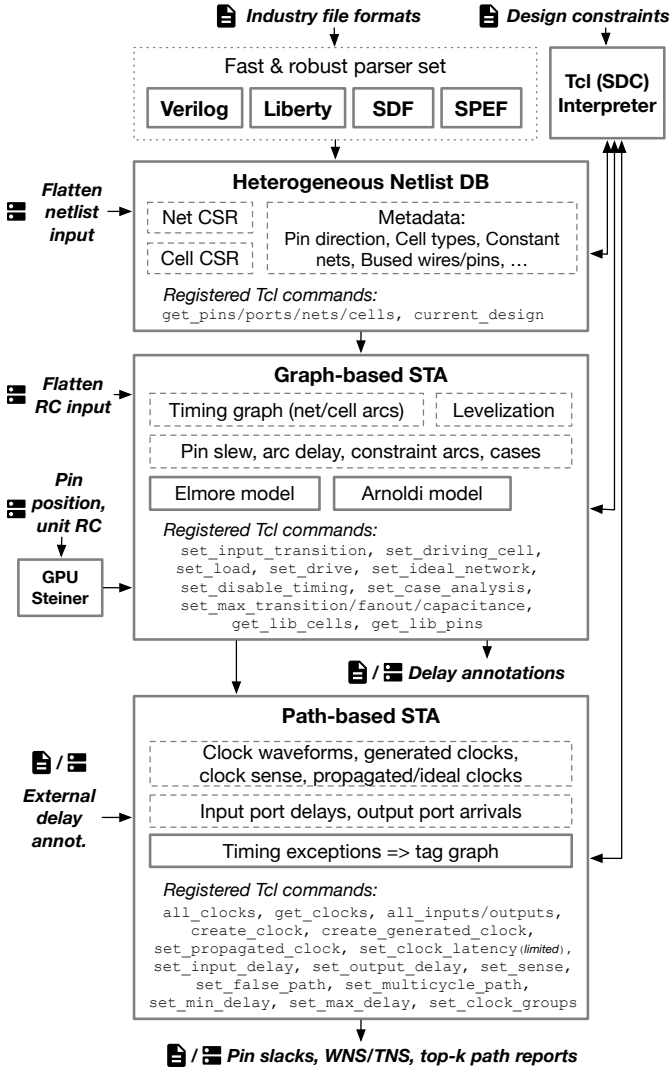

Fig. 3: The modular design of HeteroSTA showing inputs, outputs, and data interactions between submodules.

parsers are implemented with byte-level parsing expression grammar (PEG) that features a memory-efficient single-pass parsing strategy. We support streamed parsing and the parsing of gzipped inputs. Our gate-level Verilog parser supports hierarchical designs, buses, assignments, and constant wires (e.g., `1'b0`). Our Liberty parser is verified on several industrial 7–14 nm PDKs with slew derates, *when*-guarded timing arcs, pin functions, etc. All parsers support multi-threaded parsing: the Verilog, SDF, and SPEF parsers can run multi-threaded on a single file input by splitting the file heuristically into chunks, while the Liberty parser can read multiple `.lib` files from modern PDKs in parallel.

Tcl interpreter-based timing exceptions manager. We embed a standalone Tcl script interpreter in HeteroSTA to handle design constraints in `.sdc` files. This enables HeteroSTA to process advanced scripted SDCs, such as the ones with expressions, for-loops, condition branches, etc. SDC commands in the Tcl interpreter are registered by individual HeteroSTA

function modules across levels of abstraction (Figure 3).

Heterogeneous netlist database. We store the structure of the netlist as a central gate-level flattened netlist database. Rather than based on OOP styles, this database is natively flattened represented by compressed sparse rows (CSR) edge lists. We provide two ways to initialize this database. By inputting a Verilog, HeteroSTA builds the netlist database itself recursively through the hierarchy. Alternatively, HeteroSTA accepts external net and cell CSR arrays directly from a heterogeneous EDA flow outside, e.g., DREAMPlace. The latter way eliminates the need to rebuild the netlist or maintain pin indices mappings for STA invocated by other flows.

Parasitics (RC) inputs. HeteroSTA accepts 3 ways to input parasitics (RC) annotations. Annotations can be given through industry-standard `.spef` files with built-in parsers. For a heterogeneous optimization loop, parasitics are usually generated on-the-fly with an algorithm, so we also provide ways to directly send RC to HeteroSTA without file-based communications. One general way to do this is through a flattened RC interface, where a user packs the generated RC into a set of predefined CSR structures on either CPU or GPU – this is useful in timing-driven routing where the flow has control over the layer-based RC generation algorithms. For the placement context, we also embed a built-in Steiner tree-based RC extraction module based on GPU-accelerated FLUTE [29]. A placer only needs to provide HeteroSTA with pin positions and unit resistance/capacitance values along x/y directions and HeteroSTA will run built-in FLUTE to generate RC estimations and incorporate them in delay calculation.

Delay models. HeteroSTA currently supports two delay calculators: the simple classical Elmore delay calculator, and an Arnoldi-based reduced-order model. The Elmore model is the fastest yet not accurate enough in late design stages, where Arnoldi might be a better option. Our delay calculator framework is extensible and in the future we plan to add CCS delay models in the next release of HeteroSTA.

Report options. HeteroSTA supports a variety of timing reports to fit different design needs, including delay annotations, WNS/TNS, pin slacks, and top-k path reports. Delay annotations can be in the standard `.sdf` file output as well as in-memory delay arrays directly. Path reports can be controlled with top-k, per-endpoint report limit, and slack-less-than thresholds. The format of path report can be human-readable plain texts as well as a flattened CSR-based path pin arrays with slacks and other useful metadata. All output arrays can be on CPU or GPU at user's option, enabling fully heterogeneous optimization loops where the majority of design data never leaves GPU memory.

IV. Experimental Results

We release HeteroSTA as a Linux shared library (`.so`), a set of C header files (`.h`), example sources, and an online documentation.[3] Under the hood, HeteroSTA is implemented in Rust, C++, and CUDA. HeteroSTA supports both OpenMP-based CPU multithreading and CUDA-based GPU massive parallelism. Our experiments aim to evaluate HeteroSTA in two aspects: (1) its correlation with commercial tool PrimeTime, in both *delay calculation accuracy* (Section IV-A) and *timing exception correctness* (Section IV-B); (2) its end-to-end performance benefit when integrated into representative heterogeneous GPU-accelerated EDA flows, including two *timing-driven global placement* flows (Section IV-C) and one *timing-driven global routing* flow (Section IV-D). All our experiments are run on a Linux machine with 64 Intel Xeon Platinum 8358 CPU cores, 1 TB memory, and 8 NVIDIA A100-SXM4-80GB GPUs. Unless otherwise noted, we run all benchmarks 3 times and report average runtime, using 16 CPU cores and 1 GPU (if applicable), which is when their performance generally saturates.

A. Delay Calculation Accuracy

We test the delay modeling accuracy by comparing the delay annotations generated by HeteroSTA, PrimeTime, and OpenSTA on the TAU 2015 contest benchmarks [57] transpiled to a 14 nm process [40]. Both HeteroSTA and OpenSTA use their respective NLDM+Arnoldi delay model, while PrimeTime uses its basic (non-CCS) calculation mode. [4] Table I shows a detailed comparison of both runtime and accuracy. HeteroSTA achieves better correlation (R^2 and MAE) than OpenSTA compared to PrimeTime. Specifically, HeteroSTA achieves an average R^2 score of 0.985 and a MAE of 2.34 ps. Our accuracy is also stable and consistent across all designs we have tested.

Thanks to heterogeneous CPU/GPU acceleration, the runtime of HeteroSTA significantly outperformed both PrimeTime and OpenSTA among all benchmarks. We are on average 4.89× faster than PrimeTime and 10.04× faster than OpenSTA. On the largest design `leon2`, we are 7.56× and 14.53× faster, respectively.

B. Timing Exception Correctness

The timing propagation step involves handling of complex timing exceptions, which is another important source of mismatch. By instructing PrimeTime to write out its delay calculation results, we are able to isolate the delay calculation and timing propagation steps and test them separately – the latter of which is exactly the problem formulation of INSTA [43], our baseline. INSTA expects a set of `.csv` files containing circuit startpoint clock periods, endpoint required arrival times, arc delays, and a set of "timing-disabled" pins, which we all generate using PrimeTime Tcl scripts. The timing-disabled pin list captures some of the false path settings but not all of them, because false path exceptions may contain multiple `-through` patterns that eliminate only paths that go through a predefined pin sequence. Such patterns cannot be replaced by disabling certain pins in the graph completely. Moreover, there are other types of exceptions unhandled in INSTA, including multi-cycle paths, set min/max delays, etc.

[3] https://heterosta.pkueda.org.cn/documentation

[4] CCS support is in our roadmap for next-version release and is not yet available in HeteroSTA 1.0.

TABLE I: Comparison of runtime, timing arc delay correlation (R^2 and MAE) between PrimeTime, OpenSTA, and HeteroSTA. PrimeTime is set as the golden result. Runtimes are in milliseconds.

Benchmark	Statistics			PrimeTime (16C)				OpenSTA (16C)				HeteroSTA (16C + 1 GPU)			
	#Cells	#Nets	#Pins	Runtime	RTR	MAE	R2	Runtime	RTR	MAE	R2	Runtime	RTR	MAE	R2
aes_core	22938	23199	66221	738.70	2.26	0.00	1.000	1092.93	3.34	0.18	0.995	326.91	1.00	0.24	0.988
b19	255278	255300	776320	7165.86	5.69	0.00	1.000	11246.62	8.92	10.02	0.977	1260.33	1.00	4.84	0.985
des_perf	138878	139112	371587	2592.86	3.27	0.00	1.000	4558.99	5.75	0.65	0.981	792.54	1.00	0.54	0.993
edit_dist	147650	150212	416609	3696.77	4.62	0.00	1.000	10136.41	12.66	1.05	0.978	800.55	1.00	1.05	0.984
fft	38158	39184	116139	1065.16	2.88	0.00	1.000	2049.21	5.54	0.68	0.979	370.17	1.00	0.91	0.973
leon2	1616369	1616984	4178874	41594.73	7.56	0.00	1.000	79924.12	14.53	6.01	0.963	5500.07	1.00	1.76	0.993
leon3mp	1247725	1247979	3267993	32634.40	7.34	0.00	1.000	76333.45	17.17	10.00	0.976	4446.62	1.00	4.77	0.985
matrix_mult	164040	167242	475186	3124.04	3.26	0.00	1.000	9964.24	10.41	0.90	0.983	957.29	1.00	1.04	0.976
mgc_edit_dist	161692	164254	444693	5244.76	5.72	0.00	1.000	11713.04	12.77	3.58	0.921	917.38	1.00	2.40	0.988
mgc_matrix_mult	171282	174484	489670	5299.49	6.25	0.00	1.000	9520.60	11.22	2.70	0.932	848.20	1.00	2.40	0.969
netcard	1496719	1498555	3901343	34789.72	6.35	0.00	1.000	80598.09	14.72	9.00	0.984	5476.07	1.00	5.78	0.985
pci_bridge32	40790	40950	108172	1049.68	2.78	0.00	1.000	1539.64	4.08	1.34	0.992	377.05	1.00	0.66	0.998
vga_lcd	259067	259152	662179	6099.74	5.63	0.00	1.000	10167.42	9.39	6.41	0.988	1083.08	1.00	4.00	0.984
Average				11161.22	4.89	0.00	1.000	23757.29	10.04	4.04	0.973	1781.25	1.00	2.34	0.985

TABLE II: Comparison of endpoint slack correlation (R^2) between INSTA [43] and HeteroSTA under simple and complex SDC exceptions. PrimeTime is set as the golden result.

Benchmark	Statistics			Simple SDC Test					Complex SDC Test				
	#Pins	#Edges	#Cells	#FPs	#MCPs	PT	INSTA	HeteroSTA	#FPs	#MCPs	PT	INSTA	HeteroSTA
tau2015_crc32d16N	738	848	246	0	0	1.000	0.996	0.998	0	2	1.000	0.923	0.998
usb_phy_ispd	1759	2105	604	0	0	1.000	0.999	1.000	0	3	1.000	0.962	1.000
aes_core	62142	80420	21347	0	0	1.000	0.999	1.000	17	65	1.000	0.997	1.000
tau2015_softusb_navre	14377	18538	4653	0	0	1.000	1.000	1.000	5	14	1.000	0.987	1.000
pci_bridge32_ispd	87435	106840	30673	0	0	1.000	0.998	0.999	180	911	1.000	0.989	0.999
cordic_ispd	120378	154979	41601	0	0	1.000	1.000	1.000	13	73	1.000	0.953	1.000
fft_ispd	101375	127510	32281	0	0	1.000	1.000	1.000	23	121	1.000	1.000	1.000
tau2015_tip_master	56151	66792	18851	0	0	1.000	1.000	1.000	14	81	1.000	0.698	1.000

PT: PrimeTime. #FPs: number of false path exceptions. #MCPs: number of multi-cycle paths.

INSTA and HeteroSTA both read in SDF delay annotations generated by PrimeTime to bypass delay calculation and only focus on timing propagation and exception handling correctness. R^2 scores under 0.99 but above 0.95 are colored brown, while scores under 0.95 are colored red.

We note that INSTA always calculate top-256 arrival times for every pin due to its CPPR approximation trick and this feature is currently partly hard-coded into its binary CPython extension and thus we were unable to switch it off. As a result, we can only use a set of small benchmarks from TAU 2015 contest to avoid GPU memory overflow in INSTA. We did not report runtime because (1) the benchmarks are too small to provide valuable runtime comparison, and (2) it would be unfair for INSTA since it computes 256× more arrival times.

By generating a `.sdf` file containing all delay annotations, we are able to compare our endpoint slacks with INSTA directly. The results are shown in Table II. We use two sets of design constraints for every benchmark. The simple constraints contain only clock definitions (`create_clock`) and port annotations (`set_input_delay` and `set_output_delay`). The complex constraints additionally contain randomly-sampled `set_false_path` and `set_multicycle_path` exceptions, in approximately 1:5 ratio. With the simple constraints, both INSTA and HeteroSTA can achieve 0.999 correlation with PrimeTime endpoint slacks. However, the correlation of INSTA degrades severely given the complex constraints, down to 0.698 R^2, whereas HeteroSTA maintains 0.999 correlation even with timing exceptions.

C. Case Study: Timing-Driven Global Placement

We integrate HeteroSTA into two state-of-the-art open-source timing-driven global placement flows, DREAMPlace 4.0 [52] and Efficient-TDP [53]. Both of the flows are based on the GPU-accelerated DREAMPlace for global placement and the CPU-based OpenTimer for timing analysis. Their flows suffer from severe runtime overhead due to frequent CPU-GPU data transfer and the translation between flattened and OOP-style data structures.

By simply substituting OpenTimer with HeteroSTA in their flows and leave other algorithms unchanged, we achieve significant end-to-end runtime speedup in both of the flows, as shown in Table III on ICCAD 2015 contest benchmarks [58]. In DREAMPlace 4.0, the runtime of the entire flow has been accelerated by 4.5× on average. In Efficient-TDP, the end-to-end flow runtime has been accelerated by 5.77× on average. All these runtime speedups have come with equal or slightly better WNS and TNS at the end of their optimization loops. These results have proven the necessity and effectiveness of HeteroSTA as a heterogeneous STA library in future fully-heterogeneous physical design flows.

We note that this also demonstrates the generality of HeteroSTA, as DREAMPlace 4.0 and Efficient-TDP use the STA engine differently. DREAMPlace 4.0 is based on back-propagated pin slacks whereas Efficient-TDP is based on top-k critical paths. HeteroSTA supports both STA report styles efficiently and can be applied to a large range of applications. We have released our code changes to DREAMPlace 4.0 and Efficient-TDP on GitHub as HeteroSTA integration demos and

TABLE III: Comparison of TNS ($\times 10^5$ ps), WNS ($\times 10^3$ ps), HPWL ($\times 10^6$), and runtime among DREAMPlace 4.0 [52], Efficient-TDP [53], and their respective integrations with HeteroSTA. Runtimes are in seconds.

Benchmark	DREAMPlace 4.0 [52]				DREAMPlace 4.0 [52] + HeteroSTA				Efficient-TDP [53]				Efficient-TDP [53] + HeteroSTA			
	TNS	WNS	HPWL	Runtime	TNS	WNS	HPWL	Runtime	TNS	WNS	HPWL	Runtime	TNS	WNS	HPWL	Runtime
superblue1	-87.91	-14.23	493.86	526.45	-59.62	-12.58	449.19	112.35	-15.71	-7.77	418.73	594.72	-16.39	-7.65	418.83	129.89
superblue3	-48.74	-15.10	483.06	626.18	-59.28	-15.52	476.63	110.15	-19.98	-11.72	462.68	607.01	-19.88	-11.96	462.54	143.63
superblue4	-145.90	-12.84	334.05	231.97	-150.46	-12.80	333.91	75.42	-87.09	-9.38	317.47	1589.17	-91.69	-8.78	317.72	104.20
superblue5	-95.79	-29.55	536.72	539.41	-92.16	-25.79	523.97	139.39	-61.11	-24.65	477.40	797.55	-62.65	-23.99	483.98	159.44
superblue7	-59.74	-15.22	603.61	736.89	-61.86	-15.22	604.46	174.66	-50.91	-15.22	597.33	791.24	-37.80	-15.22	597.78	200.10
superblue10	-655.36	-23.11	1087.73	917.77	-628.81	-22.17	1042.94	244.52	-559.76	-24.10	911.92	1305.33	-567.71	-23.60	911.38	272.70
superblue16	-63.69	-10.02	459.08	318.42	-51.93	-11.87	467.60	55.66	-21.91	-9.03	474.88	300.53	-18.51	-9.85	463.77	81.25
superblue18	-46.75	-11.53	248.83	279.61	-47.34	-11.73	244.39	48.80	-16.58	-6.73	234.92	277.85	-15.70	-6.83	234.41	59.36
Average Ratio	2.511	1.313	1.080	3.712	2.287	1.292	1.055	0.813	1.058	1.002	1.001	5.772	1.000	1.000	1.000	1.000

Our HeteroSTA integration code for both baselines are released on GitHub: https://github.com/limbo018/DREAMPlace, https://github.com/PKU-IDEA/Efficient-TDP-HeteroSTA. Efficient-TDP had a few bugs that made it non-deterministic, which we fixed to ensure reproducibility.

TABLE IV: The ablation study of timing, power, and congestion in a HeteroSTA-powered timing-driven global routing flow based on HeLEM-GR [34].

ID	Benchmark Name	#Nets	WNS w/o	WNS w	TNS w/o	TNS w	Power w/o	Power w	Congestion w/o	Congestion w	IO RT (s) Read .v	IO RT (s) Read .sdc	Optim. RT (s) w/o	Optim. RT (s) w
1	ariane_v	123900	-0.503	-0.365	-1263.5	-997.9	0.646	0.646	6625796	6662339	1	3	4	6
2	bsg_v	736883	-0.444	-0.388	-10504.2	-9602.9	3.052	3.053	24503278	22947622	9	8	10	19
3	nvdla_v	199481	-67.563	-56.273	-579481.9	-515380.2	2.942	2.938	13251396	13475963	1	2	5	12
4	tile_v	136120	-0.670	-0.387	-3405.3	-1711.6	0.145	0.144	2260103	2689765	1	2	4	7
5	group_v	3274611	-0.443	-0.327	-28230.2	-17619.7	7.643	7.582	70923835	67934255	31	32	44	79
6	cluster_v	12047279	-0.416	-0.247	-85758.7	-52800.0	23.398	23.150	349322586	244407995	136	131	124	261
7	ariane_b	105924	-1.418	-0.645	-444.0	-111.5	0.156	0.156	4473420	4371048	1	2	5	7
8	bsg_b	768239	0.000	0.000	0.0	0.0	0.305	0.305	26635950	22506663	9	8	15	23
9	nvdla_b	157744	0.000	0.000	0.0	0.0	0.136	0.136	13465281	13242875	1	1	6	9
10	tile_b	135814	-0.570	-0.353	-2430.7	-1312.2	0.144	0.144	2136877	2087873	1	2	3	7
11	group_b	3218496	-0.661	-0.404	-47936.6	-27116.0	8.357	8.192	71610825	68117588	31	34	45	78
12	cluster_b	12168735	-0.520	-0.196	-78834.9	-37832.8	24.179	24.021	248823650	235469814	137	140	128	199
	Average Ratio		1.000	0.700	1.000	0.681	1.000	0.996	1.000	0.957			1.000	1.810

w/o: without timing-driven routing; w: with HeteroSTA-powered timing-driven routing.
IO RT: file I/O runtime, including netlist read and database initialization; Optim. RT: the timing-driven routing optimization loop runtime.

for the ease of reproducibility.

D. Case Study: Timing-Driven Global Routing

In addition to global placement, we have also integrated HeteroSTA to other physical design flows such as GPU-accelerated global routing. Our timing-driven router is based on the GPU-accelerated global router HeLEM-GR [34] and powered by HeteroSTA. It has won the first place at the ISPD 2025 performance-driven large-scale global routing contest [59]. Table IV shows an ablation study of timing-driven optimization in the heterogeneous global routing flow. All metrics including WNS, TNS, power, and routing congestion have been improved remarkbly with HeteroSTA in the loop providing real-time timing feedback. Although the one-time initialization (reading netlist and constraints) takes a lot of additional time compared to the flow without timing-driven optimization, we note that optimizations such as persistent database across design stages are possible in a more realistic flow beyond the contest benchmarks.

V. Conclusion

This paper presents HeteroSTA, the first CPU-GPU heterogeneous STA engine with holistic industrial design support, powered by a self-contained advanced Arnoldi delay calculator, rich support for timing exceptions, and a zero-overhead heterogeneous API targeting widespread adoption in heterogeneous EDA flows. As a STA engine, HeteroSTA provides comparable arc delay and slack correlation to leading commercial tools. As a heterogeneous library, HeteroSTA brings remarkable end-to-end runtime speedup to various heterogeneous physical design optimization flows. In the future, we plan to extend HeteroSTA to support other uncovered functionalities in Figure 2, especially CCS and other sign-off related features.

Acknowledge

This work is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

References

[1] H. Qian and Y. Deng, "Accelerating RTL simulation with GPUs," in *Proc. ICCAD*. San Jose, CA, USA: IEEE, 2011, pp. 687–693.
[2] D.-L. Lin, H. Ren, Y. Zhang, and T.-W. Huang, "From RTL to CUDA: A GPU acceleration flow for RTL simulation with batch stimulus," in *Proc. ICPP*, 2022.
[3] Y. Zhang, H. Ren, and B. Khailany, "GL0AM: GPU Logic Simulation Using 0-Delay and Re-simulation Acceleration Method," in *Proc. ICCAD*. New York, NY, USA: IEEE, 2024, pp. 1–9.
[4] Z. Guo, Y. Zhang, R. Wang, Y. Lin, and H. Ren, "GEM: GPU-accelerated emulator-inspired RTL simulation," in *Proc. DAC*. IEEE, 2025.
[5] S. Lin, J. Liu, T. Liu, M. D. F. Wong, and E. F. Y. Young, "Novelrewrite: node-level parallel aig rewriting," in *Proc. DAC*. New York, NY, USA: ACM, 2022, p. 427–432.

[6] G. Pasandi, S. Pratty, D. Brown, Y. Zhang, H. Ren, and B. Khailany, "2021 ICCAD CAD contest problem C: GPU accelerated logic rewriting," in *Proc. ICCAD*. IEEE, 2021, pp. 1–6.

[7] T. Liu and E. F. Young, "Rethinking AIG resynthesis in parallel," in *Proc. DAC*. IEEE, 2023, pp. 1–6.

[8] Y. Sun, T. Liu, M. D. Wong, and E. F. Young, "Massively parallel AIG resubstitution," in *Proc. DAC*, 2024, pp. 1–6.

[9] T. Liu, L. Chen, X. Li, M. Yuan, and E. F. Young, "FineMap: A fine-grained GPU-parallel LUT mapping engine," in *Proc. ASPDAC*, 2024, pp. 392–397.

[10] K. Thorat *et al.*, "GROOT: Graph edge re-growth and partitioning for the verification of large designs in logic synthesis," in *Proc. ICCAD*. IEEE, 2025, pp. 1–6.

[11] B. Goodarzi, M. Burtscher, and D. Goswami, "Parallel graph partitioning on a cpu-gpu architecture," in *Proc. IPDPS Workshops*, 2016, pp. 58–66.

[12] R. Liang, A. Agnesina, and H. Ren, "MedPart: A multi-level evolutionary differentiable hypergraph partitioner," in *Proc. ISPD*, 2024, pp. 3–11.

[13] W. L. Lee, D.-L. Lin, T.-W. Huang, S. Jiang, T.-Y. Ho, Y. Lin, and B. Yu, "G-kway: Multilevel gpu-accelerated k-way graph partitioner," in *Proc. DAC*. ACM, 2024.

[14] Z. Wu, H. Zhao, H. Liu, W. Wen, and J. Li, "gHyPart: GPU-friendly end-to-end hypergraph partitioner," *ACM TACO*, vol. 22, no. 1, pp. 1–25, 2025.

[15] W. L. Lee, D.-L. Lin, C.-H. Chiu, U. Schlichtmann, and T.-W. Huang, "HyperG: Multilevel GPU-accelerated k-way hypergraph partitioner," in *Proc. ASPDAC*. ACM, 2025, p. 1031–1040.

[16] W. L. Lee *et al.*, "iG-kway: Incremental k-way graph partitioning on GPU," in *Proc. DAC*. IEEE, 2025, pp. 1–7.

[17] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE TCAD*, 2020.

[18] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCD-Place: Accelerated batch-based concurrent detailed placement on multi-threaded cpus and gpus," *IEEE TCAD*, 2020.

[19] Z. Guo and Y. Lin, "Differentiable-timing-driven global placement," in *Proc. DAC*. ACM, 2022, p. 1315–1320.

[20] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao, B. Keller, B. Khailany, and H. Ren, "Autodmp: Automated dreamplace-based macro placement," in *Proc. ISPD*, 2023, pp. 149–157.

[21] L. Liu, B. Fu, S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Xplace: An extremely fast and extensible placement framework," *IEEE TCAD*, vol. 43, no. 6, pp. 1872–1885, 2023.

[22] L. Liu, B. Fu, S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Xplace: An extremely fast and extensible placement framework," *IEEE TCAD*, 2023.

[23] A. B. Kahng and Z. Wang, "Dg-replace: A dataflow-driven gpu-accelerated analytical global placement framework for machine learning accelerators," *IEEE TCAD*, 2024.

[24] Y. Du, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Fusion of global placement and gate sizing with differentiable optimization," in *Proc. ICCAD*. ACM, 2024.

[25] B. Fu, L. Liu, M. D. F. Wong, and E. F. Y. Young, "Hybrid modeling and weighting for timing-driven placement with efficient calibration," in *Proc. ICCAD*, 2024.

[26] Y. Du, Z. Guo, R. Wang, and Y. Lin, "Differentiable physical optimization," in *Proc. ICCAD*. IEEE, 2025, pp. 1–6.

[27] C.-H. Lu, W.-H. Liu, H. Ren, and T.-C. Wang, "Leveraging GPU for better detailed placement quality," in *Proc. ICCAD*. IEEE, 2025, pp. 1–6.

[28] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "GAMER: GPU-accelerated maze routing," *IEEE TCAD*, vol. 42, no. 2, pp. 583–593, 2022.

[29] Z. Guo, F. Gu, and Y. Lin, "GPU-accelerated rectilinear steiner tree generation," in *Proc. ICCAD*. ACM, 2022.

[30] S. Liu *et al.*, "FastGR: Global routing on CPU–GPU with heterogeneous task graph scheduler," *IEEE TCAD*, vol. 42, no. 7, pp. 2317–2330, 2022.

[31] S. Lin, L. Xiao, J. Liu, and E. F. Young, "InstantGR: Scalable GPU parallelization for global routing," in *Proc. ICCAD*, 2024, pp. 1–8.

[32] L. Xiao, S. Lin, J. Liu, Q. Duan, T.-Y. Ho, and E. F. Young, "InstantGR: Scalable GPU parallelization for 3-d global routing," *IEEE TCAD*, 2025.

[33] W. Li, R. Liang, A. Agnesina, H. Yang, C.-T. Ho, A. Rajaram, and H. Ren, "DGR: Differentiable global router," in *Proc. DAC*, 2024, pp. 1–6.

[34] C. Zhao, Z. Guo, R. Wang, Z. Wen, Y. Liang, and Y. Lin, "HeLEM-GR: Heterogeneous global routing with linearized exponential multiplier method," in *Proc. ICCAD*, 2024, pp. 1–9.

[35] R. Liang, A. Agnesina, W.-H. Liu, M. Liberty, H.-T. Chang, and H. Ren, "ISPD 2025 performance-driven large scale global routing contest," in *Proc. ISPD*, 2025, pp. 252–256.

[36] C. Zhao, J. Wang, X. Jiang, J. Lou, and Y. Lin, "GTA: GPU-accelerated track assignment with lightweight lookup table for conflict detection," in *Proc. ICCAD*, 2025, pp. 1–9.

[37] Z. He, Y. Ma, and B. Yu, "X-Check: GPU-accelerated design rule checking via parallel sweepline algorithms," in *Proc. ICCAD*, 2022, pp. 1–9.

[38] Z. He, Y. Zuo, J. Jiang, H. Zheng, Y. Ma, and B. Yu, "OpenDRC: An efficient open-source design rule checking engine with hierarchical GPU acceleration," in *Proc. DAC*. IEEE, 2023, pp. 1–6.

[39] Z. Guo, T.-W. Huang, and Y. Lin, "GPU-accelerated static timing analysis," in *Proc. ICCAD*. ACM, 2020.

[40] Z. Guo, T.-W. Huang, Z. Jin, C. Zhuo, Y. Lin, R. Wang, and R. Huang, "Heterogeneous static timing analysis with advanced delay calculator," in *Proc. DATE*, 2024.

[41] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. Young, and M. Wong, "GCS-timer: GPU-accelerated current source model based static timing analysis," in *Proc. DAC*. ACM, 2024.

[42] Z. Guo, T.-W. Huang, and Y. Lin, "Accelerating static timing analysis using CPU-GPU heterogeneous parallelism," *IEEE TCAD*, pp. 1–1, 2023.

[43] 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 *Proc. DAC*. IEEE, 2025.

[44] H. Liu, Z. Guo, R. Wang, and Y. Lin, "IncreGPUSTA: GPU-accelerated incremental static timing analysis for iterative design flows," in *Proc. ICCAD*. IEEE, 2025, pp. 1–6.

[45] Z. Guo *et al.*, "HeteroExcept: A CPU-GPU heterogeneous algorithm to accelerate exception-aware static timing analysis," in *Proc. ICCAD*. ACM, 2024.

[46] Z. Guo, T.-W. Huang, and Y. Lin, "HeteroCPPR: Accelerating common path pessimism removal with heterogeneous CPU-GPU parallelism," in *Proc. ICCAD*. ACM/IEEE, 2021.

[47] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "GPU-accelerated path-based timing analysis," in *Proc. DAC*. ACM, 2021.

[48] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. D. Wong, "A GPU-accelerated framework for path-based timing analysis," *IEEE TCAD*, 2023.

[49] C. Chang *et al.*, "PathGen: An efficient parallel critical path generation algorithm," in *Proc. ASPDAC*. IEEE, 2025.

[50] T. Huang, G. Guo, C. Lin, and M. D. F. Wong, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, vol. 40, no. 4, pp. 776–786, 2021.

[51] "OpenSTA," https://github.com/The-OpenROAD-Project/OpenSTA.

[52] P. Liao, D. Guo, Z. Guo, S. Liu, Y. Lin, and B. Yu, "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*, vol. 42, no. 10, pp. 3374–3387, 2023.

[53] Y. Shi, S. Xu, S. Kai, X. Lin, K. Xue, M. Yuan, and C. Qian, "Timing-driven global placement by efficient critical path extraction," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.

[54] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[55] "Synopsys PrimeTime," https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html.

[56] "Cadence Tempus," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/tempus-timing-signoff-solution.html.

[57] J. Hu, G. Schaeffer, and V. Garg, "TAU 2015 contest on incremental timing analysis," in *Proc. ICCAD*. IEEE, 2015, pp. 882–889.

[58] M. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. ICCAD*, 2015, pp. 921–926.

[59] R. Liang, A. Agnesina, W.-H. Liu, M. Liberty, H.-T. Chang, and H. Ren, "Invited: ISPD 2025 performance-driven large scale global routing contest," in *Proc. ISPD*. New York, NY, USA: ACM, 2025.