Mixed-Cell-Height Legalization on CPU-GPU Heterogeneous Systems

Haoyu Yang[1], Kit Fung[2], Yuxuan Zhao[2], Yibo Lin[3], Bei Yu[2]
[1]nVIDIA [2]Chinese University of Hong Kong [3]Peking University
haoyuy@nvidia.com, byu@cse.cuhk.edu.hk, yibolin@pku.edu.cn

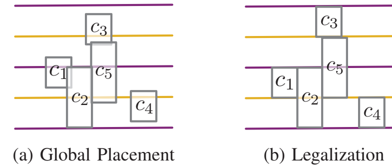
***Abstract*—Legalization conducts refinements on post-global-placement cell location to compromise design constraints and parameters. These include placement fence regions, power/ground rail alignments, timing, wire length and etc. In advanced technology nodes, designs can easily contain millions of mutiple-row standard cells, which challenges the scalability of modern legalization algorithms. In this paper, for the first time, we investigate dedicated legalization algorithms on heterogeneous platforms, which promises intelligent usage of CPU and GPU resources and hence provides new algorithm design methodologies for large scale physical design problems. Experimental results on IC/CAD 2017 and ISPD 2015 contest benchmarks demonstrate the effectiveness and the efficiency of the proposed algorithm, compared to the state-of-the-art legalization solution for mixed-cell-height designs.**

I. Introduction

Legalization is a critical step frequently invoked in modern backend optimization flow. It is often a required step to clean up any design rule violations after global placement optimization or engineering change order (ECO) steps [1] (see Fig. 1). In advanced technology nodes, legalization has to handle large-scale designs with complicated design constraints. Million-size designs are becoming common and constraints like fence regions and multi-row cells increase the difficulty in searching for legal solutions [2], [3]. Meanwhile, the quality of legalization has high impacts on the performance of the follow-up steps like detailed placement and even routing. Thus, ultrafast yet high-quality legalization algorithms are desired to speedup design closure.

Existing legalization algorithms mostly follow a sequential and greedy procedure. NTUplace series adopt Tetris-like approaches [4]–[7] to sort the cells from left to right or right to left and find empty spaces for cells one by one. FastPlace [8], [9] and BonnPlace [10] series choose to distribute cells into bins first and then perform row-based algorithms to remove overlaps [11]–[13]. To handle multi-row cells, Chen *et al*. [14], [15] propose to perform full-chip ordered-row legalization algorithm with relaxed right boundaries and greedily fix the out-of-boundary cells with Tetris-like approaches. Chow *et al*. [16] propose a multi-row local legalization (MLL) algorithm to search for the insertion points for each multi-row cell in a window and choose the one with minimum perturbation. Eh?Placer [17], [18] develops a successive shortest path based algorithm to gradually push cells away from congested regions and remove overlaps. While these algorithms can effectively remove overlaps and design rule violations, they are usually implemented on CPUs without enough parallelism for acceleration.

Recently, several successful attempts to accelerate placement



(a) Global Placement (b) Legalization

Fig. 1 Legalization cleans global placement design rule violations and compromises design constraints and parameters.

algorithms on heterogeneous platforms with CPUs and GPUs have been demonstrated, including global placement [19], [20] and detailed placement [21], [22]. By exploring the massive parallelism in fine granularity, GPU can achieve significant speedup for tasks that are too small to afford the threading overhead on CPU. However, very few efforts have been spent on investigating the possibility of accelerated legalization algorithms so far. Li *et al*. [23] propose multi-row global legalization (MGL) that explores the parallelism of Chow's MLL algorithm by simultaneously searching the insertion points of multiple cells that are far away from each other, while the speedup saturates quickly to $1.13\times$ with 8 or more threads. There are also attempts to use integer linear programming (ILP) for legalizing a local region with minimum perturbation [16], [24]. While disjoint regions can be executed in parallel, the runtime is still unacceptable due to the exponential complexity of solving ILP instances.

To accommodate the demand for ultrafast legalization, in this work, we propose a heterogeneous legalization algorithm for large-scale mixed-cell-height designs leveraging hybrid CPU-GPU platforms. We explore the massive parallelism and develop dedicated GPU acceleration techniques for legalization algorithms in advanced technology nodes. The major contributions are summarized as follows.

- For the first time, we propose an ultrafast legalization framework on heterogeneous computing platforms, targeting at further speedup design closure.
- We propose dedicated CPU and GPU legalization algorithms that promise efficient utilization of CPU and GPU resources respectively.
- We also develop a legalization-aware task scheduling algorithm that can efficiently distribute cell legalization jobs properly on computing resources.
- Experimental results on ISPD 2015 [3] and IC/CAD 2017 contest benchmarks [2] demonstrate that the proposed algorithm can achieve $2\times-4\times$ speedup without quality

978-3-9819263-6-1/DATE22/©2022 EDAA

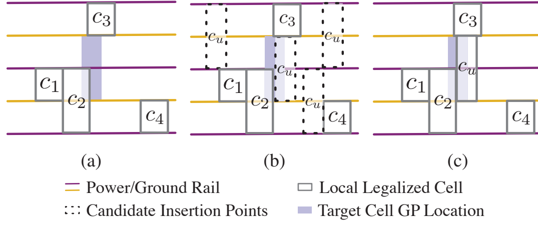


Fig. 2 Visualization of multi-row global legalization algorithm in [23]: (a) Local cell search; (b) Insertion interval evaluation; (c) Target cell legalization.

degradation compared to the state-of-the-art multi-threaded legalizer [23].

II. Preliminaries

A. Legalization Basis

Given a set of multi-row cells $\mathcal{C} = \{c_i\}$ and their global placement location (x_i, y_i), legalization dispatches placed cells to compromise design constraints. As introduced in IC/CAD 2017 Contest [2], legalized cells are (1) not overlapping with each other (2) aligned with layout power lines and (3) entirely placed within assigned fences and regions. According to [3], fence regions are hard constraints that cells assigned to a fence region must be placed inside its boundary, and cells not assigned to it must be placed outside that boundary.

Additionally, to preserve the optimized global placement quality, legalization is expected to introduce smallest cell displacements. Let (x'_i, y'_i) be the location of a legalized cell c_i. The legalization displacement δ_i is defined with Manhattan distance

$$\delta_i = |x_i - x'_i| + |y_i - y'_i|. \tag{1}$$

A good legalization will make as smallest movements as possible. Thus the legalization quality is hence evaluated with average displacement score following [2]:

$$S_{\text{am}} = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \frac{1}{|\mathcal{C}_h|} \sum_{c_i \in \mathcal{C}_h} \delta_i, \tag{2}$$

where $\mathcal{H}$ is the set of all possible cell heights, $\mathcal{C}_h$ is the set of cells that have height h and $|\mathcal{C}_h|$ is the total number of instances in set $\mathcal{C}_h$.

B. Multi-row Global Legalization

To handle multi-row cells, Multi-row Global Legalization (MGL) algorithm is proposed recently [23], which is among state-of-the-art techniques. Fig. 2 visualizes the basic idea of MGL. The algorithm takes an unplaced cell as input and place the cell into a *local region* near its global placement location. Here, c_u is the *target cell* for legalization, and c_i's, $i = 1, 2, 3, 4$ are legalized *local cells* within the rectangular local region $\mathcal{W}_u$. MGL tries to find all the possible positions in $\mathcal{W}_u$ that fit c_u and chooses the one achieving the best overall displacement in the local region after inserting c_u. To avoid unnecessary cell displacements, MGL collects $\mathcal{W}_u$ centered at the the global placement position of c_u. For cells that cannot fit into their local regions, it also gradually increases the window sizes until all the cells are legalized. Results of [23] have demonstrated promising legalization quality. However, with increased complexity and

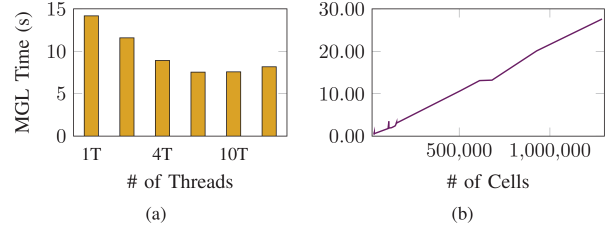


Fig. 3 MGL runtime scales with (a) multi-threading enabled on `des_perf_1` and (b) increasing cell number on 8-thread execution.

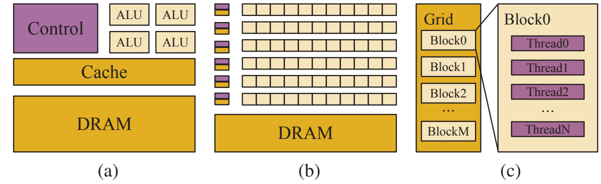


Fig. 4 Comparison between CPU and GPU architectures. (a) Multi-core CPU architecture. (b) GPU architecture. (c) GPU thread hierarchy.

sizes of modern chip designs, legalization overhead cannot be ignored in the physical design flow. Fig. 3(a) plots the runtime scaling of MGL with the number of threads and the benefits start to degrade with eight threads and beyond. With the number of cells growing to beyond millions, legalization overhead becomes significant in physical design especially when it needs to be invoked repeatedly during optimization, as shown in Fig. 3(b).

C. CPU Versus GPU

Fig. 4 illustrates the differences between the representative multi-core CPU architecture and GPU architecture. Although modern CPU and GPU share similar design hierarchy with ALUs, caches, controller and main memory, these components usually come with different scale and characteristics. CPUs are usually designed with small number of powerful ALUs that support a complicated instruction set, with the help of large data caches and strong control units. GPUs, on the other hand, are equipped with computing unit grids of thread blocks and simple control units, which favor massive parallel execution of light-weighted tasks [25] and bring opportunities for efficient legalization algorithms.

III. Heterogeneous Legalization

A. Overall Flow

The overall flow of our heterogeneous legalization framework is summarized in Fig. 5, which includes task scheduling, CPU and GPU legalization kernels. The target of task scheduling is to provide high parallelism and attain legalization quality. This is achieved by assigning cells to the correct device (CPU or GPU) at a proper runtime phase. CPU and GPU legalization kernels are the central components that complete the legalization tasks.

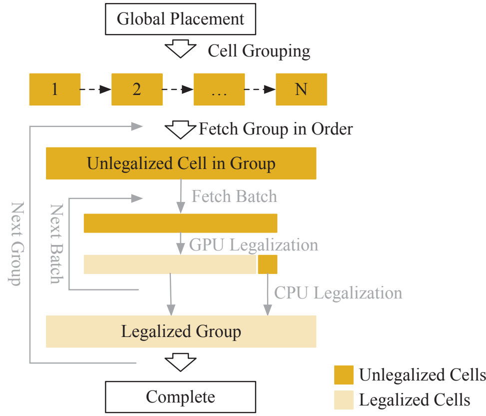


Fig. 5 Flow Summary of Heterogeneous Legalization.

B. CPU Legalization

We first introduce the CPU Legalization algorithm that places a given cell into a legalized location with minimum movements w.r.t. its GP position. Overall the CPU legalization algorithm includes local region extraction, cell insertion point evaluation and cell legalization, which resembles MGL [23] with improved techniques for better runtime.

Local Region Extraction. According to the objectives of legalization, a cell c_u should not be moved far away from its GP position to preserve the GP quality. Therefore, the cell legalization/placement region is supposed to be close to its initial coordinates. Here, following [16], we select a rectangular window $\mathcal{W}_u$ with size (h, w) that is centered at c_u's GP position. $\mathcal{W}_u$ will be the region where legalization of c_u happens and cells outside $\mathcal{W}_u$ will not be affected. Intuitively, CPU legalization can be applied on multiple cells simultaneously as long as their corresponding windows are not overlapped with each other. We will discuss the details of supporting parallelization run later in Section III-D.

After we get the initial search space $\mathcal{W}_u$, we need to collect all the legalized cells (local cells) that touch the window, which will interact with the target cell to be legalized. The simplest way is looping over all the placed cells and comparing their coordinates with $\mathcal{W}_u$, as shown in Fig. 6(a). However, in modern large scale designs, there are usually millions of cells to check, which is far more than the number of threads that can be allocated on both CPU and GPU. Thus the $\mathcal{O}(n)$ complexity makes this step be the runtime bottleneck. To address the concern, we propose an indexed approach to balance the parallelism and resource demands. Here we store all the legalized cells hierarchically into different bins instead of flattened arrays that can reduce the cell query operation into $\mathcal{O}(\log n)$. Observing that cells are seldom moved vertically during the legalization procedure, the entire chip is divided into bins along y-axis. Such bin construction strategy significantly reduces local cell search region as well as controls the overhead that assign legalized cell into certain bins. However, this technique will also cause overhead while maintaining those bins. Thus it can be disabled if the test case is extremely small. Now we only need to examine over the bin regions that touch the legalization window as in Fig. 6(b). Fence

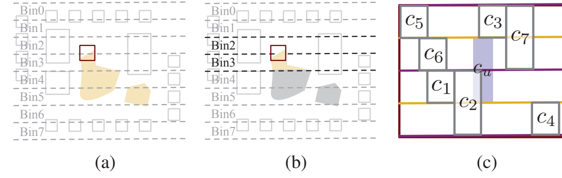


Fig. 6 Local region extraction with bin-aided local cell search: (a) Bin extraction; (b) Local cell search on bins that touch the given search window; (c) Extracted region with local cells.

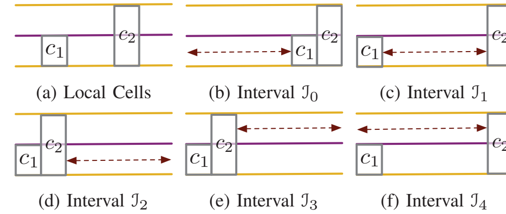


Fig. 7 Enumeration of placement intervals in different rows. Intervals in adjacent rows can be jointed together to support multi-row cell insertion: (a) An example search window with 2 local cells; (b) Bottom row interval with c_1 and c_2 pushed to right; (c) Bottom row interval with c_1 pushed to left and c_2 pushed to right; (d) Bottom row interval with c_1 and c_2 pushed to left; (e) Top row interval with c_1 and c_2 pushed to left; (f) Top row interval with c_2 pushed to right.

regions and non-local cells are processed similarly as in MGL.

Insertion Point Evaluation. The next step is to evaluate the insertion points for the target cell. This includes single-row insertion interval search, multi-row insertion interval search and insertion point evaluation. A single-row insertion interval is a continuous empty area on a row that allows new cell insertion. Candidate intervals consist of cell-to-cell and cell-to-boundary spaces. Let c_l and c_r be the cells that form a single-row interval. To find the maximum width of a single-row interval, we push c_l and c_r of each interval to the left most and right most position in $\mathcal{W}_u$ while not overlapping with other local cells. To place the multi-row height cells, we need multi-row insertion intervals which can be formed by combining the single-row insertion intervals. An example of insertion point enumeration is depicted in Fig. 7.

Cell Legalization. After finding out the best insertion point, we can insert the target cell and move the other local cells in case overlapping occurs. To do so, we build up the neighboring relationship of the local cells and then we can do a breath-first search to place target cell and spread away the overlapped local cells. If a local cell is moved left/right, check whether it will overlap to its left/right neighbor and resolve overlapping by further finetuning.

C. GPU Legalization

The core of GPU legalization algorithm lies in the design of GPU kernel functions which should support multi-level parallelism to maximize computation power. To accommodate the GPU thread hierarchy, each thread block is responsible

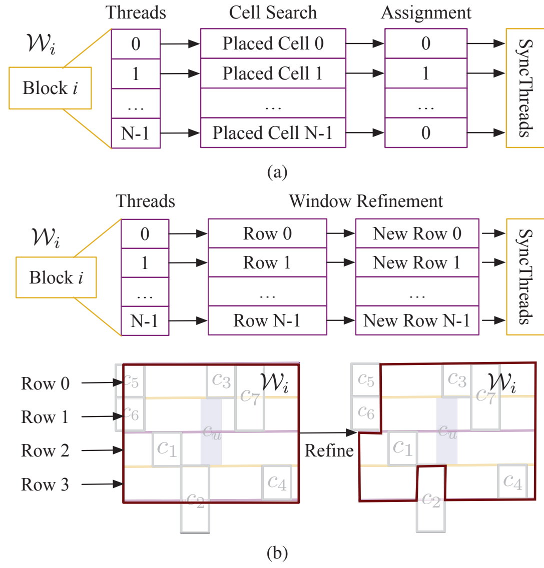


Fig. 8 GPU local region $\mathcal{W}_i$ extraction. CUDA thread management for (a) local cell search and (b) search window refinement, where each thread block is responsible for the search window of one target cell.

to legalize one cell. Note that there are also multiple threads available in each block which enables algorithm design for lower level parallelism.

GPU Local Region Extraction. In CPU legalization, local regions are extracted by checking legalized cells sequentially. GPU thread hierarchy offers a huge speed up when large amount of threads can be allocated to conduct local cell search. In our design, one thread block targets at one search window and all available threads are working simultaneously to search for local cells, as shown in Fig. 8(a). We adopt atomic operation [25] to collect local cells of a given search window in case of conflicts when multiple threads are making modifications on same memory space. At the end of this stage, all threads will be synchronized to assign placed cells to each individual search window $\mathcal{W}_i$. The search window will be further refined by considering cells (fixed cells) that touch the window boundary. Because all cells in the same row are sorted according to their x coordinates, we apply row-level parallelization (each thread searches fixed cells in a single row) in this step (see Fig. 8(b)). Compared to sequential CPU runs, above strategy benefits from GPU thread hierarchy that promises tremendous runtime improvement.

GPU Insertion Point Evaluation. Similar to the search window refinement, we also apply row-level parallelism when extracting single-row insertion intervals with each thread takes care of one row. The difficulty comes from multi-row intervals, because there might be a large amount of multi-row combinations that is far beyond the number of available threads in a thread block. CPU easily handles such situation with queue data structure that is not naturally available in GPU programming. Brute force solution is hence more preferable that we generate all possible combinations of single-row intervals and filter out invalid ones that include illegal combinations and intervals that are not large enough to place the target cell. The total number of combinations grows exponentially to the search window size. To avoid minor cases when there is an extremely large number of valid combinations that hinders parallelization, we will disable recursive window search and distribute failed case to CPU for further processing. Regarding the stage of insertion point evaluation, we assign each thread to calculate the cost of a candidate interval and get the best location in each interval. A parallel reduction will be conducted afterwards on one predetermined thread to get the best insertion location.

GPU Cell Legalization. For target cells that can be successfully inserted, we will still have CPU to spread local cells and resolve overlapping sequentially, as local cells are strictly bonded together that leaves little room for parallelization. It should be also noted that there is no recursive call in GPU legalization to increase search windows. Thus, it is inevitable to have failed cases in target cell insertion as will be discussed in the following section. The overall GPU legalization is summarized in Algorithm 1, which takes the input of a batch of target cells that can be legalized simultaneously and generates legalized cells and information of failed cells. Each GPU thread block is in charge of legalization of one target cell in $\mathcal{C}_B$ and different blocks will work in parallel (lines 2–17). Within each thread block, multiple threads will first search local cells in the given search window (lines 3–4); each thread will search single-row insertion intervals in one legalization row (lines 5–6) followed by the enumeration of multi-row intervals (lines 8–9); for thread blocks that find a valid insertion point for their target cells, multiple threads will simultaneously evaluate the cost of all valid insertion points and place the target cell in the best interval (lines 10–16); finally, we will update the information of all legalized cells and their positions (line 17).

Algorithm 1 GPU Legalization

Require: Batch of target cells $\mathcal{C}_B = \{c_{u,i}, i = 1, ..., N\}$ to be legalized simultaneously, their GP position $(x_{u,i}, y_{u,i})$, collection of all legalized cells $\mathcal{C} = \{c_j\}, j = 1, ..., M$ and their GP positions $\mathcal{P} = \{(x_j, y_j)\}$, local cell search window $(h_{u,i}, w_{u,i})$;
Ensure: Updated legalized cell collection $\mathcal{C}, \mathcal{P}$ and failed cell collection $\mathcal{C}_f$.

```
1: C_f ← C_B;
2: for each blockIdx: B ∈ [0, N − 1] do
3:     for each threadIdx: T ∈ [0, M] do
4:         Compare W_{u,B} with c_T ∈ C;
5:     for each threadIdx: T ∈ [0, R − 1] do
6:         Get single-row intervals in one row T of W_{u,B};
7:     Collect all single-row intervals in W_{u,B};
8:     for each threadIdx: T ∈ [0, C − 1] do
9:         Check validity of T^th multi-row intervals;
10:    if W_{u,B} contains valid insertion interval then
11:        C ← C ∪ c_{u,B};
12:        C_f ← C_f / c_{u,B};
13:        for each threadIdx: T ∈ [0, C_v − 1] do
14:            l_T ← Evaluate legalization cost of T^th interval;
15:        Get the best insertion position according to l_T s;
16:        Legalize cell c_{u,B} and spread local cells in W_{u,B} to resolve overlapping;
17:    Update C, P;
```

The elaborations show that the major difference between multithreading CPU and GPU is their scale and depth of parallelization. CPU legalization works in parallel scheme with each CPU thread handling one target cell legalization and insertion point evaluation are managed by queue structure sequentially. GPU legalization supports large-scale batches that run with much more target cells being processed at the same time by grids of thread blocks while multiple insertion candidates of one target cell are evaluated simultaneously with massive thread resources in each thread block.

D. Task Scheduling

Another key part of our heterogeneous legalization framework is task scheduling which needs to efficiently assign legalization tasks to CPU or/and GPU devices. The core strategy is to assign majority of easy tasks to GPU and leave minority of stubborn cells to CPU. Details of our scheduling techniques include cell grouping, batch fetching and stubborn cell handling.

Cell Grouping. The processing order of the cells will affect the final performance heavily. A well accepted ordering strategy is to legalize large cells before small ones, because small cells can be more easily placed and have lower chance to cause the extra displacement to their neighbor cells [16], [23]. In Algorithm 1, failed cells will be put aside instead of increasing their search window size for further processing. Thus, cell processing orders will be inevitably interrupted. If, on the other hand, we strictly following the pre-determined order, our parallelism will be seriously broken. To balance the runtime and performance, we propose a grouping strategy that separates the cells into different groups by cell sizes. Group orders are strictly maintained while the process orders within a group can be changed.

Batch Fetching. The key idea of parallel legalization is to place cells simultaneously. This requires that search windows of parallelly processed cells must not overlap with each other to avoid conflicts. We employ an R-tree structure to store the search windows which allows us to build window records in $\mathcal{O}(N_{\text{batch}} \log N)$ and detect overlapping in $\mathcal{O}(\log N)$, where N_{batch} is the maximum number of cells (batch size) to be placed at the same time and N is the total number of cells.

CPU Assists Legalization of Stubborn Cells. To make proper and efficient task scheduling, we have to consider the following challenge: there will be a super large amount of available insertion intervals (grows exponentially with local cell count and target cell height) in one search window, which is extremely memory consuming. Although a queue structure is applied in CPU when dealing with large scale insertion candidates, it is inefficient to adopt the same trick in CUDA kernels, due to weak support of dynamic memory in GPU devices. Thus, we leave the target cells to the CPU legalization directly if their corresponding local cell counts exceed certain threshold. Besides the above situation, we also pass cells to CPU if GPU failed to find any insertion point within a fixed search window, because recursive calls can be more efficient on CPU.

There is also a corner case that a cell may not be able to find any possible insertion point after searching through the entire space. In this case, we have to pull up some placed cell to get more possible insertion points. We start from smaller legalized cells to preserve processing order as much as possible for attaining the legalization performance.

IV. Experiments

A. Datasets and Configurations

The proposed legalization framework is implemented using C++ and CUDA on an hybrid CPU-GPU platform, that equipped with a Intel Core i5-9300HF@2.4GHz processor and NVIDIA GeForce GTX1660Ti. We adopt OpenMP for CPU multi-threading implementation. The runtime DAC'18 [23] is evaluated on a slightly different platform that has Intel Xeon E7-4830 v2 and supports up to 20 threads. To validate the effectiveness and the efficiency of our framework, we conduct experiments on public benchmarks from ISPD2015 and IC/CAD2017 contests. For the designs in ISPD2015, 10% of the single row cells are modified to double-row cells as in [23].

B. Performance Comparison with State-of-the-art Legalizers

In the first experiment, we compare the result of our framework with a state-of-the-art legalizer proposed in [23]. Details are listed in TABLE I, where column "Design" lists all test cases in IC/CAD 2017 contest benchmarks, column "cell #" shows the total number of cells to be legalized, column "DAC'18-MGL" represents the result of [23] running multi-row global legalization only, column "DAC'18" lists the results of entire [23] flow that includes MGL and other refinement techniques, columns "AveDisp" denote the average displacement of all legalized cells using Equation (2), columns "HPWL" correspond to a commonly accepted total wire length estimator in placement subjects, and columns "Time (s)" are the overall runtime of the legalization task on given designs. From the table we can observe that when runtime speedup of CPU-based solution saturates at 8 threads, our heterogeneous offers further $2\times - 4\times$ improvement in execution time without quality loss in terms of average cell displacements and HPWL.

To demonstrate the scalability of our approach, we also conduct experiments on five large designs marked with "superblue" from ISPD 2015 contest benchmarks. As can be seen in TABLE I, the advantage of HL algorithm is still tremendous over multi-threaded CPU solutions that our method completes million-cell-size design legalization with 12.98s compared to around 30s using 8-thread CPU.

We can also observe slightly smaller average cell displacements of our method. This can be explained by the cell grouping techniques introduced in Section III-D, which ensures most cells are processed in a predetermined order. Such a scheduling strategy is especially effective on designs that contain cells with more different sizes, making our solution more preferable on challenging cases.

V. Conclusion

In this paper, we present a heterogeneous legalization framework that leverages hybrid CPU-GPU platforms. We carefully design the algorithm to catch both CPU and GPU architecture characteristics. With the help of our efficient task scheduling techniques, HL can distribute cell legalization tasks to their preferable devices. The experimental results show that our framework achieves $2\times - 4\times$ speed up without quality degradation on

TABLE I Result comparison with state-of-the-art legalizer on IC/CAD 2017 and ISPD 2105 contest benchmarks.

Design	Cell #	DAC'18-MGL [23]			DAC'18 [23]			Ours		
		AveDisp	HPWL	Time (8T)	AveDisp	HPWL	Time (8T)	AveDisp	HPWL	Time
des_perf_1	112644	0.93	6787834	7.78	0.90	6772200	10.88	1.05	7051190	**3.47**
des_perf_a_md1	108288	1.13	11210840	6.87	1.12	11205727	9.95	0.92	11335860	**2.00**
des_perf_a_md2	108288	1.46	11300149	6.42	1.38	11294562	9.37	1.32	11450460	**2.00**
des_perf_b_md1	112644	0.75	10832946	7.07	0.73	10798403	8.27	0.70	10992440	**6.85**
des_perf_b_md2	112644	0.72	10988311	6.84	0.72	10979288	9.10	0.72	11043440	**1.75**
edit_dist_1_md1	130661	0.76	20478701	5.45	0.75	20473906	9.32	0.67	20406320	**1.67**
edit_dist_a_md2	127413	0.70	25927125	6.73	0.70	25916214	9.81	0.73	25939630	**1.80**
edit_dist_a_md3	127413	0.84	27271077	8.28	0.84	27257634	11.51	0.91	27457800	**3.92**
fft_2_md2	32281	0.92	2550060	1.73	0.91	2542788	2.96	0.68	2483889	**0.45**
fft_a_md2	30625	0.64	5550997	1.92	0.63	5545913	2.33	0.65	5551121	**0.32**
fft_a_md3	30625	0.61	4827920	1.79	0.61	4824641	2.23	0.56	4832620	**0.34**
pci_bridge32_a_md1	29517	0.72	2389512	1.57	0.71	2384389	2.45	0.63	2397252	**0.58**
pci_bridge32_a_md2	29517	0.88	3015534	1.74	0.87	3009951	2.44	0.91	3042192	**0.62**
pci_bridge32_b_md1	28914	0.86	3420847	1.90	0.85	3419204	2.27	0.48	3418254	**0.62**
pci_bridge32_b_md2	28914	0.79	2969299	1.85	0.79	2968204	6.95	0.63	3005030	**0.45**
pci_bridge32_b_md3	28914	1.05	3040309	1.83	1.03	3039354	2.68	0.87	3074251	**0.45**
Average		0.86	9535091	4.36	0.85	9527024	6.41	**0.78**	9592609	**1.71**
Ratio		1.10	0.99	2.55	1.08	0.99	3.75	**1.00**	1.00	**1.00**
superblue11_a	927074	0.52	429927937	33.61	0.51	429937259	37.94	0.26	430124300	**9.52**
superblue12	1287037	0.42	392914566	42.49	0.41	392927840	49.15	0.21	392996000	**34.79**
superblue14	612583	0.53	280206549	18.99	0.53	280211344	21.75	0.31	280441500	**7.83**
superblue16_a	680869	0.47	313833508	20.63	0.46	313844245	23.79	0.12	313847900	**7.20**
superblue19	506383	0.47	207879448	19.81	0.47	207883434	20.99	0.25	208004400	**5.54**
Average		0.48	324952402	27.11	0.47	324960824	30.72	**0.23**	325082820	**12.98**
Ratio		2.09	1.00	2.09	2.06	1.00	2.37	**1.00**	1.00	**1.00**

IC/CAD 2017 and ISPD 2015 contest benchmarks, compared to state-of-the-art mixed-cell-height legalizers with CPU multi-threading. With the surpassing performance, we hope this work can motivate further research of physical design algorithms for heterogeneous computing platforms to compromise the demanding efficiency in modern chip designs. Future works include flexible task scheduler design with lower CPU-GPU communication overhead, concurrent execution of CPU and GPU tasks, memory usage optimization, etc.

ACKNOWLEDGMENT

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420). The authors would like to thank Haocheng Li from Cadence Design Systems for his valuable discussion.

REFERENCES

[1] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. CRC press, 2008.

[2] N. K. Darav, I. S. Bustany, A. Kennings, and R. Mamidi, "ICCAD-2017 CAD contest in multi-deck standard cell legalization and benchmarks," in *Proc. ICCAD*, 2017, pp. 867–871.

[3] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ISPD*, 2015, pp. 157–164.

[4] D. Hill, "Method and system for high speed detailed placement of cells within an integrated circuit design," Apr. 9 2002, uS Patent 6,370,673.

[5] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang, "NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs," in *Proc. ISPD*, 2005, pp. 236–238.

[6] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE TCAD*, vol. 33, no. 12, pp. 1914–1927, 2014.

[7] C. Huang, H. Lee, B. Lin, S. Yang, C. Chang, S. Chen, Y. Chang, T. Chen, and I. Bustany, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE TCAD*, vol. 37, no. 3, pp. 669–681, 2018.

[8] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Proc. ICCAD*, 2005, pp. 48–55.

[9] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. ASPDAC*, 2007, pp. 135–140.

[10] U. Brenner, "Bonnplace legalization: Minimizing movement by iterative augmentation," *IEEE TCAD*, vol. 32, no. 8, pp. 1215–1227, 2013.

[11] U. Brenner and J. Vygen, "Legalizing a placement with minimum total movement," *IEEE TCAD*, vol. 23, no. 12, pp. 1597–1613, 2004.

[12] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: fast legalization of standard cell circuits with minimal movement," in *Proc. ISPD*, 2008, pp. 47–53.

[13] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," *IEEE TCAD*, vol. 37, no. 6, pp. 1237–1250, 2018.

[14] J. Chen, Z. Zhu, W. Zhu, and Y.-W. Chang, "Toward optimal legalization for mixed-cell-height circuit designs," in *Proc. DAC*, 2017, pp. 52:1–52:6.

[15] X. Li, J. Chen, W. Zhu, and Y.-W. Chang, "Analytical mixed-cell-height legalization considering average and maximum movement minimization," in *Proc. ISPD*, 2019, p. 27–34.

[16] W.-K. Chow, C.-W. Pui, and E. F. Y. Young, "Legalization algorithm for multiple-row height standard cell design," in *Proc. DAC*, 2016, pp. 83:1–83:6.

[17] N. K. Darav, I. S. Bustany, A. Kennings, and L. Behjat, "A fast, robust network flow-based standard-cell legalization method for minimizing maximum movement," in *Proc. ISPD*, 2017, p. 141–148.

[18] N. K. Darav, I. S. Bustany, A. Kennings, D. Westwick, and L. Behjat, "Eh?Legalizer: A high performance standard-cell legalizer observing technology constraints," *ACM TODAES*, vol. 23, no. 4, 2018.

[19] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," in *Proc. ICCAD*, 2009, pp. 681–688.

[20] 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 *Proc. DAC*, 2019.

[21] S. Dhar and D. Z. Pan, "Gdp: Gpu accelerated detailed placement," in *Proc. HPEC*, 2018, pp. 1–7.

[22] 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*, pp. 1–1, 2020.

[23] H. Li, W.-K. Chow, G. Chen, E. F. Young, and B. Yu, "Routability-driven and fence-aware legalization for mixed-cell-height circuits," in *Proc. DAC*, 2018, pp. 1–6.

[24] K. Han, A. B. Kahng, and H. Lee, "Evaluation of beol design rule impacts using an optimal ilp-based detailed router," in *Proc. DAC*. IEEE, 2015, pp. 1–6.

[25] D. Guide, "Cuda c programming guide," *NVIDIA, July*, 2013.