ExactMap: Enhancing Delay Optimization in Parallel ASIC Technology Mapping

Zhenxuan Xie Lixin Liu Tianji Liu Evangeline F.Y. Young
Department of Computer Science & Engineering, The Chinese University of Hong Kong
{zxxie24, lxliu, tjliu, fyyoung}@cse.cuhk.edu.hk

***Abstract*—ASIC technology mapping consists of mapping a technology-independent Boolean network into an equivalent circuit utilizing cells from a specified library, a process that is vital in electronic design automation (EDA). However, existing algorithms in the literature are sequential in nature and often neglect to account for the actual delay of the cells during the mapping process, resulting in significant discrepancy between estimated and actual delay. In this paper, we propose an ASIC technology mapper that considers load information of intermediate solutions. This approach enables the identification of critical nodes and a better selection of delay-oriented cells for those nodes. Furthermore, we introduce a dual recovery method for delay and area to enhance performance. Finally, these innovations are integrated within a configuration-level parallelism framework on GPU. Experimental results on different technology libraries demonstrate that our method achieves on average 32% reduction in delay with 3% area penalty compared to the public synthesis tool ABC. Additionally, our approach provides a significant speedup of 65.33×.**

I. Introduction

Technology mapping is a critical step in logic synthesis, involving the transformation of a circuit from a technology-independent format (e.g., And-Inverter Graph, AIG), into a technology-dependent representation utilizing standard cells or lookup-tables (LUT). For ASIC technology mapping, this process is essential for optimizing key performance metrics, including area, delay, and power [1]–[4]. As the complexity of circuits continues to increase, the outcomes of this phase can significantly impact subsequent physical design processes. Therefore, developing an efficient and effective ASIC technology mapping algorithm has become increasingly important.

The area minimization problem in technology mapping is known to be NP-complete, while delay minimization is considered even more intricate [5], [6]. To reduce the scale of the problem, early algorithms primarily employed tree-based approaches [7]. However, with the increasing demand for quality, several cut-based methods have been proposed [1], [8] that provide a more comprehensive perspective of the entire circuit, enabling the achievement of higher-quality mappings.

Recently, various cut-based methods have been developed to enhance the quality of ASIC technology mapping. SLAP [9] presented a supervised learning approach for prioritizing cuts and reducing the number of cuts so that the search space is changed and reduced. LEAP [10] proposed a machine learning-assisted method to further reduce used cuts. The work proposed by [11] utilized machine learning techniques to predict the delay of the gates. Liu et al. [12] proposed a heuristic method to estimate input transitions and output loads according to the structure around the nodes, based on which the gate delay can be estimated. MapTune [13] introduced an approach for reducing the technology library with reinforcement learning, resulting in a reduced search space and potentially improved mapping quality. However, gate delay is influenced by the circuit mapping [12], but all these methods do not consider the variations of gate delay across different mappings, which lead to a suboptimal delay optimization in the final mapping. This motivates us to develop an efficient method that can estimate the gate delay more accurately according to the intermediate mapping solution.

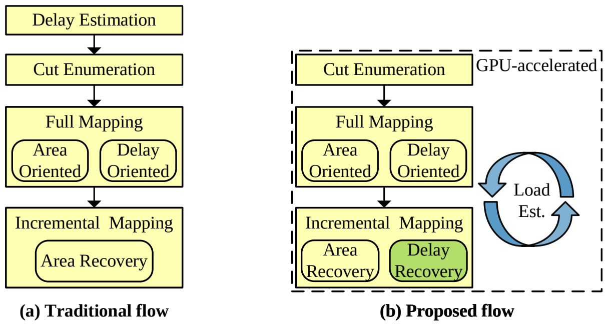

Fig. 1. Comparison of traditional flow and proposed flow.

In addition, GPU computing has emerged as a promising approach to address runtime scalability challenges, leveraging enhanced parallelism and increased memory bandwidth. Numerous GPU-accelerated algorithms have demonstrated substantial improvements in efficiency across various EDA problems [14]–[19]. Many of these algorithms are related to logic synthesis, particularly due to the level-wise characteristics of Boolean circuits. Notably, FineMap [14] proposed a data-parallelism algorithm utilizing a GPU to accelerate the LUT technology mapping, achieving 59.8× speed up over the public synthesis tool ABC [1]. Despite the algorithms for ASIC mapping sharing similarities with those for LUT mapping, there are significant differences between the two (e.g., mapping configuration). We need specific optimizations tailored for ASIC mapping to yield a greater speedup.

In this paper, we propose an ASIC mapper named ExactMap, which considers the exact load information of intermediate mapping circuits during the mapping procedure, as depicted in Figure 1. This enables the detection of critical nodes and the selection of delay-oriented cells. Furthermore, the algorithm is fully parallelized using GPU to achieve high quality in extremely short runtime through configuration-level parallelism. Our contributions can be summarized as follows:

- We introduce a gate delay estimation method that effectively utilizes the load information from the intermediate mapping solutions and relationships between two consecutive solutions.
- Based on the more exact gate delay estimation, we propose an approach to identify critical nodes and select more accurate delay-oriented cells during the mapping procedure.
- We propose a delay-area dual recovery algorithm leading to further improvements in both delay and area.
- We present a configuration-level parallelism framework designed for ASIC mapping and integrate the proposed algorithm into this framework, resulting in high-quality results with significant acceleration.

We conduct experiments using three distinct benchmark suites: ISCAS 85 [20], EPFL [21], and IWLS 2005 [22] benchmarks mapped on ASAP 7nm library [23] and Nangate 45nm library [24]. Compared to the high-performance ASIC mapper implemented in the academic logic synthesis tool ABC [1], ExactMap achieves on average 32% reduction in delay with 3% increase in area, while requiring less than 2% of the runtime of ABC. Compared to the state-of-the-art mappers SLAP [9], LEAP [10], and AiMap [11], ExactMap can averagely achieve at least 17% and 7% reduction in delay and area, respectively.

II. Preliminaries

A *Boolean network* can be represented as a directed acyclic graph (DAG) where each node corresponds to a Boolean function, and the edges signify the input/output signals of these nodes. The *transitive fanins* and *transitive fanouts* of a node represent its predecessors and successors, respectively. Specifically, the direct predecessors and successors are referred to as *fanins* and *fanouts*. The *primary inputs* (PIs) and *primary outputs* (POs) serve as the sources and sinks of the DAG.

An *And-Inverter Graph* (AIG) is a Boolean network where all nodes are two-input AND gates with optionally complemented fanin signals. *Phase* indicates whether the signal of a node is used directly (positive) or inverted (negative), and $\neg n$ denotes the opposite phase of node n.

A *technology library* is a collection of pre-designed logic cells, where each cell function is represented as a circuit of primitive gates. A library provides detailed characteristics of its logic cells, e.g, area, pin capacitance, slew and delay lookup-table. *Slew* refers to the rate of change of the output signal voltage over time. Before physical design, the input slew of a node is equivalent to the maximum output slew of its fanin and is typically assigned zero for PIs. *Load* is the summation of the pin capacitance of all fanouts connected to a given node. *Delay lookup-table* is a table that describes the delay from an input to an output of a cell, contingent upon the respective input slew and load. For simplification, their relationships can be expressed as

$$\text{delay} = f(\text{input slew}, \text{load}) \tag{1}$$

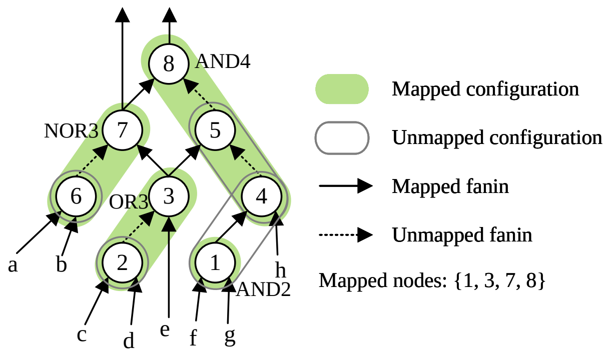

Fig. 2. Illustration of ASIC mapping.

Similarly, the slew lookup table provides information regarding the output slew. Note that delay and output slew are usually positively correlated with input slew and load.

A *cell network* is a Boolean network in which each node corresponds to a specific function cell provided by the technology library. A key aspect of the mapping process is the definition of the arrival time for each node. The arrival time of a node is defined as the longest path from any PI to that node, computed recursively as

$$A(n) = \max_{i \in fanins(n)} (d(i) + A(i)) \tag{2}$$

where $d(i)$ denotes the gate delay from fanin i to the output n. The global delay of the network is the maximum arrival time of all the POs, while the area is simply the summation of all the gates in the network.

For a node n in a Boolean network, a *cut* of n is a set of nodes in the network such that any path from a PI to n passes through at least one node in the set. The *logic cone* associated with a cut of n includes the node n itself, along with the intersection of all transitive fanins of n and all transitive fanouts of the nodes in the cut. A *configuration* of a cut corresponds to a cell from the technology library that is functionally equivalent to the logic cone of the cut.

ASIC mapping transforms a Boolean network of a particular representation into a cell network. In this paper, we focus on the application of this mapping process to AIGs but the proposed methodology is also applicable to other representations. During the ASIC technology mapping process, a final configuration for each node is computed, and a subset of these nodes is selected for mapping. The objective is to ensure that the final configurations of the selected nodes cover all necessary logic (i.e., non-PI nodes) in the original AIG. *Mapped node* denotes the selected nodes, while *unmapped node* represents the others. An illustration of ASIC mapping is shown in Figure 2.

III. Proposed Algorithms

This section introduces our ASIC mapper, ExactMap. We begin with an overview of the mapping framework. We then describe how to leverage the exact load information of the intermediate mapping to estimate the load flow for the next

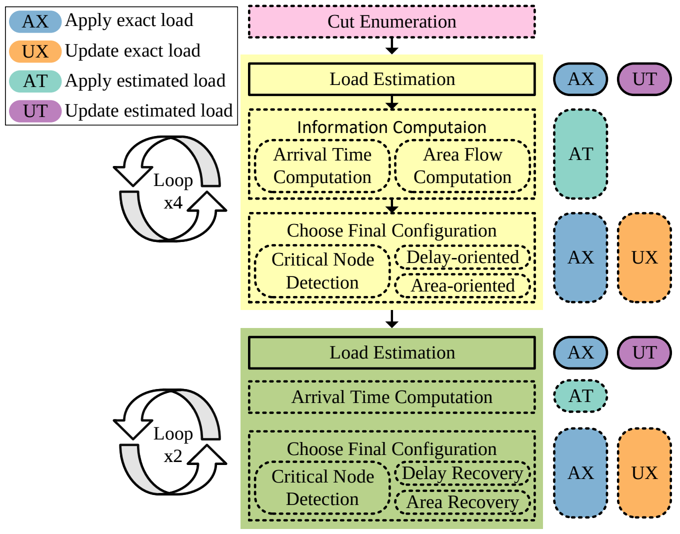

Fig. 3. Mapping framework. Yellow phase represents full mapping, green phase denotes incremental mapping. Dashed Lines indicate process level-by-level, solid lines indicate simultaneously process of the entire graph.

iteration. Next, we will present an innovative strategy for identifying the critical nodes and delay-oriented configurations based on the mapped gates. Following this, we will propose a delay-area dual recovery algorithm to further optimize area and delay. Finally, we introduce a configuration-level parallel framework designed to enhance the performance.

This work focuses on estimating load information and integrating it into ASIC mapping. For the input slew, we employ the median value derived from a delay lookup table.

A. Main Framework

The overall framework of ExactMap is illustrated in Figure 3. The mapping process consists of three major phases: cut enumeration, full mapping, and incremental mapping. Each mapping iteration starts from a load estimation to improve the accuracy of delay computation in mapping procedure.

Cut Enumeration: This is a standard phase in technology mapping. For each node n, at most C priority k-feasible cuts are computed, where k denotes the maximum number of leaves in each cut. More details can be found in [9], [25]. The priority cuts for each node will be used in the subsequent mapping phases.

Load Estimation: If no mapping solution is available (at the beginning), the estimated load of each node is initialized using the inverter's pin capacitance. Otherwise, the estimated load is computed based on the last mapping solution by propagating load flow between fanins and fanouts. The detailed estimation procedure is described in Section III-B.

Full Mapping: In this phase, the arrival time and area-flow [26] for each node are computed in topological order first. With the estimated load, arrival time $A(\cdot)$ can computed as

$$A(n) = \min_{c \in CFG(n)} \max_{n' \in FI(c)} (A(n') + d(n', c)) \tag{3}$$

where $CFG(n)$ denotes the candidate mapping configurations of n, $FI(c)$ represents the fanins of configuration c, and $d(n', c)$ is the gate delay from n' to n under configuration c computed with estimated load. The mapper will also check whether adding an inverter on the output of the opposite phase can give a smaller arrival time. The area-flow [26] can also be computed efficiently in a similar way.

At the end, mapper chooses the final configuration in reverse topological order. If a node is detected as a critical node with the method in Section III-D, the mapper will select a delay-oriented configuration as its final configuration using the method in Section III-C, otherwise an area-oriented [26] configuration will be selected.

Incremental Mapping: In this phase, the mapper will incrementally update the results of the last mapping iteration. First, final configurations of unmapped nodes, i.e., nodes sitting inside the configurations of the mapped nodes (Figure 2), are assumed to be delay-oriented configurations for conveniently computing arrival time and exact area (local area) [8]. Then, each node is visited in topological order. Mapper updates final configurations for delay and area [8] recovery according to the exact area and arrival time using the method in Section III-E.

B. Load Estimation

Due to the inter-dependency between delay computation and mapping solution [12], one can only estimate delay during the mapping process, and the accuracy of the estimation will significantly impact the overall quality of the mapped circuits. The delay estimation in traditional approaches like ABC exhibits a substantial discrepancy compared with the final measured delay [12]. Although some methods [11] have been proposed to estimate delay using machine learning techniques, these approaches still provide only static delay estimation before the mapping phases, lacking the ability to adapt dynamically during mapping. Our objective is to develop a method for online delay estimation throughout the mapping process.

Since a complete (but intermediate) mapping solution can be obtained after each mapping iteration, we can accurately calculate the load for each node. However, simply using the intermediate load information of each node as an estimation for the next iteration may lead to significant errors due to the differences between two consecutive mapping solutions. Nevertheless, we identify some relationships between two consecutive solutions based on the following observation, and we will compute some *load flow* between nodes in order to estimate the load information more accurately when the mapping solution evolves.

Observation 1. The new load of a node in a subsequent mapping solution can be primarily estimated from three components, opposite phase, fanin nodes and fanout nodes, in the previous mapping solution.

Mapped nodes and fanins of their final configurations typically exhibit minor differences between two consecutive solutions. But even the same cut with different configurations

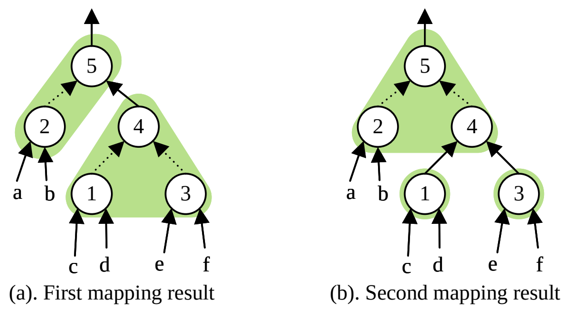

Fig. 4. Illustration of two different mapping results.

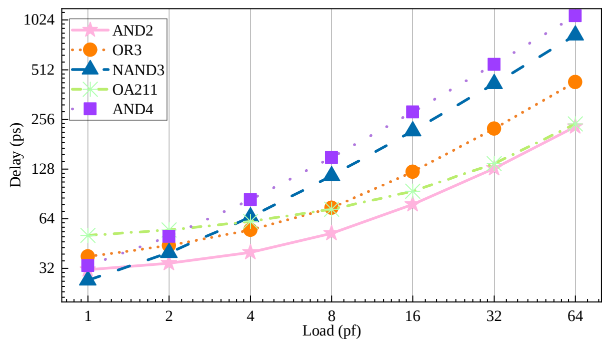

Fig. 5. Gate delay with different load.

may connect with different phases of the fanins. For instance, OR2 and NAND2 gates can be mapped with the same cut, since the Boolean algebraic expression of OR2 is $Y = A \vee B$ and that of NAND2 is $Y = (!A) \vee (!B)$. OR2 connects with the positive phase of its fanins while NAND2 connects with the negative ones. Therefore, if one node is mapped with the same cut but with different configurations, the load of the fanins may transfer from one phase to the opposite phase.

However, the configuration of a mapped node may change more between two consecutive iterations. An example is illustrated in Figure 4, when the mapping result transits from (a) to (b), node 5 is still a mapped node, but its final configuration has changed from {a,b,4} to {a,b,1,3} since the cut is extended from node 4 to its fanins. Consequently, the fanins of node 4 (i.e., nodes 1 and 3) become mapped nodes in (b), resulting in new load for them. From the perspective of load transfer, the load of nodes 1 and 3 can be considered the transferred load from their fanout (i.e., node 4).

Conversely, considering the transition from (b) back to (a), the load of node 4 comes from its fanins (i.e., nodes 1 and 3). There may also be load flow between transitive fanins and fanouts, e.g., node 5 and node 1, but balancing computational complexity and accuracy, we only consider the load flow between fanins and fanouts.

In order to capture these relationships between consecutive iterations, fanin load flow $LF_{fi}(\cdot)$ and fanout load flow $LF_{fo}(\cdot)$ are defined to estimate the new load of a node transferred from its fanins and fanouts, respectively. The fanout load flow of a node n, which estimates the amount of load node n receives from its fanouts, is calculated as

$$LF_{fo}(n) = \sum_{n' \in FO(n)} \frac{2 * \frac{\alpha}{3}(L(n') + L(\neg n'))}{2 * |FI(n')|}) \tag{4}$$

In this equation, $FI(\cdot)$ and $FO(\cdot)$ respectively denote the fanins and fanouts of the node, $L(\cdot)$ represents the *exact load* derived by the last mapping solution. Here, α is a parameter named *transfer proportion*, indicating the proportion of the exact load that will be transferred out in the next solution, with $0 \leq \alpha \leq 1$.

We first explain the numerator part. The numerator represents the total estimated load transferred from n' to its fanins. Specifically, $L(n') + L(\neg n')$ represents the total exact load of two phases of n'. Multiplying with $\frac{\alpha}{3}$ means α of the exact load is estimated to be transferred, and it will be averagely transferred in three parts (i.e., fanins, fanouts, and opposite phase), here we only get the part for fanins of n'.

Note that for fanout load flow, we assume that the cut extends from one node to its fanins. For instance, in Figure 4, we assume the cut of node 5 extends from {a,b,4} to {a,b,1,3}, i.e., extends from node 4 to nodes 1, 3. Then, both nodes 1 and 3 will get one unit of load instead of half unit. That is, one unit of node 4 becomes two units and are averagely divided to its fanins (i.e., nodes 1 and 3). Thus, we need to multiply with 2 in the numerator.

The denominator denotes all the locations that fanout load flow will be transferred to, and fanout load flow is averagely allocated to these locations. Since there are two phases for one fanin, it is $2 * |FI(n')|$ here. Note that $|FI(\cdot)| = 2$ in AIG. Similarly, the fanin load flow can be computed as

$$LF_{fi}(n) = \sum_{n' \in FI(n)} \frac{\frac{1}{2} * \frac{\alpha}{3}(L(n') + L(\neg n'))}{2 * |FO(n')|} \tag{5}$$

In this equation, the factor '$\frac{1}{2}$' represents that two units of load from the fanins will be merged into only one unit of load for the fanout.

Finally, the load flow from the opposite phase can be easily calculated by $\frac{\alpha L(\neg n)}{3}$. Once the load flows are ready, *estimated load* $EL(\cdot)$ for each node is computed as

$$EL(n) = (1 - \alpha)L(n) + LF_{fi}(n) + LF_{fo}(n) + \frac{\alpha L(\neg n)}{3} \tag{6}$$

For a node n, α of $L(n)$ will be transferred out, remaining $1 - \alpha$ of it. And it will get new load from fanins $LF_{fi}(n)$, fanouts $LF_{fo}(n)$ and opposite phase $\frac{\alpha L(\neg n)}{3}$.

C. Delay-oriented Configuration

Accurately identifying delay-oriented configurations is crucial because these configurations are applied to nodes on critical paths, directly influencing the global delay of the mapped circuit. Existing ASIC mapping algorithms [11], [12] typically assume either a constant delay for each cell or a static load for each node before the mapping procedure. However, gate delay is highly dependent on the load, and the selection priority of different gates can vary significantly under different load conditions.

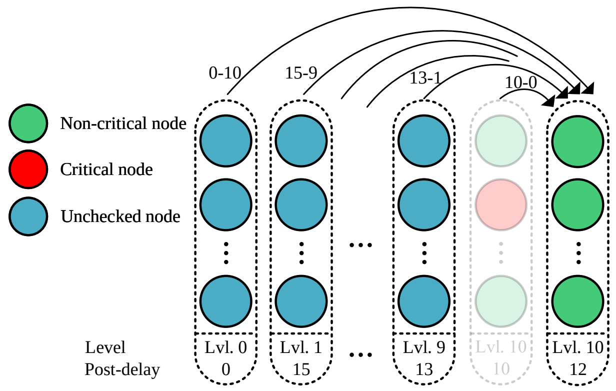

Fig. 6. Illustration of critical node detection. Assuming penalty is 1, then the expressions upside the arcs denote $PD_{level}(\cdot)$ − level_difference.

Figure 5 illustrates the gate delay of several gates from the ASAP 7nm library across different load. It is obvious that all the gate delays increase with load, yet the relative delay order may change as the load varies. For instance, when the load is 1, OR3 exhibits a larger delay than NAND3, but when the load is 32, the delay of NAND3 becomes significantly larger. Note that this is not a special case in modern cell libraries, and obtaining the exact load when selecting a delay-oriented configuration can remarkably improve the global delay.

In fact, when we select the final configuration for a node n in reverse topological order, the load of n can be computed exactly since the final configurations for the fanouts of n have already been decided. This allows us to recompute the gate delay of the candidate configurations and select a correct delay-oriented configuration. However, load estimation remains necessary because the arrival time of transitive fanins still depend on the estimated load. Specifically, we first utilize the estimated load to calculate an estimated arrival time for each node in topological order. We then choose the final configurations for the mapped nodes in reverse topological order. If a node is identified as critical (Section III-D), we recompute the gate delay and arrival time for its candidate configurations using the exact load, and then select the one with the minimum arrival time as the final configuration.

D. Critical Node Detection

Critical node is defined as a node mapped on the critical path in the final mapping solution. Accurately detecting such nodes is essential, as it guides the choice between delay-oriented and area-oriented configurations, ultimately influencing both global delay and area optimization. However, detecting critical nodes is challenging when we choose the final configuration in reverse topological order, since the final configurations of transitive fanins remain undetermined. Traditional approaches [1] classify a node whether is critical by comparing its estimated arrival time against its estimated required time. However, this method heavily relies on the precision of the delay estimation, which is often limited and may lead to incorrect classifications.

To accurately identify critical nodes, it is essential to prioritize confirmed information over uncertain information. Here, the nodes whose final configurations have been decided can provide confirmed data, whereas other undecided nodes will introduce uncertainty. Recall that the final configurations are selected in reverse topological order. When we are deciding the final configuration for a node, its exact load is precisely known, allowing us to compute an exact gate delay once a cell is chosen. Using this information, the delay from the current node to any POs can be computed by summing up the gate delay along any reachable paths. We denote the maximum of these delay as *post-delay* $PD_{node}(\cdot)$, which can be computed recursively during the final configuration decision process as

$$PD_{node}(n) = \max_{n' \in FO(n)} GD(n, n') + PD_{node}(n') \quad (7)$$

where $GD(n, n')$ represents the gate delay from n to its fanout n'. However, the arrival time of the node is still needed to be determined, but the exact load and the final configurations of the transitive fanins are unavailable, preventing one from getting the same precision as for the post-delay. Simply summing up the estimated arrival time with the precisely computed post-delay will compromise the accuracy of the overall delay assessment, undermining the benefits of having exact post-delay values.

To better separate reliable and estimated information, we adopt a level-by-level final configuration decision strategy. This strategy enables the evaluation of all nodes at the same topological level in a single batch. Moreover, given their structural similarity, it is reasonable to assume that nodes within the same level exhibit similar arrival times. Under this assumption, whether a node is critical can only be assessed by comparing its post-delay. Specifically, let $PD_{level}(l)$ denotes the maximum $PD_{node}(\cdot)$ among all the nodes in level l, a node n is a critical node in this level if $PD_{node}(n) = PD_{level}(l)$, otherwise it is not.

However, a critical node in one level may not be critical of the global circuit graph, as the critical path may not pass through the current level. To address this issue, we will also consider the post-delay of other nodes from preceding levels. Since the fanouts of these nodes may not yet be fully decided, their post-delay are computed based on currently decided fanouts. Additionally, as these nodes are at smaller topological levels, we need to add back an estimated delay due to the level differences. We introduce a *penalty-adjusted* post-delay computed as $PD_{level}(l - i) - i * penalty$, where $i > 0$ denotes the difference in level, and the term *penalty* reflects an estimated per-level delay.

In this work, we assume that the nodes between two adjacent levels are configured as a two-input AND gate (AND2) loaded with an inverter's pin capacitance and use its gate delay as the penalty term. Although the actual configurations between the two levels are unknown, we assume a baseline mapping consistent with the original AIG. An example is shown in Figure 6. Assuming the delay of this AND2 gate is 1, and critical node detection is now performed at level 10. Originally,

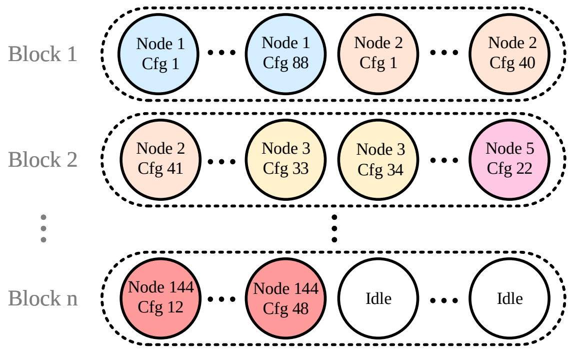

Fig. 7. Illustration of configuration parallelism. Different colors denote configurations of different nodes.

Algorithm 1 Configuration allocation

```
Input: Node set $N$, node's configuration amount $NC$
1:  $TC \leftarrow$ inclusive_scan($NC$).back() ▷ Total configurations
2:  for $n \in N$ in parallel do
3:      $nid \leftarrow$ global thread id
4:      $suf \leftarrow 0$
5:      if $nid \neq 0$ then
6:          $suf \leftarrow NC[nid-1]$        ▷ Get offset from prefix
7:      for $i = 0$ to $NC[n] - suf - 1$ do
8:          $m[suf + i] \leftarrow \{nid, i\}$  ▷ Allocate configurations
9:  for $i = 0$ to $TC - 1$ in parallel do
10:     $\{nid, j\} = m[i]$
11:     Evaluate($N[nid], j$)             ▷ Evaluate configurations
```

without considering penalty-adjusted post-delay from preceding levels, the red node is detected as a critical node with a post-delay of 10. However, when penalty-adjusted post-delay from preceding levels are considered, the critical threshold for level 10 becomes 12 (i.e., from level 9), and the red node is no longer labeled as critical.

E. Delay-area Dual Recovery

The incremental mapping phase in previous methods [1], [9], [11] can be viewed as a sequence of area recovery iterations. In this phase, the global delay obtained from the previous iteration is considered as a constraint, rather than an optimization objective. Under this constraint, the algorithm prioritizes area minimization by selecting final configurations with the smallest exact area [8] without violating the global delay constraint. However, this strategy relies heavily on precise delay estimation, which may not be available with their estimation mechanisms.

In contrast, we will continue to treat global delay as an optimization objective, aiming to jointly improve both delay and area throughout the incremental mapping phase.

First, we will determine whether a node is a critical node using the method described in Section III-D. If the node is identified as a critical node, we will aim to reduce the global delay without increasing the exact area. To achieve this, the exact area of all the candidate configurations will be evaluated, and the one with the smallest arrival time without increasing the exact area will be selected. If the node is a non-critical node, the primary goal will be shifted to area reduction without increasing the global delay. In this case, global delay is used as a filtering criterion for configuration selection. Leveraging the post-delay computation, we can directly set the global delay bound as $PD_{level}(0)$. The configurations whose estimated delay (i.e., $A(n) + PD_{node}(n)$) is larger than the global delay bound will be excluded. Among the remaining candidates, the configuration with the smallest exact area will be selected.

F. Configuration-level Parallelism

GPU acceleration has demonstrated significant improvements in logic synthesis [27] [28]. A straightforward approach to leverage GPU parallelism is to implement level-wise parallelization, where all nodes at the same topological level are evaluated simultaneously, assigning one thread per node. However, this coarse-grained method cannot fully utilize the computational capabilities of modern GPUs, leading to under-utilization of available resources.

For LUT mapping, Liu et al. [29] proposed a fine-grained parallelism strategy that achieves higher performance by exploiting intra-node parallelism. Techniques for cut enumeration, area evaluation, and candidate structures matching proposed in their approach are utilized in our ASIC parallel framework.

However, their approach allocates one thread to a node if its amount of candidate structures smaller than 16, and allows a maximum of 128 threads per node. As a result, a single thread may still be responsible for evaluating 15 or more configurations in ASIC mapping, while many other threads may remain idle, leading to suboptimal resource utilization. To fully leverage GPU capabilities in ASIC mapping, it is necessary to adopt configuration-level parallelism.

However, since the amount of the candidate configurations may be vastly different between two nodes, if one block is assigned to a single node, significant thread under-utilization may occur when a node has relatively few configurations. To deal with this issue, we allocate each block with a fixed number of 128 candidate configurations, and allow the configurations within one block coming from different nodes. This strategy ensures more balanced utilization of GPU resources.

As illustrated in Figure 7, 88 candidate configurations from node 1 and 40 configurations from node 2 are jointly evaluated within block 1. With this approach, only the last block may contain idle threads, depending on whether the total number of configurations is divisible by the block size.

To enable efficient parallel evaluation, each thread must be aware of the specific configuration it is responsible for. The configuration allocation procedure is shown in Algorithm 1. We first compute the total number of candidate configurations and generate a prefix-sum (suffix amount) for each node to indicate the cumulative number of configurations up to that node (line 1). We then construct a mapping from global thread

TABLE I
COMPARISON OF MAPPING RESULTS ON NANGATE 45NM

Benchmarks	AIG Statistics		ABC *&nf*				ExactMap (Ours)			
	#Nodes	Levels	Delay(ps)	Area(μm^2)	#Gates	Time(s)	Delay(ps)	Area(μm^2)	#Gates	Time(s)
twentythree	23339737	176	3179711.50	11722626.00	9650734	1062.88	1290931.75	12120288.00	10281216	12.41
twenty	20732893	162	2917744.50	10396335.00	8534469	911.81	983041.31	10749248.00	9091948	10.80
sixteen	16216836	140	3206460.50	8168299.00	6651909	559.92	1043753.00	8455447.00	7117697	7.99
log2_9xd	16414720	444	10162.94	18559.62	15275	475.65	8841.16	18129.50	15238	5.61
mem_ctrl_9xd	23980032	114	5939.82	28680.92	26954	633.44	3319.81	29775.51	28682	9.45
hyp_7xd	27434880	24801	942333.31	168188.61	149794	604.84	540335.19	155111.78	145766	25.06
div_8xd	14655232	4372	84361.55	48665.23	50768	380.93	91829.52	44777.38	47081	7.02
sin_10xd	5545984	225	5410.95	3729.85	3253	172.09	4503.03	3499.23	3127	2.01
sqrt_10xd	25208832	5058	344464.25	22382.04	18840	606.76	143581.20	17762.95	17216	11.27
voter_10xd	14088192	70	1924.63	14392.46	12821	283.56	1730.78	10131.94	10311	4.05
square_10xd	18927616	250	4262.53	12675.96	11301	369.87	3902.93	12694.05	11607	5.25
multiplier_10xd	27711488	274	5718.16	16411.4	13918	582.64	5730.14	15314.15	12287	8.13
ac97_ctrl_10xd	14610432	12	1358.39	8650.05	6331	101.88	812.45	8995.06	6614	2.35
vga_lcd_5xd	4054752	24	147372.78	69304.97	46894	50.75	63864.27	76480.32	48188	1.03
Geomean Ratio			**1.00**	**1.00**	**1.00**	**65.33**	**0.66**	**0.96**	**0.99**	**1.00**

"_$n$xd" means that the benchmark is generated by enlarging the original one using ABC *double* n times.

id to their corresponding configurations in parallel, using the prefix-sum information to determine the node and configuration index (lines 2–8). Once this mapping is established, the evaluation of the configurations can proceed in parallel (lines 9-11).

IV. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed GPU ASIC technology mapping algorithm, we conduct comprehensive experiments on 30 public benchmark instances using two different technology libraries. We compare the results with public synthesis tool ABC [1] and the state-of-the-art (SOTA) ASIC mappers (i.e., SLAP [9], LEAP [10] and AiMap [11]).

A. Experimental Protocol

Our proposed GPU ASIC mapping engine ExactMap is implemented in CUDA C/C++, and the experiments are performed on an Ubuntu 18.04 server with Intel Xeon Silver 4114 CPU and NVIDIA GeForce RTX 3090 GPU with 24 GB dedicated DRAM.

The testcases are from the EPFL combinational benchmark suite [21], IWLS 2005 benchmarks [22] and ISCAS 85 benchmarks [20], consisting of arithmetic, control and random designs. To demonstrate the effectiveness of parallelism, we enlarge some designs by applying multiple times of the ABC command *double*, which is a common method used in the works on parallel logic synthesis and verification [14], [27], [28]. The statistics of them are presented in Table I. ASAP 7nm library [23] and Nangate 45nm library [24] are selected as the technology library.

The results of SLAP [9], LEAP [10] and AiMap [11] are obtained from their original papers. There are two ASIC mapping algorithms in ABC with different commands (i.e., *map* and *&nf*). Despite their different interfaces, the core ideas and procedures are similar. Experiments show that their qualities are similar but *&nf* takes less memory and runtime, thus we compare with *&nf*. The maximum cut size k and the number of priority cuts per node C are respectively set as 6 and 16 in all the experiments, which is the same as in ABC. The load transfer proportion α is set as 0.7 and 0.3 for full mapping and incremental mapping, respectively. All the results are evaluated with command *stime* in ABC and pass the equivalence checking.

B. Computational Results

The comparisons between ExactMap and the reference algorithms are presented in Table I and Table II. Note that "#Nodes" and "Levels" denote the number of nodes and topological levels of the input AIGs. Column "Delay(ps)", "Area(μm^2)" and "#Gates" represent the delay, area and total number of gates of the mapped circuits, respectively. Column "Time(s)" displays the execution time which does not include the file I/O time.

1) Comparison with ABC: Table I presents the results obtained by ABC and ExactMap using Nangate 45nm library. To simplify the analysis, the results of area and gate number are derived from the original cases rather than the enlarged ones, which will not influence the conclusions. On average, ExactMap achieves 34% reduction in delay and 4% reduction in area. These improvements may generally be attributed to two factors. One is reduction in the total number of gates, which will directly impact both the area and delay. The other factor is a better selection of mapped nodes and corresponding configurations without reducing the total number of gates. From Table I, it is evident that the total number of gates is similar between the two mappers. This shows the performance gained by ExactMap comes primarily from the second factor, the selection of a better configuration for each node. This outcome aligns with the design philosophy of ExactMap.

In terms of runtime, ExactMap achieves on average 65.33× speedup over ABC, demonstrating both its effectiveness and efficiency. In particular, for case sin_10xd, ExactMap achieves an 85.6× speedup. We also evaluate the runtime on ASAP 7nm library, ExactMap achieves on average 63.18× speedup over

TABLE II
COMPARISON OF MAPPING RESULTS ON ASAP 7NM

Benchmarks	ABC *&nf*		AiMap [11]		SLAP [9]		LEAP [10]		ExactMap (Ours)	
	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)	Delay(ps)	Area(μm^2)
log2	7042.22	20138.83	6855.81	23330.10	-	-	-	-	6747.54	20475.45
mem_ctrl	3370.93	31166.21	-	-	3873.23	32861.70	3779.39	32555.62	2631.49	32710.52
hyp	526047.44	181084.77	-	-	-	-	-	-	357009.34	190478.48
div	81319.06	49418.04	-	-	80775.56	62818.57	77108.30	54247.63	71559.72	51561.41
sin	3627.46	3976.02	3599.18	4503.00	3584.79	5087.60	3817.87	5208.44	3227.95	3901.37
sqrt	159342.38	19038.68	185280.97	19918.61	-	-	-	-	102903.13	20568.06
voter	1146.05	12126.59	-	-	1242.51	17517.46	1210.45	18653.54	1161.00	11186.71
square	3910.56	14160.10	-	-	3023.01	15789.09	3125.07	14638.55	2941.79	14300.30
multiplier	4450.36	18108.83	4512.38	20106.40	4278.48	24021.07	4075.31	24860.42	3909.97	18082.70
ac97_ctrl	914.09	9942.63	-	-	-	-	-	-	452.70	10225.60
vga_lcd	31788.49	76091.73	-	-	-	-	-	-	16260.86	76403.40
adder	3992.49	833.28	3486.11	1060.96	3268.67	1031.33	3415.11	1010.34	2799.36	1079.62
bar	1074.48	2371.76	1058.98	2059.40	923.82	3083.23	1114.90	2680.39	648.58	2344.00
max	5416.57	2119.82	5035.80	2128.20	3710.54	2292.44	3970.56	2067.09	2735.62	2188.87
C6288	1306.55	1738.40	1385.40	2479.53	1236.59	3023.54	1250.44	2951.93	1395.75	1792.29
C7552	789.08	1473.63	913.78	1758.93	800.09	2002.01	787.96	1939.96	541.16	1495.56
Geomean Ratio	**1.32**	**0.97**	**1.31**	**1.07**	**1.17**	**1.22**	**1.19**	**1.17**	**1.00**	**1.00**

"-" represents the results are not reported in their papers.

ABC. Note that ExactMap is in GPU parallelism but it runs deterministically, which is enabled by our algorithmic design.

2) Comparison with SOTA algorithms: SLAP [9] and LEAP [10] are ASIC mapping algorithms that choose the priority cuts by ML techniques. AiMap [11] is proposed to estimate the delay of the cells before mapping phases using structural information. These approaches are built upon the command *map* in ABC, SLAP and LEAP target at delay optimization, while AiMap targets at area reduction.

Table II presents the results obtained by these SOTA methods and ExactMap using ASAP 7nm library. One can observe that ExactMap achieves on average over 17% reduction in delay. In terms of area, ExactMap also demonstrates 7%, 22% and 17% reduction over SLAP, LEAP and AiMap, separately. When compared with ABC under this library, ExactMap achieves an average 32% reduction in delay with a slight increase in area (i.e., 3%). Notably, for cases *bar* and *C7552*, ExactMap can achieve more than 30% reduction in delay compared to all other algorithms, which once again shows the superiority of our proposed algorithm.

We also evaluate these cases with Nangate 45nm library. Compared with ABC, ExactMap achieves average reductions of 36% in delay and 2% in area, demonstrating the robustness of ExactMap across different technology libraries. Note that existing SOTA approaches (SLAP, LEAP, and AiMap) did not report results for the Nangate 45nm library in their papers.

3) Runtime Analysis: To show the effectiveness of the configuration-level parallelism, Figure 8(a) presents the speedup ratio of the configuration-level parallelism over the cut-level parallelism. One observes that the configuration-level parallelism accelerates over 60% in all the cases over the cut-level parallelism. Especially for three cases (i.e., voter_10xd, sin_10xd and log_9xd), the speedup is more than 3 times.

The fractions of the total runtime for cut enumeration (CE), full mapping (FM) and incremental mapping (IM) are shown in Figure 8(b). The speedup ratios of these three phases/passes

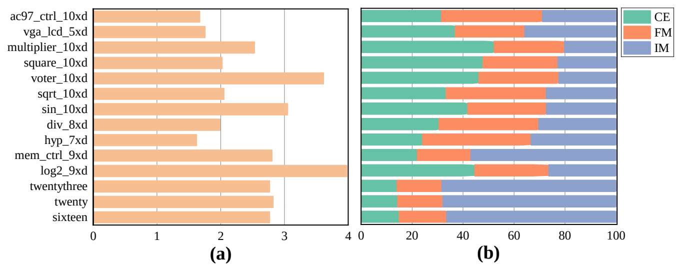

Fig. 8. (a) Speedup ratio of configuration-level parallelism. (b) Percentage of runtime.

are 86.5×, 56.7× and 104.8× over the ABC counterparts.

V. CONCLUSIONS

In this paper, we proposed a GPU-accelerated ASIC technology mapping algorithm considering load estimation during mapping process. Specifically, we estimated the load flow using the load information of the intermediate mapping circuits. Then, a new critical node detection method is presented based on the level and mapped gates after the node. Based on the above, we can select a better delay-oriented configuration for the proper node. Finally, we propose a delay-area dual recovery method to further optimize the delay and area. All of these are incorporated into a configuration-level parallelism framework. Experimental results show that our algorithm can achieve 65.33× speedup compared to the open-source synthesis tool ABC. For the quality, ExactMap achieves 32% reduction in delay with only a slight increase in area compared with ABC. Compared with the SOTA algorithms, ExactMap reduces delay and area by at least 17% and 7%, respectively.

VI. ACKNOWLEDGE

This work is partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 14209624).

References

[1] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.

[2] A. T. Calvino, A. Mishchenko, H. Schmit, E. Mahintorabi, G. De Micheli, and X. Xu, "Improving standard-cell design flow using factored form optimization," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] A. T. Calvino and G. De Micheli, "Technology mapping using multi-output library cells," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[4] A. Costamagna, A. T. Calvino, A. Mishchenko, and G. De Micheli, "Area-oriented optimization after standard-cell mapping," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1112–1119.

[5] R. Murgai, "On the complexity of minimum-delay gate resizing/technology mapping under load-dependent delay model," in *Workshop Handouts, International Workshop on Logic Synthesis*, 1999, pp. 209–211.

[6] A. T. Calvino, G. De Micheli, A. Mishchenko, and R. Brayton, "Enhancing delay-driven lut mapping with boolean decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[7] K. Keutzer, "Dagon: Technology binding and local optimization by dag matching," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, 1987, pp. 341–347.

[8] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 354–361.

[9] W. L. Neto, M. T. Moreira, Y. Li, L. Amarù, C. Yu, and P.-E. Gaillardon, "Slap: A supervised learning approach for priority cuts technology mapping," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 859–864.

[10] C. R. Chigarapally, H. N. Bhakkad, A. B. Chowdhury, C. Karfa, and S. Bhattacharjee, "Leap: Learning guided quality cut selection for faster technology mapping," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–6.

[11] J. Liu, L. Ni, X. Li, M. Zhou, L. Chen, X. Li, Q. Zhao, and S. Ma, "Aimap: Learning to improve technology mapping for asics via delay prediction," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 344–347.

[12] J. Liu, L. Ni, L. Chen, X. Li, Q. Zhao, X. Li, and S. Ma, "A delay-driven iterative technology mapping framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[13] M. Liu, D. Robinson, Y. Li, and C. Yu, "Maptune: Advancing asic technology mapping via reinforcement learning guided library tuning," 2024. [Online]. Available: https://arxiv.org/abs/2407.18110

[14] T. Liu, L. Chen, X. Li, M. Yuan, and E. F. Young, "Finemap: A fine-grained gpu-parallel lut mapping engine," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 392–397.

[15] Y. Sun, T. Liu, M. D. Wong, and E. F. Young, "Massively parallel aig resubstitution," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[16] L. Liu, B. Fu, M. D. F. Wong, and E. F. Y. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[17] S. Lin, J. Liu, and M. Wong, "'instantgr: Scalable gpu parallelization for global routing," in *2024 ACM/IEEE International Conference On Computer Aided Design (ICCAD)*, 2024.

[18] 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 *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[19] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2317–2330, 2022.

[20] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[21] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

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

[23] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[24] "Nangate 45nm open cell library," 2008. [Online]. Available: http://www.nangate.com

[25] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2894–2903, 2006.

[26] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in lut-based fpga technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2331–2340, 2006.

[27] S. Lin, J. Liu, T. Liu, M. D. Wong, and E. F. Young, "Novelrewrite: Node-level parallel aig rewriting," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 427–432.

[28] T. Liu and E. F. Young, "Rethinking aig resynthesis in parallel," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[29] T. Liu, Y. Sun, L. Chen, X. Li, M. Yuan, and E. F. Young, "A unified parallel framework for lut mapping and logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.