

**PARALLEL 3D RECTILINEAR STEINER TREE CONSTRUCTION
BASED ON SPANNING GRAPHS****R. N. KHACHATRYAN ****National Polytechnic University of Armenia (NPUA), Armenia*

Steiner tree construction is a fundamental problem in VLSI physical design. Given a set of pins in 3D space, the objective is to connect all pins with minimum total wirelength, possibly using additional Steiner points. This problem becomes particularly important in modern 3D integrated circuits, where vertical connections are realized through Through-Silicon Vias (TSVs). Since TSVs occupy valuable silicon area, their positions must be carefully determined and tracked to avoid conflicts between different nets.

Building upon the 3D rectilinear spanning graph presented in prior work, this paper proposes a parallel algorithm for 3D rectilinear Steiner tree construction. The main contributions are as follows. First, the pair-generation process is decoupled from Kruskal's algorithm. Second, Tarjan's sequential lowest-common-ancestor (LCA) computation is replaced with the Euler tour technique combined with a range-minimum-query (RMQ) data structure, enabling fast and independent parallel LCA queries. In addition, a centroid-based tree partitioning scheme is introduced, which allows edge substitutions to be computed concurrently and independently.

The proposed algorithm achieves up to $15\times$ speedup with 16 threads compared to the sequential version while maintaining solution quality. As an application, a TSV-aware multi-net routing framework is demonstrated, where dies are modeled as layers in 3D space, and TSV positions are dynamically determined during Steiner tree construction and tracked to prevent conflicts between nets.

<https://doi.org/10.46991/PYSUA.2026.60.2.091>

MSC2020: Primary: 68W10; Secondary: 94C15.

Keywords: Steiner tree, spanning graph, parallelization, TSV routing, 3D integrated circuits, LCA, Euler tour, range minimum query.

Introduction. Steiner tree construction is a fundamental problem in VLSI physical design. Given a set of terminal points, the goal is to find a tree connecting all terminals with minimum total wirelength, potentially using additional Steiner points.

* E-mail: raf.khachatryan4@gmail.com

The problem is known to be NP-complete [1], yet in modern EDA tools the number of points can reach millions, making both solution quality and computation time critical factors. Zhou et al. [2] introduced the Rectilinear Spanning Graph (RSG), a sparse $O(n)$ -edge structure constructed in $O(n \log n)$ time by partitioning the plane into eight octagonal regions per point, guaranteed to contain at least one MST without requiring Delaunay triangulation. Zhou [3] then combined this spanning graph with Borah et al.'s edge-substitution technique to build Steiner trees in $O(n \log n)$ time with quality close to the Iterated 1-Steiner heuristic. Zhu et al. [4] generalized the spanning graph to the octilinear plane for X-architecture routing, achieving up to 15% wirelength reduction over rectilinear trees with extensibility to arbitrary λ -geometries. Long et al. [5] proposed EBOARST, which adapts the spanning graph framework to obstacle-avoiding environments through an obstacle-aware spanning graph (OASG) and enhanced edge-based refinement. Lin et al. [6] independently addressed obstacle avoidance by introducing "essential edges" in the OASG construction to improve solution quality on single-layer designs. Lin et al. [7] extended the approach to multilayer obstacle avoidance (ML-OARSMT), showing that cross-layer paths and via costs require a fundamentally different spanning graph construction than the single-layer case. Wang et al. [8] further generalized this to region-to-region connections in multilayer designs. For 3D ones, Lee et al. [9] proposed a rectilinear and octilinear SMT algorithm for multidimensional environments, though without the spanning graph framework.

Lin and Kim [10] constructed all possible multilayer RSMTs on the 3D Hanan grid for monolithic 3D ICs by enumerating topologies for small nets, targeting MIV insertion rather than heuristic construction for large instances.

Yan et al. [11] addressed timing-driven 3D Steiner tree construction using hidden Steiner-point assignment across layers with Elmore delay optimization, achieving 9.7%–16% delay improvement. Tang et al. [12] provides a comprehensive survey of Steiner tree construction and global routing algorithms for VLSI design. In the previous work [13], coarse-grained parallelization of 2D RMST construction was investigated, and in [14] the extension of RSG construction to 3D using 48 spatial regions with coarse grained parallelization was presented.

In this work, a parallel 3D Steiner tree algorithm is proposed that builds on the 3D RSG from prior work. For 3D integrated circuits, Steiner tree construction must account for vertical connections through TSVs, as modern 3D ICs stack multiple dies with TSVs providing inter-die connectivity. The problem becomes more complex, because TSVs are physical structures requiring dedicated silicon area, so TSV positions from different nets cannot overlap.

In this paper, a parallel algorithm for 3D rectilinear Steiner tree construction is proposed. The main contributions are applying Steiner tree construction in 3D space, replacing Tarjan's sequential LCA algorithm [15] with an Euler tour combined with a RMQ data structure to enable independent parallel LCA queries and parallel gain calculation, introducing centroid-based tree partitioning to allow concurrent processing of independent edge substitutions, and demonstrating a TSV-

aware multi-net routing framework that dynamically determines and tracks TSV positions to avoid conflicts between nets.

Main Algorithm. The proposed approach begins by constructing a Minimum Spanning Tree from the RSG edges using Kruskal's algorithm. Fig. 1 visualizes the MST constructed based on the spanning graph using Kruskal's algorithm. During this process, a merging binary tree is simultaneously built. When Kruskal's algorithm merges two components by adding edge (u, v) , an internal node is created in the merging tree that represents this edge. The weight stored at this internal node is the weight of edge (u, v) . The leaves of the merging tree correspond to the original input points. This merging tree has the property that the LCA of any two leaf nodes corresponds to the longest edge on the path between those two points in the MST. This property is fundamental to the Steiner point identification process.

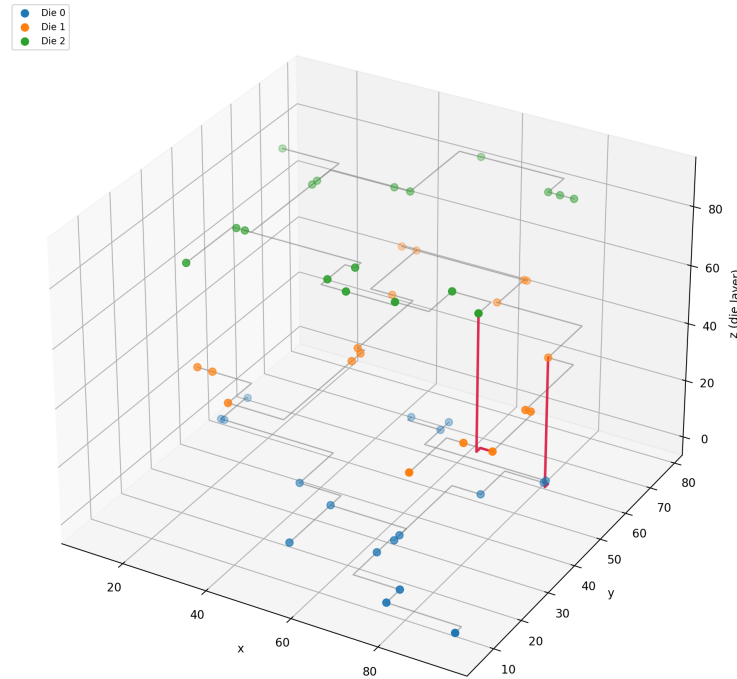


Fig. 1. The MST constructed based on the spanning graph using Kruskal's algorithm.

After building the Kruskal merging tree, a Euler Tour is constructed by performing a DFS traversal and recording each node as it is visited. The depth of each node in the traversal is also recorded. Using these depth values, a Sparse Table is built that enables $O(1)$ range minimum queries. This approach replaces Tarjan's offline LCA algorithm. While Tarjan's algorithm has $O(n)$ construction time versus the proposed $O(n \log n)$, the key advantage is that the queries are completely read-only. In the Figs. 2 and 3 is illustrated merging binary tree and corresponding Euler tour and LCA query process visualization.

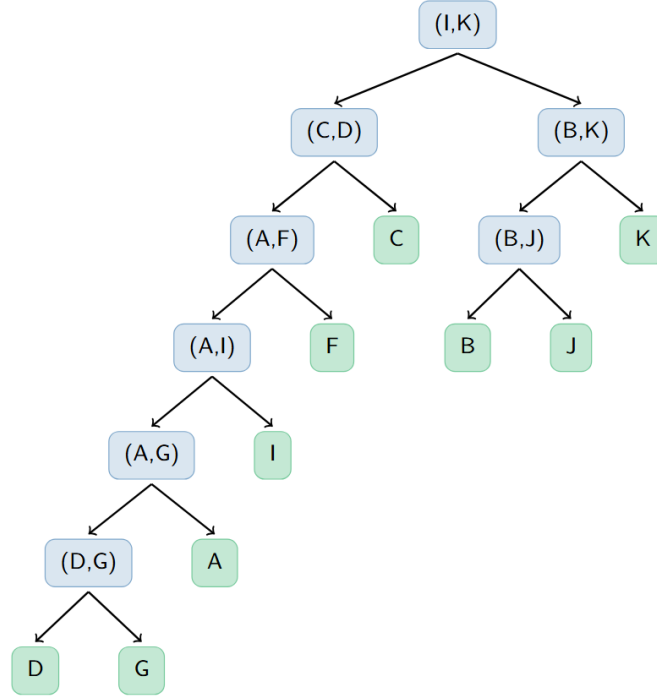


Fig. 2. The Merging Binary Tree.

Tarjan's algorithm answers queries during a DFS traversal while simultaneously modifying a union-find structure, making parallelization impossible. With Euler Tour and RMQ, the sparse table is built once and then all queries are independent lookups, enabling independent parallel query execution. Unlike Zhou's algorithm [3], where pair generation is interleaved with Kruskal's merging, it is decoupled into a separate stage.

After the sparse table is built, candidate pairs for Steiner point insertion are identified. For each MST edge (u, v) , all RSG neighbors of u and v are examined. Each such neighbor p (where $p \neq u$ and $p \neq v$) together with edge (u, v) forms a candidate pair (p, u, v) . For parallelization, the MST edges are divided among threads. Each thread iterates over its assigned MST edges, looks up the RSG neighbors of both endpoints, and collects the resulting pairs into a local vector. After all threads are complete, the results are concatenated, a parallel sort by (p, u, v) is applied, and duplicates are removed. Next, for each candidate pair (p, u, v) , the gain is evaluated independently using two $O(1)$ range minimum queries on the Sparse Table to find the bottleneck edge that would be replaced by a Steiner point insertion. The Steiner point s is placed at the rectilinear median of p , u , and v . Since all RMQ queries are read-only, this step is parallelized by dividing the pairs processing to separate threads. Each thread evaluates its pairs independently and collects positive-gain candidates. After all threads are complete, the results are merged and sorted by gain in descending order.

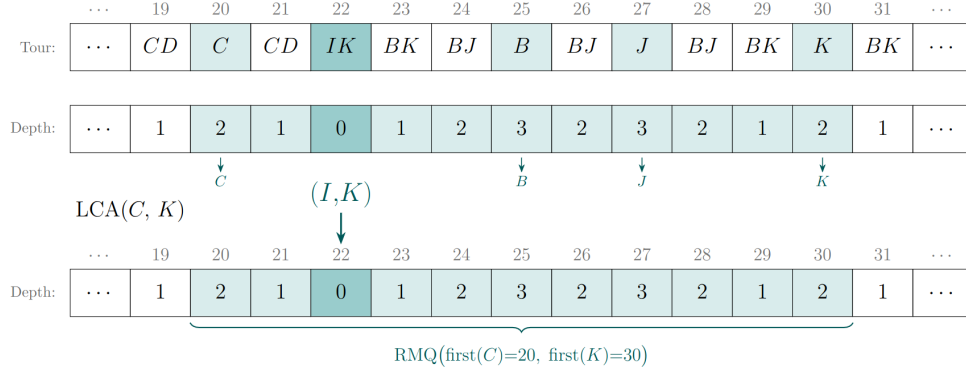


Fig. 3. LCA in merging binary tree using Euler tour and Range Minimum Query.

In the final step, the actual edge substitutions are performed. Candidates are processed in order of decreasing gain. For each candidate, it is checked whether both the original edge (u, v) and the longest edge (l_u, l_v) are still present in the active edge set. If either has been removed by a previous substitution, this candidate is skipped. If both edges are still active, the substitution is performed by removing the original edge and the longest edge, and adding the Steiner point and three new edges connecting the Steiner point to p , u , and v . For parallelization, tree partitioning is used. The centroid edge that splits the MST into two subtrees of roughly equal size is found. The candidates are then partitioned into those involving only vertices in subtree 1, those involving only vertices in subtree 2, and those that cross the partition. The first two groups are processed in parallel as they not conflict with each other. The cross-partition candidates are processed sequentially afterward. This partitioning is limited to a single split, so as to not affect the quality of the Steiner tree construction.

As an application of the proposed algorithm, it is applied to TSV-aware multi-net routing. The routing problem is defined as follows. Given a set of nets where each net has pins on different dies, the goal is to connect all pins of each net using wires and TSVs. TSV positions are not predefined. Instead, the algorithm finds TSV locations during Steiner tree construction. Two nets cannot use the same TSV position because TSVs occupy physical area on the chip. Dies are treated as layers in 3D space. The z -coordinate is the die number scaled by a TSV cost factor. This scaling makes vertical distances comparable to horizontal distances. When the Steiner tree algorithm adds a Steiner point that connects pins on different dies, that point becomes a TSV location. A simple data structure is used to track used TSV positions. After routing each net, its TSV positions are stored in a set. Before routing the next net, this set is passed to the Steiner tree algorithm. During gain calculation, if a candidate Steiner point would create a TSV at an occupied position, the nearest available position is searched. If no nearby position gives positive gain, that substitution is skipped. Smaller nets are routed first. Smaller nets have fewer options and are more likely to be blocked

by larger nets. After all nets are routed, a complete routing with no TSV conflicts is obtained. Algorithm 1 presents the high-level structure of the proposed parallel 3D Steiner tree construction algorithm.

Algorithm 1 Parallel 3D RST Construction (High-Level)

Require: Points P , RSG edges, number of threads T

Ensure: Steiner tree RST

- 1: **Step 1:** Build MST and Kruskal Merging Tree using Kruskal's algorithm
 - 2: **Step 2:** Build Euler Tour and Sparse Table RMQ on the merging tree. $O(n \log n)$
 - 3: **Step 3: Parallel Pair Generation**
 - 4: Divide MST edges among T threads
 - 5: Each thread: for each MST edge (u, v) collect RSG neighbors as candidate pairs
 - 6: Concatenate results, parallel sort, remove duplicates
 - 7: **Step 4: Parallel LCA + Gain Calculation**
 - 8: Divide pairs among T threads
 - 9: Each thread: query max edge on path using RMQ. $O(1)$ per query
 - 10: Calculate gain and Steiner point for positive-gain candidates
 - 11: Merge results and sort by gain (descending)
 - 12: **Step 5: Parallel Edge Substitution with Tree Partitioning**
 - 13: Find centroid edge that splits tree most evenly
 - 14: Partition candidates into: subtree-1-only, subtree-2-only, cross-partition
 - 15: Process subtree-1 and subtree-2 candidates in parallel
 - 16: Process cross-partition candidates sequentially
 - 17: Merge results
-

Experimental Results. With this approach, significant improvements are achieved in both quality and performance. In terms of solution quality, the proposed method achieves 12–15% improvement over the MST, compared to Zhou's 9–12% for one iteration. This represents approximately 3% additional improvement. This quality gain is attributed to the Euler Tour + RMQ data structure, which enables accurate evaluation of all potential Steiner point candidates. The read-only nature of the queries ensures consistent results regardless of processing order. In Fig. 4 is presented the time comparison of Zhou's original algorithm and the proposed parallel algorithm on the same rectilinear spanning graph set. In terms of performance, up to 15x speedup over the original sequential algorithm is achieved when using 16 threads. And in Fig. 5 is shown comparison of the improvement percentages of Zhou's and the proposed algorithm. The parallel Steiner tree algorithm was evaluated on point sets ranging from 1,000 to 100,000 points distributed across 3 to 4 dies. For TSV-aware routing, the algorithm was tested on netlists with 20 to 30 nets across 4 dies. The router successfully routes all nets with zero TSV position conflicts. For a 30-net testcase with 27 multi-die nets, the router used 28 TSVs, which equals the minimum required for connectivity. Fig. 6 shows the 3D visualization of a routed netlist with wires on each die and TSVs connecting them.

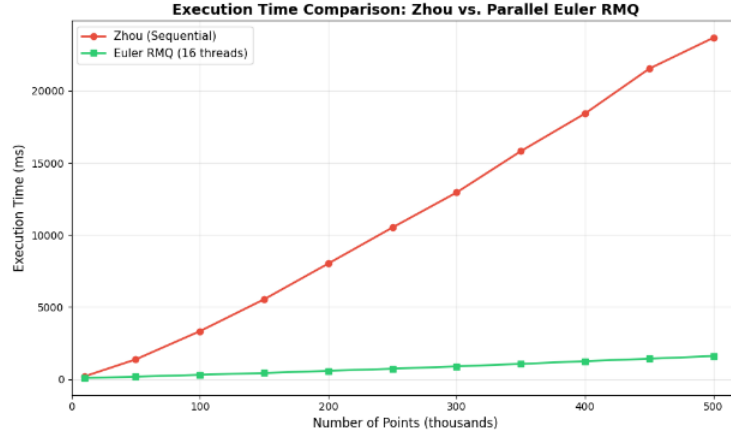


Fig. 4. Execution time comparison of Zhou's and the proposed parallel Euler RMQ algorithms.

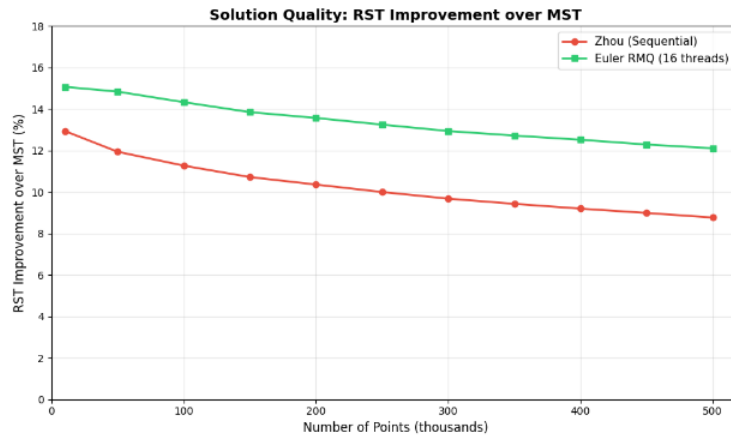


Fig. 5. Solution improvement comparison of Zhou's and the proposed parallel Euler RMQ algorithms for one iteration.

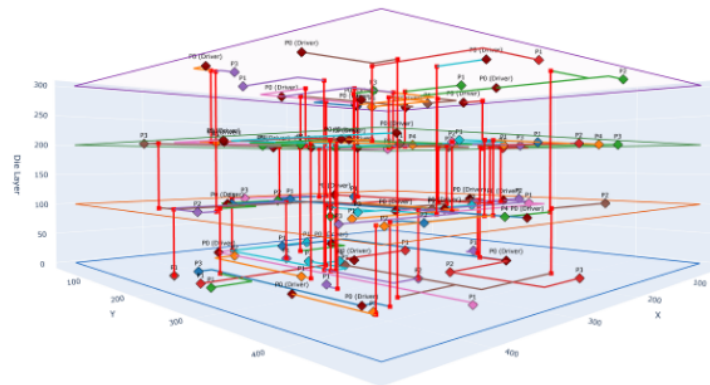


Fig. 6. 3D visualization of TSV-aware multi-net routing result.

Conclusion. A parallel 3D Rectilinear Steiner Tree construction algorithm has been proposed. Building on the 3D RSG from prior work, the main contributions are replacing Tarjan’s sequential LCA algorithm with Euler Tour and RMQ to enable parallel gain calculation, and using centroid-based tree partitioning for parallel edge substitutions. Experimental results show that the proposed algorithm achieves up to 15 times speedup with 16 threads and approximately 3 percent quality improvement over the sequential algorithm. As an application, TSV-aware multi-net routing was demonstrated, where TSV positions are determined during Steiner tree construction and tracked to avoid conflicts between nets. Future work includes extending the algorithm to handle routing obstacles and incorporating timing constraints.

Received 20.05.2026

Reviewed 31.08.2026

Accepted 09.09.2026

REFERENCES

1. Garey M.R., Johnson D.S. The Rectilinear Steiner Tree Problem is NP-Complete. *SIAM Journal on Applied Mathematics* **32** (1977), 826–834.
<https://doi.org/10.1137/0132071>
2. Zhou H., Shenoy N., Nicholls W. Efficient Minimum Spanning Tree Construction without Delaunay Triangulation. *Proc. Asia South Pacific Design Automation Conf. (ASP-DAC)* (2001), 25–30.
[https://doi.org/10.1016/S0020-0190\(01\)00232-0](https://doi.org/10.1016/S0020-0190(01)00232-0)
3. Zhou H. Efficient Steiner Tree Construction Based on Spanning Graphs. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **23** (2004), 704–710.
<https://doi.org/10.1109/TCAD.2004.826557>
4. Zhu Q., Zhou H., et al. Spanning Graph-based Nonrectilinear Steiner Tree Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **24** (2005), 1066–1075.
<https://doi.org/10.1109/TCAD.2005.850862>
5. Long J., Zhou H., Memik S.O. EBOARST: An Efficient Edge-Based Obstacle-Avoiding Rectilinear Steiner Tree Construction Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27** (2008), 2169–2182.
<https://doi.org/10.1109/TCAD.2008.2006098>
6. Lin C.-W., Chen S.-Y., et al. Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27** (2008), 643–653.
<https://doi.org/10.1109/TCAD.2008.917583>
7. Lin C.-W., Huang S.-L., et al. Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27** (2008), 2007–2016.
<https://doi.org/10.1109/TCAD.2008.2006095>

8. Wang R.-Y., et al. Efficient Multi-Layer Obstacle-Avoiding Region-to-Region Rectilinear Steiner Tree Construction. *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. USA, CA, San Francisco (2018), 1–6.
9. Lee M.C., Jan G.E., Luo C.C. An Efficient Rectilinear and Octilinear Steiner Minimal Tree Algorithm for Multidimensional Environments. *IEEE Access* **8** (2020), 48141–48150.
<https://doi.org/10.1109/ACCESS.2020.2977825>
10. Lin S.-E.D., Kim D.H. Construction of All Multilayer Monolithic Rectilinear Steiner Minimum Trees on the 3D Hanan Grid for Monolithic 3D IC Routing. *Proceedings of the International Symposium on Physical Design (ISPD)* (2019), 57–64.
<https://doi.org/10.1145/3299902.3309749>
11. Yan J.-T., Chen Z.-W., Hu D.-H. Timing-driven Steiner Tree Construction for Three-dimensional ICs. *Proceedings of the IEEE International Conference on IC Design Technology* (2008), 335–338.
<https://doi.org/10.1109/NEWCAS.2008.4606389>
12. Tang H., Liu G., et al. A Survey on Steiner Tree Construction and Global Routing for VLSI Design. *IEEE Access* **8** (2020), 68593–68622.
<https://doi.org/10.1109/ACCESS.2020.2986138>
13. Khachatryan R.N., Harutyunyan A.G. Parallelization of Rectilinear Minimum Spanning Tree Construction Algorithm. *Proceedings of ISPC “Modern Information and Electronic Technologies”*. Ukraine, Odesa (2025), 35–36.
<https://arar.sci.am/Content/427647/121-128.pdf>
14. Harutyunyan A.G., Khachatryan R.N. “Coarse-Grained Parallelization of the Spanning Graph Construction Algorithm. *Proc. of the RA NAS and NPUA, Ser. of Technical Sciences* **78** (2025), 121–128.
<https://doi.org/10.53297/0002306X-2025.v78.1-121>
15. Cormen T.H., Leiserson C.E., Rivest R.L. *Introduction to Algorithms*. Cambridge, MA, MIT Press (1989).
<https://doi.org/10.5555/1614191>

Ռ. Ն. ԽԱՉԱՏՐՅԱՆ

ԿՄԱԽԶԱՅԻՆ ԳՐԱՖՆԵՐԻ ՎՐԱ ՆԻՄՆՎԱԾ 3D ՈՒՂՂԱՆԿՅՈՒՆ
ՇՏԵՅՆԵՐԻ ԾԱՌԻ ԶՈՒԳԱՆԵՌ ԿԱՌՈՒՅՈՒՄ

Շտեյների ծառի կառուցումը ԳՄԻՍ-ի ֆիզիկական նախագծման հիմնարար խնդիր է: 3D Տարածության հանգույցների բազմության դեպքում նպատակն է բոլոր հանգույցները միացնել նվազագույն ընդհանուր մեքաղալարի երկարությամբ՝ հնարավորության դեպքում օգտագործելով լրացուցիչ Շտեյների կետեր: Այս խնդիրը հափկապես կարևոր է դառնում ժամանակակից 3D ինտեգրալ սխեմաներում, որտեղ ուղղահայաց կապերը իրականացվում են TSV-ների միջոցով: Քանի որ TSV-ները զբաղեցնում են թանկարժեք սիլիկոնային փարածություն, նրանց դիրքերը պետք է խնամքով որոշվեն և հեղուկափոխվեն՝ փարեր շղթաների միջև կոնֆլիկտներից խուսափելու համար:

Տիմնվելով նախկին աշխարհում առաջարկված 3D ուղղանկյուն կմախքային գրաֆի վրա՝ այս աշխարհում ներկայացնում է 3D ուղղանկյուն Շտեյների ծառի զուգահեռ կառուցման ալգորիթմ: Աշխարհի հիմնական ներդրումներն են հետևյալը: Առաջինը՝ զույգերի ձևավորման գործընթացն առանձնացվել է Կրուսկալի ալգորիթմից: Երկրորդը՝ Տարժանի հաջորդական ամենամոտ ընդհանուր նախնու (LCA) ալգորիթմը փոխարինվել է Էյլերի շրջագայության տեխնիկայով՝ զուգակցված փրույթի նվազագույնի հարցման (RMQ) տվյալների կառուցվածքի հետ, ինչը թույլ է տալիս արագ և անկախ զուգահեռ LCA հարցումներ: Բացի այդ, ներդրվել է կենտրոնիկ-հիմնված ծառի բաժանման մեթոդ, որը հնարավորություն է տալիս կողերի փոխարինումները հաշվարկել զուգահեռ և անկախ:

Առաջարկված ալգորիթմը հասնում է մինչև 15-ապարիկ արագացման 16 հոսքերով՝ համեմատած հաջորդական փարբերակի հետ՝ պահպանելով լուծման որակը: Որպես կիրառություն ցուցադրվում է TSV-երի դիրքերը հաշվի առնող բազմաշղթայական ծրագրում, որտեղ die-երը դիտարկվում են որպես 3D փարածության շերտեր, իսկ TSV-երի դիրքերը դինամիկորեն որոշվում են Շտեյների ծառի կառուցման ընթացքում և հետևվում՝ շղթաների միջև կոնֆլիկտները կանխելու համար:

Р. Н. ХАЧАТРЯН

ПАРАЛЛЕЛЬНОЕ ПОСТРОЕНИЕ 3D ПРЯМОЛИНЕЙНОГО ДЕРЕВА ШТЕЙНЕРА НА ОСНОВЕ ОСТОВНЫХ ГРАФОВ

Построение дерева Штейнера является фундаментальной задачей физического проектирования СБИС. При наличии множества контактов (pins) в 3-мерном пространстве целью является соединение всех контактов проводниками минимальной суммарной длины, возможно, с использованием дополнительных точек Штейнера. Эта задача особенно важна в современных трехмерных интегральных схемах, где вертикальные соединения реализуются через сквозные кремниевые переходные отверстия (СПО). Поскольку СПО занимают ценную площадь кремния, их позиции должны тщательно определяться и отслеживаться для предотвращения конфликтов между различными цепями.

Опираясь на 3D ортогональный остовный граф, предложенный в предыдущей работе, в данной статье представлен параллельный алгоритм построения 3D ортогонального дерева Штейнера. Основная новизна работы заключается в следующем. Во-первых, процесс генерации пар отделен от алгоритма Крускала. Во-вторых, последовательный алгоритм поиска ближайшего общего предка Тарьяна (LCA) заменен на технику Эйлерова

тура в сочетании со структурой данных RMQ (поиск минимума в диапазоне), что позволяет выполнять быстрые и независимые параллельные запросы LCA. Кроме того, введен метод разбиения дерева на основе центроидов, который позволяет параллельно и независимо вычислять замены ребер.

Предложенный алгоритм достигает ускорения до 15 раз при использовании 16 потоков по сравнению с последовательной версией, сохраняя при этом качество решения. В качестве приложения демонстрируется система многопутевой трассировки с учетом СПО, в которой кристаллы (dies) рассматриваются как слои в 3D-пространстве, а позиции СПО динамически определяются в процессе построения дерева Штейнера и отслеживаются для предотвращения конфликтов между цепями.