Accelerating Gate Sizing using GPU

Yi-Hua Chung[1], Nahmsuk Oh[2], Malleswara Gupta Balabhadra Naga Venkata[2], Aditya Shiledar[2], Sudipto Kundu[2], Vishal Khandelwal[2], and Tsung-Wei Huang[1]

[1] University of Wisconsin-Madison, Madison, WI, USA
[2] Synopsys Inc, Sunnyvale, CA, USA

EURO-PAR CONFERENCE 2025

Department of Electrical and Computer Engineering
UNIVERSITY OF WISCONSIN-MADISON

SYNOPSYS®

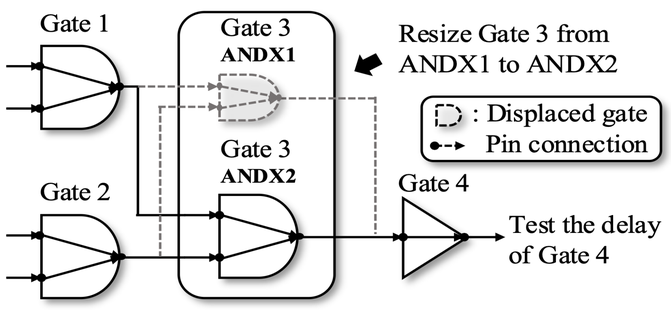
Problem Definition

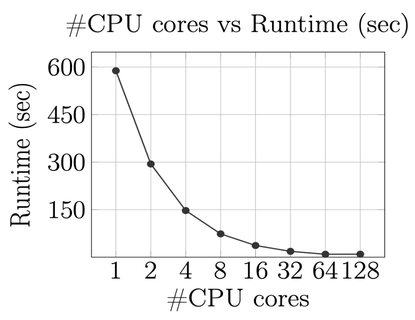
Background

- **Gate sizing** is a fundamental optimization step in VLSI design that directly impacts circuit performance, power consumption, and area.
- **Goal:** Select the best standard cell size for each gate to achieve optimal trade-offs between timing, power, and area constraints.
- Proper gate sizing helps ensures critical paths meet setup/hold constraints while minimizing power

Problem Definition

- **Input:** A set of independent standard cells with input/output pins.
- **Goal:** Select the optimal library cell (libcell) for each gate to minimize timing violations and design constraints.
- **Example:** Resize Gate 3 From size X1 (cell ANDX1) to size X2 (cell ANDX2) to minimize the overall timing delay.

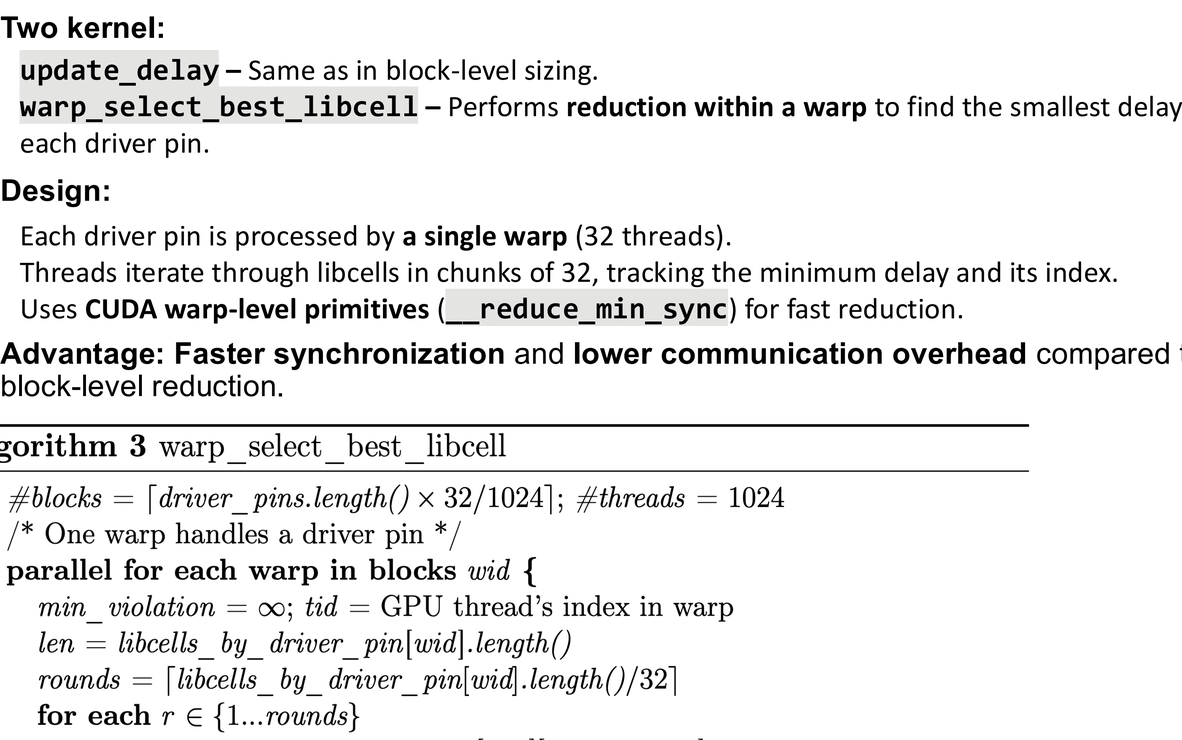




Motivation and Overview

Motivation

- **Industrial sizers** use CPU parallelism to accelerate multiple steps in gate sizing.
- **Libcell selection** process has been widely studied, as it exhibits a high degree of parallelism.
- **Limitation:** CPU scalability plateaus at ~16–64 threads.
- **Opportunity:** Modern GPUs offer thousands of threads, enabling far greater parallelism, which motivates us to leverage GPU to accelerate the libcell selection process in gate sizing.



Algorithm

We introduce two GPU kernels to accelerate the libcell selection in gate sizing: (1) block-level sizing and (2) warp-level sizing, both designed for diverse GPU architectures and maximum runtime efficiency.

Block-level Sizing

- **Block-level sizing** targets GPUs without advanced warp-level primitives, focusing on distributing the workload at the thread-block level.
- **Two kernel:**
  1. `update_delay` – Computes delay for all libcells of each driver pin in parallel.
  2. `select_best_libcell` – Selects the libcell with the smallest delay for each driver pin.
- **Data layout:** Store libcells in a 1D array for coalesced memory access.
- Driver pin processed **independently**; results stored in `best_libcell` array.

Algorithm 1 update_delay

```
1: /* One thread handles one libcell of a driver pin */
2: #blocks = ⌈libcells_of_driver_pins.length() / 1024⌉; #threads = 1024
3: parallel for each thread tid {
4:     libcell = libcells_of_driver_pins[tid]
5:     delay[tid] = compute_delay(driver_pins, libcell)
6: }
```

Algorithm 1 `update_delay`: Computes delays for all libcells of each driver pin in parallel, assigning **one thread per libcell**. Stores libcells in a 1D array for **coalesced memory access** and writes computed delays to the `delay` array.

Algorithm 2 select_best_libcell

```
1: /* One thread performs a reduction of all delays of a driver pin */
2: #blocks = ⌈driver_pins.length() / 1024⌉; #threads = 1024
3: parallel for each thread tid {
4:     idx = argmin_i{delay[i] | i ∈ driver_pins[tid]}
5:     best_libcell[tid] = the libcell at index idx within driver_pins[tid]
6: }
```

Algorithm 2 `select_best_libcell`: For each driver pin, a single thread performs a **reduction** to find the libcell with the **smallest delay** from the `delay` array and stores the result in the `best_libcell` array.

Warp-level Sizing

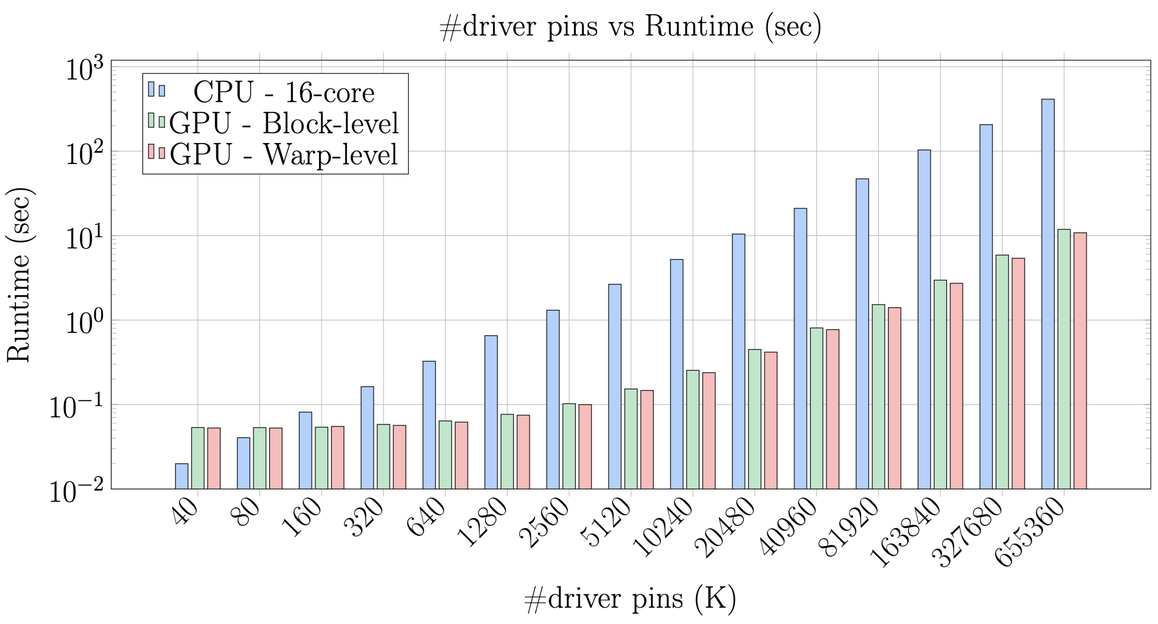
- **Warp-level sizing** targets GPUs that support **warp-level primitives** to further achieve fine-grained synchronization and efficient communication during the libcell selection process.
- **Two kernel:**
  1. `update_delay` – Same as in block-level sizing.
  2. `warp_select_best_libcell` – Performs **reduction within a warp** to find the smallest delay for each driver pin.
- **Design:**
  - Each driver pin is processed by **a single warp** (32 threads).
  - Threads iterate through libcells in chunks of 32, tracking the minimum delay and its index.
  - Uses **CUDA warp-level primitives** (`__reduce_min_sync`) for fast reduction.
- **Advantage: Faster synchronization** and **lower communication overhead** compared to block-level reduction.

Algorithm 3 warp_select_best_libcell

```
1: #blocks = ⌈driver_pins.length() × 32/1024⌉; #threads = 1024
2: /* One warp handles a driver pin */
3: parallel for each warp in blocks wid {
4:     min_violation = ∞; tid = GPU thread's index in warp
5:     len = libcells_by_driver_pin[wid].length()
6:     rounds = ⌈libcells_by_driver_pin[wid].length()/32⌉
7:     for each r ∈ {1...rounds}
8:         libcell = libcells_by_driver_pin[wid][r × 32 + tid]
9:         tmp ← time violation of libcell (r × 32 + tid) by driver pin
10:        min_id = (min_violation > tmp) ? (r × 32 + tid) : (min_id)
11:        min_violation = min(min_violation, tmp)
12:    w_min = __reduce_min_sync(0xFFFFFFFF, min_violation)
13:    min_index = (min_violation == w_min) ? min_id : len
14:    w_min_index = __reduce_min_sync(0xFFFFFFFF, min_index)
15:    best_libcell[wid] = libcells_by_driver_pin[wid][w_min_index]
16: }
```

Algorithm 3 `warp_select_best_libcell`: Each driver pin is processed by **one warp**. Threads iterate through its libcells, track the **smallest delay** and index, and use **warp-level reduction** to select and store the optimal libcell.

Experimental Results



Conclusions

- We introduced a GPU-accelerated algorithm to speed up the libcell selection routine in gate sizing, a critical step in VLSI design optimization.
- We proposed algorithm addresses the scalability bottlenecks of multi-core CPUs by leveraging both block-level and warp-level GPU parallelism to independently evaluate and select the optimal libcell for each gate.
- Compared to a 16-core CPU baseline, our approach achieves up to **38×** speedup, with warp-level sizing further improving performance by an additional **4.77%** on average.

Acknowledgement

- This project is supported by NSF grants 2235276, 2349144, 2349143, 2349582, and 2349141.
- Work done during internship at Synopsys Inc.

Personal Blog