HeteroPower: A CPU-GPU Heterogeneous Engine to Accelerate Gate-Level Power Analysis

Xizhe Shi[1], Zizheng Guo[1,2], Zuodong Zhang[2], Yun Liang[1,2,3], Runsheng Wang[1,2,3*], Yibo Lin[1,2,3*]
[1]School of Integrated Circuits, Peking University [2]Institute of Electronic Design Automation, Peking University
[3]Beijing Advanced Innovation Center for Integrated Circuits
Email: xizheshi@stu.pku.edu.cn, {gzz, ericlyun, r.wang, yibolin}@pku.edu.cn, zhangzd@pkueda.org.cn

***Abstract*—Gate-level power analysis is critical for modern SoC signoff yet imposes a severe computational burden, especially in the vectorless mode where switching activities must be propagated across the entire design. We present HeteroPower, a CPU-GPU heterogeneous gate-level power analysis engine built upon a prior GPU-accelerated static timing analysis (STA) framework. HeteroPower reuses GPU-resident timing data, including slews, capacitances, and the levelized timing graph, to seamlessly integrate power analysis with STA, minimizing redundant data transfers. For vectorless analysis, we exploit the structural isomorphism between activity propagation and STA forward traversal, deploying a lightweight bytecode-driven probabilistic engine that maps naturally to the GPU's SIMT model. Four dedicated GPU kernels then compute switching, internal, and leakage power, featuring fused per-cell evaluation and a postfix bytecode VM for state-dependent leakage. Experiments on industrial benchmarks demonstrate that HeteroPower achieves an average speedup of 6.36× over PrimeTime PX and 10.97× over OpenSTA in the annotated mode, scaling to 21.80× over PrimeTime PX and 5.44× over OpenSTA in the vectorless mode, and maintains an average gate-level total-power mean absolute relative error (MARE) of 0.001% (annotated) and 6.76% (vectorless) against PrimeTime PX.**

I. Introduction

As process technology scales into deep sub-micron nodes, power has become a primary design constraint rivaling timing and area [1]. Gate-level power analysis is critical for signoff, verifying power budgets and guiding optimizations like clock gating [2]. However, inefficient or inaccurate power analysis leads to costly design iterations and delayed tapeouts [3].

Modern SoC designs with millions of gates impose a severe computational burden on power analysis. Specifically, physical design stages such as clock tree synthesis and ECO require repeated engine runs to guide power optimizations [4] (in Figure 1). Furthermore, in early stages without simulation vectors, vectorless analysis requires global activity propagation, linearly scaling with design size [5]. Therefore, accelerating power analysis is imperative to shorten design turnaround time.

Fundamentally, gate-level power analysis is tightly coupled with STA [6]. Proper switching power calculation relies on parasitic capacitance and voltage data from timers, while internal power lookups depend on input slews from timing updates [7]. Further, vectorless activity propagation shares the same graph traversal pattern as STA. Industry tools like PrimeTime PX [8] validate this synergy by embedding power analysis within the STA engines. This inherent coupling suggests that power analysis can naturally leverage the framework of CPU-GPU heterogeneous accelerated STA.

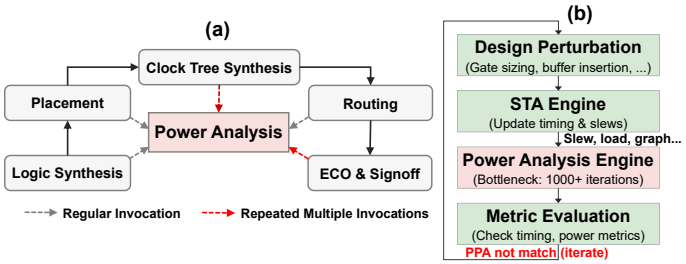

Fig. 1: Overview of power analysis in modern EDA flow. (a) Power analysis across physical design stages. (b) The computational bottleneck in iterative optimizations.

However, existing power analysis solutions have not leveraged the opportunity. Both PrimeTime PX and OpenSTA [9] remain CPU-bound, suffering from long runtimes on large designs. Recent academic research has explored GPU acceleration for specific sub-tasks: GATSPI [3] accelerates gate-level simulation to generate switching activity data, GRANNITE [4] employs graph neural networks to predict toggle rates, and SyfriPow [10] uses GPU-accelerated sparse polynomial inference for vectorless activity estimation. However, these efforts focus exclusively on deriving switching activity. No prior work delivers a complete power analysis flow on a CPU-GPU heterogeneous platform, covering vectorless activity propagation and switching, internal, and leakage power calculations.

To bridge the gap, we propose HeteroPower, a CPU-GPU heterogeneous gate-level power analysis engine built upon prior GPU-accelerated STA infrastructure [11]. Our main contributions are as follows:

1) We propose HeteroPower, the first complete CPU-GPU heterogeneous gate-level power analysis engine that reuses GPU-resident STA data and covers vectorless activity propagation together with switching, internal, and leakage power calculation on a unified platform.
2) We design a GPU-parallel vectorless activity propagation engine that exploits the structural isomorphism with STA forward traversal, employing a compact bytecode VM to evaluate cell Boolean functions in the probabilistic

* Corresponding authors.
This project is supported in part by the National Science Foundation of China (Grant No. T2293701), the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

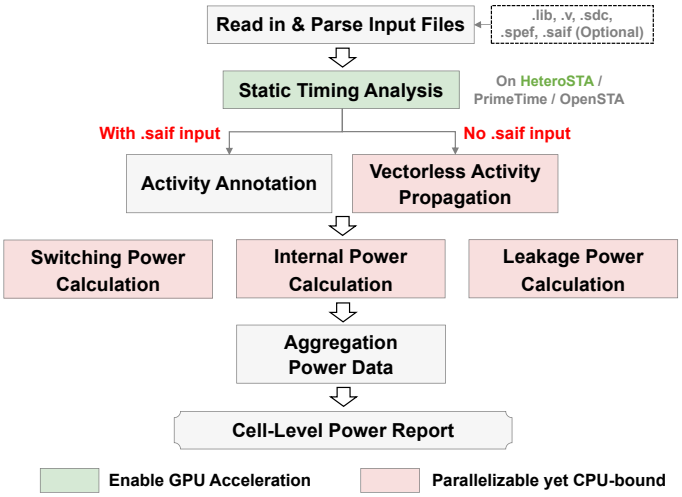

Fig. 2: Overview of gate-level averaged power analysis flow.

domain with near-zero additional framework cost.

3) We develop four fused GPU power kernels with tailored parallelism: per-cell switching and leakage kernels, and a two-phase internal-power evaluation with per-arc LUT interpolation followed by per-cell aggregation, incorporating toggle-rate-based arc normalization and a postfix bytecode VM for state-dependent leakage evaluation.

Experiments on industrial-scale benchmarks show that HeteroPower achieves average speedups of 6.36× over PrimeTime PX and 10.97× over OpenSTA in annotated mode, and 21.80× and 5.44×, respectively, in vectorless mode, while maintaining an average gate-level total-power MARE of 0.001% of PrimeTime PX with SAIF annotation and 6.76% in vectorless mode.

II. Preliminaries

A. Gate-Level Power Analysis Flow

Averaged power analysis estimates total design power using netlists, liberties, parasitics, constraints and optional switching activity interchange format (SAIF) files. As shown in Figure 2, the flow heavily reuses STA data such as the timing graph, slews, and load capacitance. Switching activities are then annotated from SAIF files or derived via vectorless propagation. Based on these data, the engine calculates three types of power, finally aggregating them into a gate-level power data report.

GPU-accelerated STA engines [11] efficiently map levelized timing graphs to GPUs. Subsequent vectorless propagation and power calculation steps share this parallelism but remain CPU-bound, presenting a clear opportunity for GPU acceleration.

B. Switching Activity and Propagation

Power calculation requires two per-pin quantities: the *toggle rate* (TR), measuring transition frequency in Hz, and the *static probability* (SP), the fraction of time a signal stays at logic one. When available, these are annotated from a SAIF file produced by gate-level simulation, where $TR = TC/Duration$.

When simulation data are unavailable, activities are derived via vectorless propagation. Seed activities are first assigned to primary inputs (TR=$0.1/T_{clk}$, SP=0.5), clock pins (TR=$2/T_{clk}$, SP=$duty$), and constants (TR=0). Activities then propagate forward along the levelized timing graph. For a combinational cell with Boolean function $Y = f(x_1, \ldots, x_n)$, the output toggle rate is:

$$TR_Y = \sum_{i=1}^{n} TR_{x_i} \cdot P\left(\frac{\partial f}{\partial x_i}\right)$$

where $P(\partial f/\partial x_i)$ is the Boolean difference probability, representing the likelihood that a transition on x_i sensitizes Y.

For sequential elements (D-flip-flops), the output Q acts as a filter bounded by both data (D) and clock (CLK) activities:

$$TR_Q = \min\left(TR_D, 2 \cdot SP_D \cdot (1 - SP_D) \cdot TR_{CLK}\right)$$

The propagation proceeds via BFS over the levelized graph, iterating across sequential boundaries until convergence in the general case. In practice, users frequently fix register boundary activities via `set_switching_activity` or partial SAIF annotation; under this setting all register Q pins and primary inputs serve as known seeds, and the propagation reduces to a single forward pass through combinational logic. In either case, the levelized traversal mirrors STA forward propagation, enabling direct reuse of GPU-accelerated graph framework.

C. Power Components and Calculation

The total gate-level power dissipation sums switching, internal, and leakage power components. Their evaluation requires STA-derived electrical metrics and switching activities.

Switching power represents the dynamic power of charging and discharging a cell's external load. Evaluated exclusively at output pins of cells, it depends on the Liberty supply voltage (V_{dd}), total load capacitance (C_{load}) combining SPEF and downstream pin capacitances, and the net toggle rate (TR):

$$P_{sw} = 0.5 \cdot C_{load} \cdot V_{dd}^2 \cdot TR$$

Internal power accounts for dynamic energy consumed in the cell, from short-circuit currents and internal node switching. Evaluated per timing arc, transition energy (E) is queried from Liberty tables using input slew and output load. To handle logical masking, E is scaled by a sensitization weight derived from Boolean difference, and multiplied by the active state probability ($Duty_{when}$) for state-dependent arcs:

$$P_{int} = \sum_{arc \in cell} E(slew, C_{load}) \cdot weight \cdot Duty_{when} \cdot TR$$

Leakage power is the static power dissipated by a cell independent of switching. Because it is highly state-dependent, Liberty specifies leakage under different inputs via `when` attributes. The cell's total leakage weights each state-specific value (P_{state}) by occupancy probability ($Duty_{state}$). A default leakage ($P_{default}$) covers any remaining undefined states:

$$P_{leak} = \sum_{state} P_{state} \cdot Duty_{state} + P_{default} \cdot (1 - \sum_{state} Duty_{state})$$

III. ALGORITHMS

To enable CPU-GPU heterogeneous gate-level power analysis integrated with STA, we propose **HeteroPower**, a heterogeneous power analysis engine that pipelines directly after HeteroSTA. In Figure 2, the parallelizable yet CPU-bound modules highlighted in the flow are accelerated on the GPU in HeteroPower.

Specifically, HeteroPower reuses GPU-resident timing data from HeteroSTA (Subsection III-A), derives switching activity via SAIF annotation or GPU-accelerated vectorless propagation (Subsection III-B), and launches GPU kernels to compute switching, internal, and leakage power (Subsection III-C).

A. Integration with HeteroSTA and Data Reuse

The power kernels in HeteroPower are built upon the GPU-resident framework of HeteroSTA [11]. Timing data computed in STA, including pin transition times and net load capacitance, are retained on the GPU and passed directly to power kernels, eliminating redundant host-device transfers. For brevity, timing graph topology data shared with HeteroSTA (levelization, pin-to-net mappings, etc.) are omitted from the algorithm inputs in the following, as they are assumed GPU-resident throughout.

Beyond timing data reuse, HeteroPower performs a one-time *power graph construction* phase that augments the STA timing graph with power-specific data. This phase indexes each cell type's Liberty power tables (both `internal_power` and `leakage_power` groups) into GPU-friendly flattened arrays, pre-compiles `when`-condition Boolean expressions into postfix bytecode programs, and allocates per-pin activity arrays for SP and TR. Most constructed data are transferred to GPU memory once and remain resident, ensuring the core analysis proceeds with minimal host-device synchronization.

B. GPU-Accelerated Vectorless Activity Propagation

When SAIF data are unavailable or only partially cover the design, HeteroPower derives per-pin switching activity through GPU-accelerated vectorless propagation. The key observation is that this propagation is isomorphic to STA forward arrival-time analysis: both traverse a levelized timing graph from seeds to endpoints, evaluating a per-cell transfer function at each level. By reusing the same levelization, topology arrays, and GPU dispatch infrastructure already built for HeteroSTA, the activity engine incurs near-zero additional framework cost, with only the per-cell evaluation logic changed.

Following the standard industrial workflow, register outputs and primary inputs carry known activities (via SAIF annotation or `set_switching_activity`), serving as fixed seeds at sequential boundaries. The propagation then proceeds forward through combinational logic, as summarized in Algorithm 1. Each level is dispatched as a GPU kernel with one thread per pin: input pins copy activities from their net drivers, while combinational output pins evaluate a pre-compiled bytecode program that maps Boolean operators to their probabilistic counterparts (e.g., AND: $SP \leftarrow SP_a \cdot SP_b, TR \leftarrow TR_a \cdot SP_b + TR_b \cdot SP_a$) under the signal-independence assumption. The bytecodes and pin-to-slot mappings are compiled once during context construction, mirroring the leakage `when`-expression VM already present in HeteroPower. Pins already carrying SAIF annotations are never overwritten, allowing vectorless propagation to fill only the unannotated portion of the design.

Algorithm 1: GPU-Accelerated Activity Propagation

Input : Seed Activities SP, TR (annotated pins), Cell Boolean Function Bytecodes, Pin Slews $slew$ (for density capping)
Output: Per-pin SP and TR for combinational pins

for *each level* ℓ *in forward topological order* **do**
  Call Activity_Prop_Kernel on all pins in level ℓ
Kernel Function Activity_Prop_Kernel:
  $pin_{id} \leftarrow$ levels_nd$[blockIdx.x \times blockDim.x + threadIdx.x]$
  if pin is annotated **then return**
  if $dir[pin_{id}] =$ INPUT **then**
    Copy (SP, TR) from net driver pin
  else if $dir[pin_{id}] =$ OUTPUT and not sequential **then**
    Evaluate bytecode: {AND, OR, NOT, XOR} $\rightarrow$ (SP, TR)
    Cap TR by $1/slew[pin_{id}]$ if slew is available

This design reflects a deliberate trade-off. OpenSTA computes $P(\partial f/\partial x_i)$ via BDD (CUDD), which captures multi-input correlations exactly but resists GPU parallelization due to pointer-chasing BDD traversals. PrimeTime PX employs zero-delay Monte Carlo simulation, propagating random bitvectors through the netlist for higher fidelity at the cost of greater memory and compute. HeteroPower instead adopts lightweight closed-form probability formulas that map naturally to the GPU's SIMT execution model. The resulting independence-assumption error is small for standard cells and is further bounded by a slew-based density cap, preventing unrealizable toggle rates from accumulating along deep logic cones.

C. GPU-Accelerated Three-Component Power Calculation

Once the activity propagation converges, the per-pin TR and SP are fully resident in GPU memory. Because HeteroPower shares the underlying infrastructure of HeteroSTA, the timing graph, parasitics, and transition slews are already available.

Algorithm 2 assigns one GPU thread per cell to accumulate **switching power** across its output pins. Since Liberty characterizes rise and fall capacitance separately, C_{eff} is conservatively approximated as $\max(C_{rise},\ C_{fall})$.

The evaluation of **internal power** for each cell proceeds in two consecutive GPU kernel phases, outlined in Algorithm 3. Phase 1 assigns one thread per power arc for LUT2D interpolation. Each thread queries `rise_power` and `fall_power` tables from Liberty, indexed by output load capacitance and an averaged input slew (sl). This averaging provides a lightweight approximation of unate-aware selection without expensive per-arc checks. The interpolations yield the base internal transition energies (E_{rise} and E_{fall}) of each power arc. All instantiated power arcs are evaluated concurrently with the data from STA.

Algorithm 2: Fused-Switching-Power-Calculation

Input : Supply Voltage V_{dd}, Pin Directions dir, Static Probability SP, Toggle Rates TR, Load Capacitance C_{rise}, C_{fall}
Output: Cell Switching Power P_{sw}

Call Switching_Power_Kernel with N_{cells}
Kernel Function Switching_Power_Kernel:
  $cell_{id} \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
  if $cell_{id} \geq N_{cells}$ **then return**
  for *each pin* p *in* $cells[cell_{id}]$ **do**
    if $dir[p] \neq$ OUTPUT **then continue**
    $n \leftarrow pin2net[p]$
    $C_{eff} \leftarrow \max(C_{rise}[n],\ C_{fall}[n])$
    $P_{sw}[cell_{id}] \leftarrow P_{sw}[cell_{id}] + 0.5 \cdot V_{dd}^2 \cdot C_{eff} \cdot TR[p]$

Algorithm 3: Fused-Internal-Power-Calculation

Input : Static Probability SP, Toggle Rates TR, Input Slew $slew_{rise}, slew_{fall}$, Load Capacitance C_{rise}, C_{fall}, Liberty Power Tables $\mathbf{E}_{rise}, \mathbf{E}_{fall}$
Output: Cell Internal Power P_{int}

▷ Phase 1: Arc-level LUT interpolation
Call Internal_Arc_Kernel with N_{arcs} threads
Kernel Function Internal_Arc_Kernel:
  $arc_{id} \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
  if $arc_{id} \geq N_{arcs}$ **then return**
  $sl \leftarrow (slew_{rise}[from] + slew_{fall}[from])/2$
  $C_{eff} \leftarrow \max(C_{rise}[to],\ C_{fall}[to])$
  $E_{rise}[arc_{id}] \leftarrow \text{LUT2D}(\mathbf{E}_{rise}[arc_{id}],\ sl,\ C_{eff})$
  $E_{fall}[arc_{id}] \leftarrow \text{LUT2D}(\mathbf{E}_{fall}[arc_{id}],\ sl,\ C_{eff})$
▷ Phase 2: Cell-level weighted aggregation
Call Internal_Cell_Kernel with N_{cells} threads
Kernel Function Internal_Cell_Kernel:
  $cell_{id} \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
  if $cell_{id} \geq N_{cells}$ **then return**
  for *each output pin* p *in* $cells[cell_{id}]$ **do**
    for *each* p*'s fanin arc* $k(from, p)$ **do**
      if $when_k \neq \emptyset$ **then**
        $d_k \leftarrow \text{EvalWhen}(when_k,\ SP)$
        $E \leftarrow d_k \cdot (E_{rise}[k] + E_{fall}[k])$
        $P_{int}[cell_{id}] \leftarrow P_{int}[cell_{id}] + E \cdot (TR[p]/2)$
      else
        $effTR_k \leftarrow TR[from] \cdot P(\partial f/\partial x_{from})$
        $W \leftarrow W + effTR_k$
    for *each default fanin arc* $k(from, p)$ *with* $when_k = \emptyset$ **do**
      $w_k \leftarrow effTR_k / W$
      $E \leftarrow w_k \cdot (E_{rise}[k] + E_{fall}[k])$
      $P_{int}[cell_{id}] \leftarrow P_{int}[cell_{id}] + E \cdot (TR[p]/2)$

Phase 2 launches one thread per cell to aggregate these arc energies. For arcs with explicit `when` conditions, contributions are weighted directly by the evaluated state probability. For default arcs without `when` conditions, effective related-pin toggle rates are first computed using Boolean-difference sensitization and then normalized to avoid double-counting across multiple related pins. The aggregated energies are then summed and scaled by $TR[p]/2$ to yield the final pin power. For brevity, the input-pin LUT1D evaluation of power tables (e.g., clock pins of flip-flops) is omitted from Algorithm 3; it is handled analogously within the same kernel.

Algorithm 4 computes cell **leakage power** with one thread per cell. Each liberty leakage block associates a leakage value with a `when` state condition; the key challenge is evaluating these Boolean expressions efficiently on the GPU. EvalWhen solves this by running a postfix-bytecode VM that replaces Boolean operators with their probabilistic counterparts under the signal independence assumption, converting each `when` clause into a duty factor d_j representing the probability that the cell resides in that state. Conditional block contributions are weighted by d_j, and the residual probability $\max(0, 1-D_{sum})$ is assigned to the default leakage. All intermediate values are accumulated in thread-local registers, yielding a fully parallel kernel with no synchronization overhead.

IV. Experimental Results

A. Experimental Setup

We developed HeteroPower as a heterogeneous gate-level power analysis engine built upon HeteroSTA, implemented in C++, CUDA, and Rust. HeteroPower provides a flexible user interface compatible with standard EDA input formats including `.lib`, `.v`, `.sdc`, `.spef`, and optional `.saif`.

To evaluate the runtime benefits of HeteroPower, we compare against two representative baselines: (1) Synopsys PrimeTime PX (2021.06), the industry-standard commercial power analysis tool; and (2) OpenSTA, the state-of-the-art open-source STA engine with power analysis support. Runtime is measured by the wall-clock time of the `report_power` command after loading design files. Each reported value is the average of three independent runs. We strictly exclude the execution time of STA to isolate the performance benefits of our power analysis engine for fair comparison.

Our experimental environment is a CentOS 7.9 Linux server with a 64-core Intel Xeon Platinum 8358 CPU and 8 NVIDIA A800 GPUs. The server is configured with 1 TB of system memory, and each GPU provides 80 GB of dedicated memory. Table I details the statistics of the six evaluated benchmarks synthesized using the Nangate 45nm Open Cell Library.

We evaluate our framework under two input configurations. In the *annotated* mode, a `.saif` file from gate-level simulation is provided to supply accurate switching activities. In the *vectorless* mode, switching activities are estimated entirely via GPU-accelerated vectorless propagation; this configuration is used to assess the accuracy and efficiency of our scheme against PrimeTime PX vectorless flow. For fair comparison, all designs are evaluated using identical netlists, Liberty, parasitic, SDC, and, where applicable, SAIF files. Power results are validated against PrimeTime PX as the golden reference across

Algorithm 4: Leakage-Power-Calculation

Input : Static Probability SP, Default Leakage P_{def}, Leakage Blocks $\{(value_j,\ when_j)\}$, Bytecode VM opcodes & operands
Output: Cell Leakage Power P_{leak}

Call Leakage_Cell_Kernel with N_{cells} threads
Kernel Function Leakage_Cell_Kernel:
  $cell_{id} \leftarrow blockIdx.x \times blockDim.x + threadIdx.x$
  if $cell_{id} \geq N_{cells}$ **then return**
  for *each leakage block* j *of* $cell_{id}$*'s type* **do**
    if $when_j \neq \emptyset$ **then**
      $d_j \leftarrow \text{EvalWhen}(when_j,\ SP)$
      $P_{cond} \leftarrow P_{cond} + value_j \cdot d_j$
      $D_{sum} \leftarrow D_{sum} + d_j$, $has\_cond \leftarrow$ **true**
    else
      $P_{uncond} \leftarrow P_{uncond} + value_j$
      $has\_uncond \leftarrow$ **true**
  if has_cond **then**
    $P_{leak}[cell_{id}] \leftarrow P_{cond} + P_{def} \cdot \max(0,\ 1 - D_{sum})$
  else if has_uncond **then** $P_{leak}[cell_{id}] \leftarrow P_{uncond}$
  else $P_{leak}[cell_{id}] \leftarrow P_{def}$

both configurations.

B. Accuracy Analysis

We first verify the accuracy of HeteroPower against the commercial golden standard, PrimeTime PX. Table I reports the MARE of switching (P_{sw}), internal (P_{int}), leakage (P_{leak}), and total power (P_{tot}) across all cells in the design.

In the SAIF-annotated mode, HeteroPower achieves near-perfect alignment with PrimeTime PX. The average MARE for total power is merely **0.001%**, with internal power error strictly constrained to **0.024%**. This validates the exactness of our GPU-accelerated power calculation algorithms. While OpenSTA exhibits noticeable component-level deviations (averaging **63.51%** in internal power), HeteroPower accurately resolves state-dependent power arcs and input slews, yielding consistent fine-grained power estimation across benchmarks.

In the vectorless mode, accurate power analysis is challenging due to complex reconvergent fanouts and spatial signal correlations. Despite adopting a lightweight propagation model tailored for GPU parallelism, HeteroPower maintains an excellent total power MARE of **6.759%** on average against PrimeTime PX. This also remains lower than that of OpenSTA (**7.910%**), demonstrating that substantial acceleration is achieved in our framework with limited accuracy loss.

C. Runtime Performance

To show the acceleration of our CPU-GPU heterogeneous architecture, we compare core power evaluation runtimes under both modes. As shown in Table II, HeteroPower delivers exceptional speedups over both CPU-bound baselines.

In the SAIF-annotated mode, workload is dominated by massive independent Liberty table lookups and capacitance evaluations. HeteroPower finishes the largest design (`case6`) in just 1.829 seconds, achieving an average speedup of **6.36**× over PrimeTime PX and **10.97**× over OpenSTA.

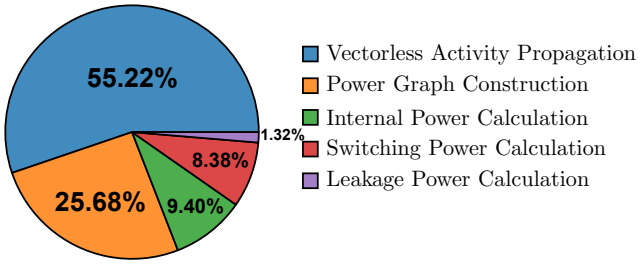

Fig. 3: Runtime breakdown of HeteroPower evaluating **case6** in vectorless mode.

HeteroPower exhibits an even larger advantage in vectorless mode. While PrimeTime PX suffers from the heavy computational burden of Monte Carlo zero-delay simulation, our lightweight parallel propagation engine excels. HeteroPower achieves average speedups of **21.80**× against PrimeTime PX (peaking at **31.09**× on `case4`) and **5.44**× against OpenSTA. These results firmly establish the scalability and efficiency of mapping power analysis tasks to a heterogeneous platform.

D. Runtime Breakdown

To deeply understand the computational bottlenecks and the efficiency of our heterogeneous engine, we analyze the runtime breakdown of HeteroPower under the vectorless mode.

As illustrated in Figure 3, we decompose the total execution time into five constituent stages. *Vectorless Activity Propagation* dominates, accounting for 55.22% of the total execution. This is expected as global propagation carries inherent topological dependencies that require synchronized level-by-level traversal. *Power Graph Construction*, which augments the STA timing graph with power data structures and initializes the GPU memory context, consumes an additional 25.68%.

Notably, the four power evaluation kernels together account for under 20% of the runtime, confirming that the per-cell power calculations are highly efficient once activity data are available. The dominant cost lies in vectorless propagation, which, despite full GPU acceleration, is inherently constrained by level-by-level topological dependencies.

V. Conclusion

This paper presents HeteroPower, a CPU-GPU heterogeneous gate-level power analysis engine that tightly integrates with the GPU-accelerated STA infrastructure of HeteroSTA. By reusing GPU-resident timing data and exploiting the structural isomorphism between vectorless activity propagation and STA forward traversal, HeteroPower delivers a complete power analysis flow, covering switching, internal, and leakage power, mainly on the GPU. Experiments on six industrial-scale benchmarks show that HeteroPower achieves average speedups of 6.36× over PrimeTime PX and 10.97× over OpenSTA in annotated mode, and 21.80× and 5.44× in vectorless mode, while maintaining a gate-level total-power MARE within 0.001% (annotated) and 6.76% (vectorless) of the commercial golden reference. These results demonstrate

TABLE I: Accuracy comparison (MARE vs. PrimeTime PX) of power analysis between HeteroPower and OpenSTA.

Benchmark	Circuit Statistics			HeteroPower Error (%)					OpenSTA Error (%)				
				SAIF Annotation				Vec.	SAIF Annotation				Vec.
	#Cells	#Nets	#Pins	P_{sw}	P_{int}	P_{leak}	P_{tot}	P_{tot}	P_{sw}	P_{int}	P_{leak}	P_{tot}	P_{tot}
case1	131K	133K	383K	0.001	0.061	0.001	0.002	9.521	2.467	25.74	0.002	0.027	9.941
case2	165K	165K	474K	0.001	0.013	0.001	0.001	1.195	13.40	82.98	0.002	0.075	2.450
case3	219K	219K	704K	0.001	0.016	0.001	0.001	9.210	6.732	23.59	0.231	0.247	9.267
case4	649K	649K	2.07M	0.001	0.021	0.001	0.001	8.993	12.11	23.59	0.001	0.003	10.12
case5	794K	795K	2.53M	0.001	0.022	0.001	0.001	8.470	28.01	148.6	0.020	0.128	11.36
case6	959K	961K	2.83M	0.001	0.012	0.001	0.001	3.164	11.78	76.57	0.004	0.071	4.324
Average	—	—	—	**0.001**	**0.024**	**0.001**	**0.001**	**6.759**	**12.42**	**63.51**	**0.043**	**0.092**	**7.910**

SAIF Annotation: Power analysis with toggle rates and static probabilities annotated from SAIF files.
Vec. (Vectorless): Power analysis using vectorless activity propagation (without SAIF).
MARE: Mean Absolute Relative Error of gate-level power on the design, using PrimeTime PX as the golden reference.

TABLE II: Runtime comparison (in seconds) between HeteroPower, PrimeTime PX, and OpenSTA under different modes.

Benchmark	HeteroPower		PrimeTime PX				OpenSTA			
	SAIF	Vec.	SAIF		Vec.		SAIF		Vec.	
	RT (s)	RT (s)	RT (s)	Speedup	RT (s)	Speedup	RT (s)	Speedup	RT (s)	Speedup
case1	0.290	0.631	1.438	4.96×	10.81	17.13×	1.804	6.22×	3.463	5.49×
case2	0.293	0.747	1.812	6.18×	16.06	21.50×	2.855	9.74×	3.605	4.83×
case3	0.382	1.143	2.493	6.53×	13.23	11.57×	4.547	11.90×	5.545	4.85×
case4	1.330	3.229	8.413	6.33×	100.4	31.09×	18.56	13.95×	23.47	7.27×
case5	1.717	5.139	11.77	6.85×	136.1	26.48×	24.33	14.17×	26.27	5.11×
case6	1.829	4.379	13.36	7.30×	100.8	23.02×	18.04	9.86×	22.28	5.09×
Average	—	—	—	**6.36**×	—	**21.80**×	—	**10.97**×	—	**5.44**×

RT: Runtime in seconds. **Speedup**: Runtime ratio compared to HeteroPower.
HeteroPower is executed with 16 CPU cores and 1 GPU. PrimeTime PX and OpenSTA are evaluated using 16 CPU cores.

that mapping the full power analysis flow to a heterogeneous platform is both practical and highly effective, establishing a foundation for further acceleration of signoff-quality power estimation in million-gate SoC designs. As the current vectorless propagation adopts the signal-independence assumption, future work will explore correlation-aware models to further narrow the accuracy gap. We also plan to extend HeteroPower with multi-corner support and validate its scalability on larger industrial designs through hierarchical partitioning of the GPU memory space.

REFERENCES

[1] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2002.

[2] F. N. Najm, "A survey of power estimation techniques in vlsi circuits," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 2, no. 4, pp. 446–455, 2002.

[3] Y. Zhang, H. Ren, A. Sridharan, and B. Khailany, "Gatspi: Gpu accelerated gate-level simulation for power improvement," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1231–1236.

[4] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[5] F. N. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 310–323, 2002.

[6] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," 1992.

[7] S. Gupta and F. N. Najm, "Power modeling for high-level power estimation," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 8, no. 1, pp. 18–29,

2000.

[8] Synopsys, *PrimeTime, version: S-2021.06-SP1*, Synopsys, Inc., Mountain View, California, USA, 2021, synopsys, Inc. Software. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html

[9] "OpenSTA, version: 2.5.0," https://github.com/abk-openroad/OpenSTA.

[10] Z. Lyu and J. Shen, "Symbolic-functional representation inference for gate-level power estimation," *Microelectronics Journal*, vol. 154, p. 106443, 2024.

[11] 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," *arXiv preprint arXiv:2511.11660*, 2025.