ElmoreCeff: A GPU-Friendly Elmore-Like Delay Calculator with a Closed-Form Effective Capacitance Model

Haichuan Liu[1,†], Zizheng Guo[1,2,†], 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
lhaic@outlook.com, {gzz, r.wang, yibolin}@pku.edu.cn

***Abstract*—Accurate and efficient delay calculation is critical for timing-driven optimization, yet existing methods require a trade-off between fast but inaccurate Elmore timers and accurate but slow model order reduction (MOR) techniques. To bridge this gap, we introduce ElmoreCeff, a GPU-friendly Elmore-like delay calculator based on a closed-form effective capacitance model. ElmoreCeff can achieve speedups of up to $40.95\times$ and $5.95\times$ over CPU- and GPU-based MOR timers, respectively, while achieving comparable accuracy. This advantage enhances final quality of results (QoR) of timing optimization, improving worst negative slack (WNS) and total negative slack (TNS) by up to 8.4% and 5.4% compared to optimizations guided by the standard Elmore timer.**

I. Introduction

Static timing analysis (STA) is a fundamental pillar of the VLSI design flow, validating circuit functionality and guiding timing-driven optimization at every stage [1]. However, as semiconductor technology scales to advanced nodes, dominant interconnect effects, particularly resistive shielding, render traditional delay models highly inaccurate. To ensure signoff confidence, modern timing analysis therefore relies on effective capacitance (Ceff) [2] to accurately capture the interaction between interconnect parasitics and signal propagation. The accurate and efficient calculation of Ceff has thus become a non-negligible challenge in modern STA.

This challenge can be mitigated by advanced cell library models, such as Composite Current Source (CCS) models. While CCS models provide superior accuracy, their computational expense makes them impractical for the thousands of timing queries required in an optimization loop. As a result, the entire optimization flow relies on the faster Non-Linear Delay Model (NLDM) cell model. Therefore, a difficult trade-off between speed and accuracy has emerged, as summarized in Table I. On one hand, the classic Elmore delay model is computationally cheap and widely adopted in open-source timers like OpenTimer [3], [4]. However, its accuracy degrades severely under strong resistive shielding rendering it unreliable at advanced nodes. On the other hand, advanced Model Order Reduction (MOR) techniques, such as the Arnoldi algorithm, deliver high accuracy by implicitly modeling the circuit's transient response, which captures the effects of Ceff [5], [6]. Yet, these methods are orders of magnitude slower than analytical models like Elmore, due to their iterative, matrix-based nature.

The inherent structure of those advanced models also poses significant challenges for parallelization. The iterative procedures for Ceff calculation involve complex linear algebra and data-dependent convergence patterns, leading to irregular workloads and thread divergence. These characteristics make such methods difficult to map efficiently onto the single-instruction, multiple-thread (SIMT) execution model of GPUs, even though GPU acceleration can still improve throughput. This is particularly problematic given the accelerating trend of leveraging GPUs to tackle the immense computational complexity of modern EDA flows. This trend is evident across the entire design automation spectrum, with seminal works demonstrating massive speedups in physical design—including analytical and detailed placement [7], [8], [9], [10], [11] and global routing [12], [13], [14]—as well as in logic and RTL simulation [15], [16], [17], [18].

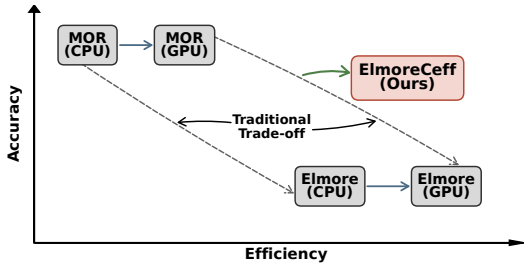


Fig. 1: GPU acceleration shifts MOR and Elmore rightward in efficiency, yet both the CPU and GPU MOR-to-Elmore trajectories still lie on the traditional accuracy-efficiency trade-off. ElmoreCeff instead occupies the previously missing region of high efficiency with improved accuracy.

Within static timing analysis itself, numerous computation components have been successfully mapped to the GPU architecture, including various methods for delay calculation [19], [5], [20], timing propagation [21], [22], [23], path analysis [24], [25], [26], statistical timing [27], [28], and path exception handling [29]. This widespread adoption underscores a clear industry and academic trajectory towards a full-stack, GPU-native optimization toolchain. However, the absence of a delay calculator that is simultaneously accurate, fast, and GPU-friendly forces a reliance on slow or CPU-based methods for timing-critical decisions.

This gap creates a severe practical bottleneck in timing-driven optimization flows. To address this, a compelling methodology involves developing a "proxy-golden" timer that is fast yet accurate enough to guide optimization directly [30], [31]. An emerging technique to create such timers is calibration, where a lightweight timer is correlated or annotated against a signoff reference timer [32], [33]. While these calibration frameworks are powerful, their ultimate effectiveness still hinges on the physical fidelity of the simple

† Equal contribution, * Corresponding author.

TABLE I: Comparison of Delay Calculation Methodologies with the NLDM Cell Model.

	Elmore [4]	**MOR** [5], [6]	**ElmoreCeff (Ours)**
Speed	**Fast**	*Moderate*	**Fast**
Accuracy	*Low*	**High**	**Moderate**
Effective Cap.	×	✓	✓
Implementation	**Easy**	*Difficult*	**Easy**
GPU-Friendly	✓	*Limited*	✓

timer's core delay model.

To overcome this challenge and the traditional trade-off between speed and accuracy (Figure 1), we introduce **ElmoreCeff**, a novel GPU-friendly Elmore-like delay calculator based on a closed-form effective capacitance model. Our primary contributions are summarized as follows:

1) A closed-form effective capacitance model for a basic circuit, eliminating the iterative overhead of conventional methods.
2) An Elmore-like Ceff calculation algorithm structurally isomorphic to Elmore traversal.
3) A GPU-friendly delay calculator that is Ceff-aware yet non-iterative.

We evaluate ElmoreCeff on both academic and industrial benchmarks [34], [35]. Experimental results demonstrate that ElmoreCeff achieves speedups of up to 40.95× and 5.95× over CPU- and GPU-based MOR timers, while offering higher accuracy than the Elmore model. This directly enhances the final quality of results (QoR) from timing optimization, yielding improvements of up to 8.4% in worst negative slack (WNS) and 5.4% in total negative slack (TNS) over a standard Elmore timer.

The rest of this paper is organized as follows. Section II introduces the background of delay and effective capacitance calculation. Section III details our closed-form Ceff model and the corresponding delay calculation algorithms. Section IV presents our experimental results and analysis. Finally, Section V concludes the paper and discusses future work.

II. Preliminary

In this section, we introduce the Non-Linear Delay Model (NLDM), establish the critical need for effective capacitance (Ceff) to ensure timing accuracy, and detail the computational bottlenecks of conventional iterative methods for Ceff calculation.

A. NLDM and the Need for Ceff

In modern STA, the timing behavior of standard cells is commonly pre-characterized and stored in libraries with the Non-Linear Delay Model (NLDM). With the NLDM model, cell delays and output slews are captured in two-dimensional look-up tables (LUTs), typically indexed by input transition time and output load capacitance:

$$\text{delay} = \text{LUT}_{\text{delay}}(slew_{\text{in}}, C_{\text{load}}), \tag{1}$$

$$\text{slew} = \text{LUT}_{\text{slew}}(slew_{\text{in}}, C_{\text{load}}). \tag{2}$$

Therefore, the fidelity of the NLDM model is critically dependent on the accuracy of its inputs, particularly the load capacitance C_{load}.

The most straightforward method for determining C_{load} is to use the total capacitance of the downstream interconnect tree . This approach aligns with the characterization of NLDM libraries, which are built using purely capacitive loads. While computationally simple, this approach has a critical drawback in advanced process nodes: it ignores the *resistive shielding effect*. Interconnect resistance "shields" the driver from far-end capacitances. Those capacitances do not contribute their full value to the effective load. Consequently, using C_{total} overestimates the load, leading to pessimistic and inaccurate delay and slew predictions from the LUTs.

To resolve this, the concept of effective capacitance (Ceff) was introduced [2]. Ceff provides a single capacitance value that accurately represents the loading effect of the entire RC interconnect. Formally, Ceff is defined as the capacitance that draws the same amount of charge from the driver over the signal transition period as the full RC network. Prior analytical Ceff approximations were also explored [36], [37], but they rely on simplified assumptions or iterative refinement.

B. Conventional Iterative Ceff Calculation

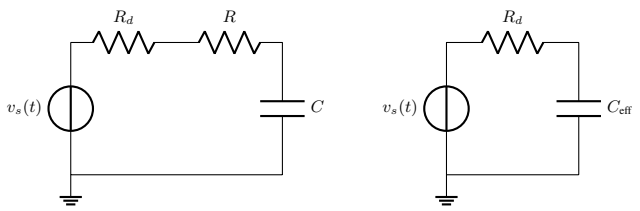
Despite these analytical approximations, modern high-accuracy Ceff calculation still relies primarily on computationally expensive iterative numerical procedures.

The foundational iterative approach, established by Dartu et al. [38], models the driving gate as a time-varying Thevenin equivalent source characterized by a driver resistance R_d and a ramped voltage source $v_s(t)$. It then solves for C_{eff} together with the source parameters $(t_0, \Delta t)$ through the following coupled nonlinear system:

$$\begin{cases} Q_{\text{real}}(\Delta t, t_{\text{av}}) = Q_{C_{\text{eff}}}(\Delta t, t_{\text{av}}, C_{\text{eff}}) \\ V(t_{50}(C_{\text{eff}}), t_0, \Delta t) \equiv 0.5 \cdot V_{\text{DD}} \\ V(t_{20}(C_{\text{eff}}), t_0, \Delta t) = \begin{cases} 0.2 \cdot V_{\text{DD}} & \text{if rising transition,} \\ 0.8 \cdot V_{\text{DD}} & \text{if falling transition.} \end{cases} \end{cases}, \tag{3}$$

where the first equation enforces charge conservation, and the latter two constrain the simplified waveform to match key voltage points from the library characterization. Because these equations are interdependent, the system is typically solved using a two-level iterative solver (e.g., Newton-Raphson), which alternates between updating C_{eff} and the source parameters until convergence. This process, which must be executed for every net, is computationally intensive and a primary source of slowdown in advanced timers.

More recent state-of-the-art timers employ advanced Model Order Reduction (MOR) techniques, sometimes with GPU acceleration, to enhance accuracy and speed [6], [5]. In this approach, the RC interconnect is first represented as a system of linear differential equations using Modified Nodal Analysis (MNA). A projection-based MOR method, such as the Arnoldi algorithm, then reduces this large-scale system into a highly accurate, fixed-size model. Although GPU acceleration can improve the throughput of these matrix-heavy computations, the fundamental iterative complexity of the algorithm remains. Consequently, its runtime is still unsuitable for the high-throughput demands of timing-driven optimization workloads.



(a) Full RC Circuit (b) Equivalent Ceff Circuit

Fig. 2: Comparison of the full 1R1C circuit (a) and its effective capacitance model (b).

III. Algorithm

This section presents our non-iterative delay calculation framework, ElmoreCeff. At the heart of our approach is a closed-form analytical model for effective capacitance, which enables an algorithm isomorphic to the classic Elmore delay calculation. We will detail the model's theoretical derivation on a simple circuit, its generalization to arbitrary RC trees, and the refined slew propagation model to ensure accuracy.

A. Closed-Form Ceff Model for 1R1C Circuits

The key to avoiding the computational cost of iterative Ceff solvers is to formulate a non-iterative, analytical solution. We begin this process by deriving a closed-form expression for the basic but representative 1R1C circuit.

As depicted in Fig. 2(a), the original circuit consists of a driving resistance R_d, a load resistance R, and a load capacitance C. The input voltage $v_s(t)$ is modeled as a linear ramp from 0 to V_{DD} over a duration of T_s. Fig. 2(b) shows the equivalent circuit where the RC load is replaced by a single effective capacitance C_{eff}. The definition of C_{eff} is based on the principle of charge conservation: the total charge, Q, drawn from the driving source over the input ramp duration T_s must be identical for both the full RC network and its simplified $R_d - C_{\text{eff}}$ equivalent. This principle is expressed in its integral form:

$$Q_{\text{real}} = \int_0^{T_s} I_{\text{real}}(t)dt = \int_0^{T_s} I_{\text{eff}}(t)dt = Q_{\text{eff}}, \tag{4}$$

where $I_{\text{real}}(t)$ and $I_{\text{eff}}(t)$ are the currents drawn from the cell in the real and effective circuits, respectively.

Solving these integrals for the transient response of the RC circuits reveals that the total charge drawn from the source follows a common mathematical structure. We define this general function, which depends on a variable capacitance C_{var} and resistance R_{var}:

$$h(C_{\text{var}}, R_{\text{var}}) = C_{\text{var}} \left(1 + \frac{R_{\text{var}}C_{\text{var}}}{T_s}\left(e^{-\frac{T_s}{R_{\text{var}}C_{\text{var}}}} - 1\right)\right), \tag{5}$$

where T_s is the duration of the input signal ramp. Using this function, Eq. (4) can be expressed in a compact form:

$$h(C, R + R_d) = h(C_{\text{eff}}, R_d). \tag{6}$$

An analytical solution to Eq. (6) requires approximating its exponential term. While a Taylor series is a common choice, it is a local approximation, and its accuracy degrades rapidly away from the point of expansion. We therefore employ the Padé approximant [39], [40], a rational function known to provide superior accuracy over a wider domain than a truncated Taylor series.

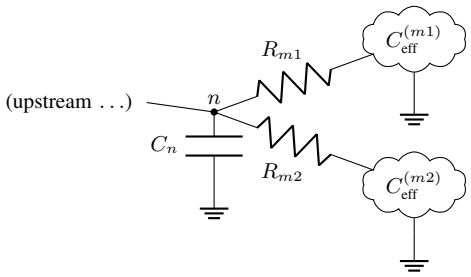


Fig. 3: From the perspective of node n, each downstream sub-tree is modeled as a single pre-computed effective capacitance.

The most widely used and effective rational approximation for the exponential function is the [1,1] Padé approximant:

$$e^{-z} = \frac{1 - z/2}{1 + z/2} + O(z^3). \tag{7}$$

This approximant is highly accurate, matching the Taylor series of e^{-z} up to the quadratic term. Applying the Padé approximant from Eq. (7) to the exponential term in Eq. (5) yields a simple and accurate rational approximation for the function h:

$$h(C_{\text{var}}, R_{\text{var}}) \approx \frac{C_{\text{var}}T_s}{T_s + 2R_{\text{var}}C_{\text{var}}}. \tag{8}$$

The approximation is most reliable for small to moderate $\zeta = R_{\text{var}}C_{\text{var}}/T_s$, while very large ζ can increase the mismatch.

Applying this accurate approximation to both sides of the original equality in Eq. (6), and solving for C_{eff} reveals a final simplification. After cross-multiplication, the driver resistance term R_d is analytically eliminated from the equation, leading to the proposed closed-form Ceff model:

$$C_{\text{eff}} = \frac{CT_s}{T_s + 2RC}. \tag{9}$$

This expression, where C and R represent the real interconnect capacitance and resistance, forms the core of the proposed efficient delay calculator. It provides a direct analytical mapping from circuit parameters to effective capacitance without any iterative overhead.

B. Generalization via an Elmore-like Traversal

The true power of our analytical 1R1C model lies in its generalization to arbitrary RC trees. To achieve this, we need a systematic method to traverse the tree and aggregate the capacitive effects from all downstream branches. We introduce a recursive algorithm whose computational structure is isomorphic to the classic Elmore delay framework. This structural property not only ensures computational efficiency but also enables seamless integration into existing timing engines.

The generalization hinges on a key physical insight derived from Kirchhoff's Current Law (KCL). At any node n, the total charge that must be supplied is simply the sum of the charge stored in its local grounded capacitance, C_n, and the charge delivered into each downstream branch (Fig. 3). Since effective capacitance is defined by this total charge, it naturally follows a principle of superposition: the total effective capacitance at node n, $C_{\text{eff}}^{(n)}$, is the direct sum of its local capacitance and the contributions from each child branch.

This superposition principle forms the basis for a recursive algorithm. The core challenge is that the load from each child

branch is not a simple capacitor but a complex sub-tree. Our algorithm resolves this by decomposing the problem: for each node, we first recursively compute the effective capacitance of the entire sub-tree rooted below it. Then the entire complex sub-tree at m is reduced to a single capacitive load $C_{\text{eff}}^{(m)}$.

With this abstraction, the contribution of the branch connecting n to m can be calculated using our 1R1C model. The total effective capacitance at node n is thus given by:

$$C_{\text{eff}}^{(n)} = C_n + \sum_{m \in \text{children}(n)} C_{\text{eff}}^{(\text{branch } n \to m)}, \tag{10}$$

where $C_{\text{eff}}^{(\text{branch } n \to m)}$ is the effective capacitance contribution from the branch, calculated using our 1R1C model from Eq. (9) with the sub-tree's capacitance $C_{\text{eff}}^{(m)}$ acting as the load and R_m as the interconnect resistance. Substituting our closed-form solution yields the final recursive update rule:

$$C_{\text{eff}}^{(n)} = C_n + \sum_{m \in \text{children}(n)} \frac{C_{\text{eff}}^{(m)} T_n}{T_n + 2R_m C_{\text{eff}}^{(m)}}. \tag{11}$$

In this formulation, T_n is the effective ramp duration at node n (derived from its slew), and R_m is the resistance of the interconnect branch between nodes n and m.

The recursive rule in Eq. (11) is implemented via an efficient post-order traversal, as detailed in Algorithm 2. The key insight is that the bottom-up computational flow is structurally isomorphic to the capacitance summation pass of the classic Elmore Algorithm 1.

This isomorphism allows our method to function as a highly efficient refinement pass appended after a conventional Elmore analysis, leveraging the initially computed Elmore slew values ($slew[n]$) to calculate a more accurate effective capacitance. Because the traversal structure is identical, the pass can be integrated seamlessly into existing parallel timing engines and reuse the same optimized data structures and parallelization strategies with minimal code modification.

Note that in Algorithm 2, the ramp duration T_n for a branch (m, n) is derived from the parent node's slew $slew[m]$ using a `ramp_factor` (e.g., a 10%-90% slew definition yields a factor of 0.8).

C. Refined Slew Modeling and Propagation

The accuracy of the recursive Ceff model in Eq. (11) is critically dependent on the accurate slew values at each node, for which conventional slew models, as used in open-source timers [3], [4], [6], are insufficient. This section thus details a refined slew model, based on the probabilistic interpretation of a circuit's transient response.

The foundation of this framework is to model a circuit's step response as a cumulative distribution function (CDF), and the impulse response as a probability density function (PDF) [41]. As depicted in Fig. 4, with a step input, the output slew (T_S) can be defined from the second central moment (variance, σ_h^2) of the impulse response:

$$T_S^2 = \text{constant} \times \int_0^\infty (t - T_D)^2 h'(t) \mathrm{d}t = \text{constant} \cdot \sigma_h^2. \tag{12}$$

The constant can be adapted to different waveforms, such as 2π (for Gaussian) or $\ln(9)$ (for 10-90% definitions) [42].

Algorithm 1 Elmore Delay Calculation for a RC Tree

slew_factor $\leftarrow \sqrt{2\pi}$ for slow corners, else 1.0
Initialize arrays $load$, $delay$, $ldelay$, $beta$, $slew$ to all zeros
for each node n in T in post-order **do**
  $load[n] \leftarrow load[n] + C_n$
  $m \leftarrow$ father of n
  $load[m] \leftarrow load[m] + load[n]$
end for
for each node n in T in pre-order **do**
  $m \leftarrow$ father of n
  $R_n \leftarrow$ resistance of branch (m, n)
  $delay[n] \leftarrow delay[m] + R_n \cdot load[n]$
end for
for each node n in T in post-order **do**
  $ldelay[n] \leftarrow ldelay[n] + C_n \cdot delay[n]$
  $m \leftarrow$ father of n
  $ldelay[m] \leftarrow ldelay[m] + ldelay[n]$
end for
$slew_i \leftarrow$ slew of cell output pin from LUT
for each node n in T in pre-order **do**
  $m \leftarrow$ father of n
  $R_n \leftarrow$ resistance of branch (m, n)
  $beta[n] \leftarrow beta[m] + R_n \cdot ldelay[n]$
  $impulse[n] \leftarrow 2 \cdot beta[n] - delay[n]^2$
  $slew[n] \leftarrow \sqrt{slew_i^2 + \text{slew\_factor}^2 \cdot impulse[n]}$
end for
return $delay$, $slew$

Algorithm 2 Elmore-like C_{eff} Calculation for a RC Tree

ramp_factor $\leftarrow 0.8$
Initialize array C_{eff} to all zeros
for each node n in T in post-order **do**
  $C_{\text{eff}}[n] \leftarrow C_{\text{eff}}[n] + C_n$
  $m \leftarrow$ father of n
  $R_n \leftarrow$ resistance of branch (m, n)
  $T_n \leftarrow slew[m]/\text{ramp\_factor}$
  $C_{\text{eff}}[m] \leftarrow C_{\text{eff}}[m] + \dfrac{C_{\text{eff}}[n] \cdot T_n}{T_n + 2 \cdot R_n \cdot C_{\text{eff}}[n]}$
end for
return C_{eff}

To extend this model from an ideal step input to arbitrary, finite-slew inputs, we leverage the properties of convolution for linear time-invariant (LTI) systems [43]. For an LTI system, the output response derivative, $y'(t)$, is the convolution of the input signal derivative, $u'(t)$, and the system's impulse response, $h(t)$. The variances of the constituent PDFs add linearly [44]:

$$\sigma_{y'}^2 = \sigma_{u'}^2 + \sigma_h^2, \tag{13}$$

where $\sigma_{y'}^2$, $\sigma_{u'}^2$, and σ_h^2 are the variances of the output, input, and impulse responses, respectively. Since variance is proportional to the square of the slew, a robust sum-of-squares formula for slew propagation is then derived as follows:

$$\text{slew(output)}^2 = \text{slew(input)}^2 + \text{slew(interconnect)}^2. \tag{14}$$

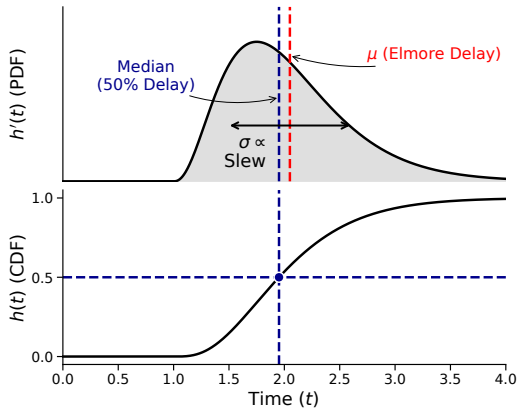


Fig. 4: Conceptual illustration of the probabilistic interpretation of transient response. The step response, $h(t)$, is modeled as a CDF, and its derivative, $h'(t)$, as a PDF.

TABLE II: Statistics of Benchmark Circuits.

Benchmark	#Gates	#Nets	#Pins
tau2015_crc32d16N	478	495	1,251
usb_phy_ispd	923	938	2,447
tau2015_softusb_navre	6,943	6,978	19,546
cordic_ispd	45,359	45,393	127,993
des_perf_ispd	138,878	139,112	371,587
edit_dist_ispd	147,650	150,212	416,609
mgc_edit_dist_iccad	161,692	164,254	444,693
mgc_matrix_mult_iccad	171,282	174,484	489,670
ARM_IP1	376,171	405,056	1,528,435
ARM_IP2	500,177	588,792	2,064,589
ARM_IP3	372,058	390,011	1,492,768

The interconnect's intrinsic slew contribution, slew(interconnect), can be computed from the first and second moments of the circuit, as demonstrated in Algorithm 1:

$$\text{slew}[n] = \sqrt{\text{slew}_i{}^2 + \text{slew\_factor}^2 \cdot (2 \cdot beta[n] - delay[n]^2)}, \tag{15}$$

where `slew_factor` is the square root of the `constant` from Eq. 12. This formulation correctly integrates the input slew effect with the interconnect's intrinsic response.

Furthermore, using a single fixed `slew_factor` across timing corners is suboptimal [44]. We therefore adopt corner-dependent factors. This choice is heuristic, but is motivated by the differing skewness of transient response waveforms across process corners. Specifically, highly skewed slow corners benefit from a larger factor (e.g., $\sqrt{2\pi}$) to better capture slew degradation, while less-skewed fast corners are better modeled with a smaller factor of 1.0, which helps avoid pessimistic over-estimation. In practice, this differentiation improves the robustness and accuracy of the model across corners.

IV. Experimental Results

We implemented our Elmore-like Ceff-aware delay calculator **ElmoreCeff** within the CPU-GPU heterogeneous STA library, `HeteroSTA` [22], [5], [45] and conducted all experiments on a system with two Intel Xeon 8358 CPUs and an NVIDIA A800 GPU. Our evaluation uses a suite of benchmarks comprising circuits from the TAU contest [34], re-synthesized with a commercial 14nm PDK, and larger industrial ARM IPs [35], implemented with a commercial 7nm PDK. For comparison, we benchmark against Synopsys `PrimeTime` [46], `OpenSTA` [6], as well as the Elmore and Arnoldi timers within `HeteroSTA`. For all comparisons, Synopsys `PrimeTime` serves as the golden reference, using extracted SPEF files to ensure a signoff-standard evaluation (Sec IV-A and IV-B).

A. Standalone Timer Performance

We compared the runtime and accuracy of all previously mentioned timers. PrimeTime and OpenSTA were configured to use 16 CPU threads and their respective Arnoldi-based models [1] [2]. The HeteroSTA-based timers utilized 16 CPU threads and 1 GPU with their corresponding models. We then measured the runtime of `report_timing` calls, with results presented in Tables III and IV.

In terms of runtime, our ElmoreCeff is significantly faster than industrial-standard timers, achieving an average speedup of 3.44× over `PrimeTime` and 13.63× over `OpenSTA`. It is also highly competitive with other GPU timers, running nearly as fast as the basic Elmore model and 2.62× faster than Arnoldi. Crucially, this efficiency does not come at the cost of accuracy. Our model achieves an average R^2 score of 0.965, dramatically improving upon the Ceff-unaware Elmore model (0.948) with negligible runtime overhead, and approaching the accuracy of the much slower `OpenSTA` (0.976) and the Arnoldi (0.989) timer.

B. Impact on Timing-Driven Optimization

To measure the impact of our timer's guidance on final quality of results (QoR), we implemented a simple timing-driven optimization flow, based on the repair timing methodology of OpenROAD [47], [48]. Starting from an identical initial placement for each design, the flow was guided with either the naive Elmore or our ElmoreCeff timer for comparison. This experiment was conducted on three industrial ARM IPs, including two CPU cores and one NPU core. The TAU contest circuits were excluded as the LEF/DEF files required by optimization are not provided. The final WNS and TNS were then reported by `PrimeTime` using the pre-route SPEF files extracted after the optimization flow.

As summarized in Table V, the ElmoreCeff guided flow consistently achieves superior final QoR across all three industrial ARM IPs. On `ARM_IP1`, the Ceff-guided flow achieves a 76.0 ns TNS reduction and a 3.3 ps WNS improvement. The benefit is also clear on `ARM_IP3`, where the flow improves the final WNS by 24.8 ps (8.4%) and reduces TNS by 25.4 ns (5.4%). These results demonstrate that the superior accuracy of our delay model directly translates to higher quality results in a timing-driven optimization context.

V. Conclusion

ElmoreCeff is a GPU-friendly Elmore-like delay calculator based on a closed-form effective capacitance model. It recovers part of the accuracy gap to MOR-based and signoff tools, delivering up

[1]For PrimeTime, the commands `set rc_driver_model_mode basic` and `set rc_receiver_model_mode basic` were used.

[2]For OpenSTA, `set_delay_calculator arnoldi` was used.

TABLE III: Timing Report Runtime and Speedup Comparison of Timing Engines.

Benchmark	PrimeTime		OpenSTA		Arnoldi		Elmore		ElmoreCeff	
	Runtime	RTR	Runtime	RTR	Runtime	RTR	Runtime	RTR	Runtime	RTR
mgc_matrix_mult_iccad	5628.28	5.98x	38549.86	40.95x	4022.38	4.27x	859.66	0.91x	941.30	1.00x
mgc_edit_dist_iccad	6019.92	6.47x	14145.95	15.21x	3460.30	3.72x	780.58	0.84x	930.16	1.00x
tau2015_softusb_navre	358.65	1.03x	579.16	1.67x	1178.56	3.39x	319.52	0.92x	347.21	1.00x
edit_dist_ispd	4968.60	5.39x	14325.10	15.53x	2323.13	2.52x	912.35	0.99x	922.27	1.00x
tau2015_crc32d16N	180.94	0.68x	111.42	0.42x	340.18	1.29x	261.54	0.99x	264.63	1.00x
cordic_ispd	1686.65	3.53x	2131.12	4.47x	2839.06	5.95x	472.90	0.99x	477.18	1.00x
usb_phy_ispd	225.52	0.89x	82.81	0.33x	388.98	1.54x	260.46	1.03x	252.98	1.00x
des_perf_ispd	2611.32	3.20x	10686.34	13.10x	1484.11	1.82x	680.15	0.83x	815.93	1.00x
ARM_IP1	10731.95	3.61x	95491.52	32.16x	4985.97	1.68x	3122.83	1.05x	2969.69	1.00x
ARM_IP2	15866.16	3.76x	57940.47	13.72x	5649.97	1.34x	4292.15	1.02x	4223.22	1.00x
ARM_IP3	10849.51	3.33x	40193.65	12.35x	4388.14	1.35x	2944.63	0.90x	3255.17	1.00x
Average	5375.23	3.44x	24930.67	13.63x	2823.71	2.62x	1355.16	0.95x	1399.98	1.00x

Runtime: in ms; RTR: Runtime Ratio over ElmoreCeff (ours).

TABLE IV: Accuracy Comparison of Timing Engines vs. Golden PrimeTime Results.

Benchmark	PrimeTime		OpenSTA		Arnoldi		Elmore		ElmoreCeff	
	R^2	MAE	R^2	MAE	R^2	MAE	R^2	MAE	R^2	MAE
mgc_matrix_mult_iccad	1.000	0.000	0.9405	2.695	0.9764	2.094	0.8972	5.046	0.9470	3.038
mgc_edit_dist_iccad	1.000	0.000	0.9341	3.542	0.9928	1.959	0.9129	5.969	0.9478	4.031
tau2015_softusb_navre	1.000	0.000	0.9642	1.333	0.9744	1.235	0.9343	2.238	0.9616	1.510
edit_dist_ispd	1.000	0.000	0.9789	1.048	0.9901	0.731	0.9139	3.274	0.9313	2.603
tau2015_crc32d16N	1.000	0.000	0.9827	0.680	0.9789	0.897	0.9573	1.544	0.9744	1.072
cordic_ispd	1.000	0.000	0.9842	0.719	0.9958	0.589	0.9454	1.479	0.9572	1.069
usb_phy_ispd	1.000	0.000	0.9858	0.614	0.9880	0.516	0.9469	1.300	0.9561	1.144
des_perf_ispd	1.000	0.000	0.9808	0.673	0.9962	0.450	0.9358	1.334	0.9444	1.061
ARM_IP1	1.000	0.000	0.9957	0.381	0.9929	0.503	0.9913	0.426	0.9976	0.297
ARM_IP2	1.000	0.000	0.9936	0.433	0.9946	0.498	0.9950	0.396	0.9983	0.287
ARM_IP3	1.000	0.000	0.9953	0.425	0.9933	0.535	0.9932	0.433	0.9984	0.292
Average	1.000	0.000	0.9760	1.140	0.9885	0.910	0.9476	2.131	0.9649	1.491

R^2: R-squared score vs. PrimeTime; MAE: Mean Absolute Error (ps) vs. PrimeTime.

TABLE V: Final signoff timing results from PrimeTime with pre-route SPEF files. Best results are in bold.

Circuit	Method	WNS (ns)	TNS (ns)
ARM_IP1	Elmore	-0.2814	-1640.8025
	ElmoreCeff	**-0.2781**	**-1564.8349**
ARM_IP2	Elmore	-0.2152	-78.669
	ElmoreCeff	**-0.2032**	**-76.2855**
ARM_IP3	Elmore	-0.2959	-468.9952
	ElmoreCeff	**-0.2711**	**-443.6291**

to 40.95× and 5.95× speedups over CPU- and GPU-based MOR timers, respectively, while improving final QoR by up to 8.4% in WNS and 5.4% in TNS. Future work will extend to more advanced timing models, such as nonlinear drivers, waveform distortion, signal integrity, and statistical timing analysis.

VI. Acknowledgment

This project is supported in part by National Science and Technology Major Project (2021ZD0114702), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 project (B18001).

REFERENCES

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

[2] J. Qian, S. Pullela, and L. Pillage, "Modeling the" effective capacitance" for the rc interconnect of cmos gates," *IEEE TCAD*, vol. 13, no. 12, pp. 1526–1535, 1994.

[3] T.-W. Huang and M. D. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proc. ICCAD*. IEEE, 2015, pp. 895–902.

[4] 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. 776–789, 2021.

[5] 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.

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

[7] 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.

[8] 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 *DAC*. ACM, 2019, pp. 1–6.

[9] Y. Lin, D. Z. Pan, H. Ren, and B. Khailany, "DREAMPlace 2.0: Open-Source GPU-Accelerated Global and Detailed Placement for Large-Scale VLSI Designs," in *CSTIC*. IEEE, 2020, pp. 1–4.

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

[11] S. Dhar and D. Z. Pan, "GDP: GPU accelerated detailed placement," in *Proc. HPEC*, Sept 2018.

[12] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "FastGR: Global Routing on CPU-GPU With Heterogeneous Task Graph Scheduler," *IEEE TCAD*, vol. 42, no. 7, pp. 2317–2330, 2023.

[13] 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. 01–09.

[14] 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.

[15] Z. Guo, Y. Zhang, R. Wang, Y. Lin, and H. Ren, "GEM: GPU-Accelerated Emulator-Inspired RTL Simulation," in *Proc. DAC*. IEEE, 2025, pp. 1–7.

[16] Y. Zhang, H. Ren, and B. Khailany, "Opportunities for RTL and gate level simulation using GPUs," in *Proc. ICCAD*. ACM, 2020, pp. 1–5.

[17] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with GP-GPUs," in *Proc. DAC*. ACM Press, 2009, p. 557.

[18] H. Qian and Y. Deng, "Accelerating RTL simulation with GPUs," in *Proc. ICCAD*. San Jose, CA, USA: IEEE, 2011, pp. 687–693.

[19] H. H.-W. Wang, L. Y.-Z. Lin, R. H.-M. Huang, and C. H.-P. Wen, "Casta: Cuda-accelerated static timing analysis for VLSI designs," in *Proc. ICPP*. IEEE, 2014, pp. 192–200.

[20] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. F. Young, and M. D. Wong, "GCS-Timer: Gpu-accelerated current source model based static timing analysis," in *Proc. DAC*, 2024.

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

[22] ——, "Accelerating static timing analysis using cpu-gpu heterogeneous parallelism," *IEEE TCAD*, pp. 1–1, 2023.

[23] 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.

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

[25] ——, "Gpu-accelerated critical path generation with path constraints," in *Proc. ICCAD*, 2021, pp. 1–9.

[26] G. Guo, T.-W. Huang, Y. Lin, Z. Guo, S. Yellapragada, and M. D. F. Wong, "A gpu-accelerated framework for path-based timing analysis," *IEEE TCAD*, pp. 1–1, 2023.

[27] K. Gulati and S. P. Khatri, "Accelerating statistical static timing analysis using graphics processing units," in *Proc. ASPDAC*. IEEE, 2009, pp. 260–265.

[28] Y. Shen and J. Hu, "GPU acceleration for PCA-based statistical static timing analysis," in *Proc. ICCD*. IEEE, 2015, pp. 674–679.

[29] Z. Guo, Z. Zhang, W. Li, T.-W. Huang, X. Shi, Y. Du, Y. Lin, R. Wang, and R. Huang, "HeteroExcept: A CPU-GPU heterogeneous algorithm to accelerate exception-aware static timing analysis," in *Proc. ICCAD*, 2024.

[30] Z. Xiong, R. S. Rajarathnam, and D. Z. Pan, "A data-driven, congestion-aware and open-source timing-driven fpga placer accelerated by gpus," in *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2024, pp. 115–125.

[31] Y.-K. Lin, Z. Xiong, and D. Z. Pan, "Differentiable Timing-Driven FPGA Placement with Smooth Optimization and ML-Based Delay Calibration," in *Proc. ICCAD*. IEEE, 2025, pp. 1–8.

[32] 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.

[33] W. Li, Y. Kukimoto, G. Servel, I. Bustany, and M. E. Dehkordi, "Calibration-based differentiable timing optimization in non-linear global placement," in *Proc. ISPD*. ACM, 2024, pp. 31–39.

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

[35] Arm Limited, "Arm university program," https://www.arm.com.

[36] C. V. Kashyap, C. J. Alpert, and A. Devgan, "An "effective" capacitance based delay metric for RC interconnect," in *Proc. ICCAD*, 2000, pp. 229–234.

[37] R. Puri, D. S. Kung, and A. D. Drumm, "Fast and accurate wire delay estimation for physical synthesis of large ASICs," in *Proc. GLSVLSI*, 2002, pp. 30–36.

[38] F. Dartu, N. Menezes, and L. Pileggi, "Performance computation for precharacterized CMOS gates with RC loads," *IEEE TCAD*, vol. 15, no. 5, pp. 544–553, 1996.

[39] G. Baker and P. Graves-Morris, *Padé Approximants*. Cambridge University Press, 1996.

[40] W. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[41] 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.

[42] K. Agarwal, D. Sylvester, and D. Blaauw, "Simple Metrics for Slew Rate of RC Circuits Based on Two Circuit Moments," in *DAC*. ACM, 2003, pp. 950–953.

[43] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "Closed Form Expressions for Extending Step Delay and Slew Metrics to Ramp Inputs," technical Report.

[44] R. Gupta, B. Krauter, B. Tutuianu, J. Willis, and L. T. Pileggi, "The Elmore Delay as a Bound for RC Trees with Generalized Input Signals," in *DAC*. ACM/IEEE, 1995, pp. 364–369.

[45] Z. Guo, H. Liu, X. Shi, S. Hua, Z. Zhang, C. Zhao, R. Wang, and Y. Lin, "HeteroSTA: A CPU-GPU heterogeneous static timing analysis engine with holistic industrial design support," in *Proc. ASPDAC*, 2026.

[46] "Synopsys PrimeTime," http://www.synopsys.com.

[47] T. Ajayi, D. Blaauw, T. Chan, C. Cheng, V. Chhabria, D. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça *et al.*, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," *Proc. GOMACTECH*, pp. 1105–1110, 2019.

[48] 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 *Proc. DAC*, 2019, pp. 1–4.