ISSN 0002-306X. Proc. of the RA NAS and NPUA Ser. of tech. sc. 2025. V. LXXVIII, N1

UDC 004.832

AUTOMATAION AND CONTROL SYSTEMS

DOI: 10.53297/0002306X-2025.v78.1-121

A.G. HARUTYUNYAN, R. N. KHACHATRYAN

COARSE-GRAINED PARALLELIZATION OF THE SPANNING GRAPH CONSTRUCTION ALGORITHM

The spanning graph construction problem is to create a sparse graph over a given set of points so that the graph contains at least one minimum spanning tree under a specified distance metric. This problem is the basis for many algorithms like rectilinear minimum spanning tree construction, efficient Steiner tree construction, obstacle aware Steiner tree construction, etc. These algorithms are used in many fields of computer science, especially in VLSI routing. As many of these algorithms are NP-complete problems and some of them use the spanning graph construction approach, parallelization of the spanning graph construction algorithm will optimize other problems as well. In this paper, we present the coarse-grained parallelization approach of the spanning graph construction algorithm. The proposed algorithm, compared to the existing sequential algorithm, achieves an average performance improvement of about 40-60% and keeps the correctness of the original algorithm.

Keywords: minimum spanning tree, rectilinear distance, parallelization, coarse-grained parallelization, VLSI Design, Sweep-line algorithm.

Introduction. The concept of a spanning graph is defined as a sparse graph that contains at least one minimum spanning tree. Unlike a complete graph, which contains all possible edges, a spanning graph includes only a subset of edges but still guarantees that a minimum spanning tree can be extracted from it. This makes spanning graphs useful for reducing computational complexity in minimum spanning tree (MST) related problems.

The concept of a spanning graph was introduced by Zhou,et al [1]. This idea was applied in their algorithm for constructing rectilinear minimum spanning trees (RMST), where they avoided Delaunay triangulation and used a sweep-line technique. This algorithm was used in many later works. Zhou extended this idea in [2] by using spanning graphs to construct rectilinear Steiner trees. Later, the method was applied to non-rectilinear geometries [3], showing the flexibility of the approach. It was also adapted for obstacle-aware routing, where the obstacle-aware Steiner tree was constructed on the same spanning graph idea [4,5]. The minimum spanning tree (MST) problem is one of the primary problems in computer science,

with applications in network design, geometric computations, and VLSI routing. The construction of a rectilinear minimum spanning tree (RMST) is a key problem in VLSI design, especially in routing phases, where it is used as a foundation for constructing Steiner trees and estimating wirelength in global routing.

The goal of constructing an RMST in VLSI is to connect a set of points (called pins) on a chip using rectilinear (Manhattan) lines so that the total length is minimal. Sometimes it is better to add extra points (called Steiner points) to get a shorter result. The final tree with these extra points is called a Steiner tree.

Zhou's proposed spanning graph construction approach is the basis for problems like RMST construction, efficient Steiner Tree construction, and others. Many of these problems are NP-complete, and improving the performance of this method leads to more efficient computation in algorithms that use it.

However, Zhou's approach is purely sequential. As VLSI design scales and multi-core processors become standard, there is room for parallelization. In this paper, a coarse-grained parallelization of Zhou's spanning graph construction algorithm is proposed. Coarse-grained parallelization refers to dividing a program into large, mostly independent tasks that can run in parallel. These tasks, such as separate threads or subroutines, are executed on different processor cores. Our method keeps the correctness and structure of the original algorithm but improves performance by solving independent subproblems in parallel. This makes it more practical for modern large-scale VLSI designs.

Problem description. Many algorithms have been introduced for computing minimum spanning trees (MST) in parallel, such as the Filter-Kruskal algorithm [6], the parallel implementation of Borůvka's algorithm [7], and others [8–10]. However, most of these do not primarily address the MST problem under the rectilinear (Manhattan) metric. This metric is especially important in the VLSI domain, where interconnections among circuits are made using rectilinear lines.

A well-known algorithm that addresses this metric is the RMST construction algorithm proposed by Zhou. This method is important because it was later used for constructing efficient Steiner trees [2]. The work combines Zhou's spanning graph construction approach with the edge substitution heuristic, resulting in a Steiner tree construction algorithm with $O(nlogn)$ runtime complexity and simpler implementation. Later, in [3], the method was extended to non-rectilinear interconnectors, which allow 45-degree connections and can significantly reduce wire length. In that work, two algorithms were introduced. The first algorithm, called OST-E, is based on edge substitution and the second algorithm, called OST-T, is based on triple contraction. Both algorithms reuse Zhou's spanning graph construction method, adapted for octilinear geometry. These algorithms maintain $O(nlogn)$ runtime.

Zhou's spanning graph approach was also extended in [4] to support obstacle-avoiding Steiner tree construction. By constructing an obstacle-avoiding spanning graph (OASG) and selecting only valid edges, the method ensures obstacle-free routing. In [5], this idea was improved in the EBOARST algorithm, which handles obstacles during the edge generation process. Instead of filtering after graph construction, EBOARST avoids adding unnecessary edges, making the algorithm more efficient. These works show that Zhou's spanning graph construction algorithm is used as a core part in many NP-complete problems.

In this paper, we present a coarse-grained parallelization strategy for Zhou's proposed sequential algorithm. Our method focuses only on the spanning graph construction step, which is reused in many problems. The constructed graph is then passed to Kruskal's algorithm to compute the MST.

The proposed solution. Zhou et, al [1] proposed rectilinear minimum spanning tree (RMST) construction algorithm, which is working in $O(nlogn)$ time. The algorithm avoids using Delaunay triangulation, as it is not well suitable for rectilinear metric. Instead of Delaunay triangulation they use sweeping line technique and defined regions across each point from R1-R8. Using regions the algorithm identified the nearest neighbors for each point.

For each point p, the algorithm searches in his active set to find points for which p lies in their corresponding R1, R2 or R3, R4 octants. Then, p is connected to the nearest point in each of these regions, keeping the connections optimal. At the end of this phase, for each point, the algorithm removes the found points from the active set for future queries. This process has been carried out in two phases by a sweeping line by $x + y$ and by $x - y$. After these two phases we have a rectilinear spanning graph (RSG) with $O(n)$ edges, which is then passed to Kruskal's algorithm to efficiently compute the MST.

While the described algorithm has quite good performance characteristics, today's problems require work on big data sets. To address this, we designed a coarse-grained parallelization strategy for RMST construction algorithm. The two main parts of coarse-grained parallelization strategy are independent sweep lines by $x + y$ and $x - y$ line functions and nearest neighbor searches in corresponding regions. These two phases are independent of each other as they have their own active sets and share only a graph representing edge list, to which the only applied operation is insertion. The independent sweep line operations by $x + y$ and $x - y$ line functions are illustrated in Fig. 1.

Based on this, we assign two independent sweep line operations to corresponding separate threads by passing the edge list. To avoid unnecessary synchronization across threads we create separate local data structures for each

thread and pass on to functions. One of the advantages of this parallelization approach is that data across threads or processes do not need to be synchronized. And as the data structures are linked lists representing graph, we can easily merge them in $O(1)$ time complexity, as we will do only one pointer operation when these two processes are finished. After the two phases, the final data structures are merged while maintaining consistency. The proposed algorithm, Parallelized Rectilinear Spanning Graph (PRSG), is shown in Fig. 2.

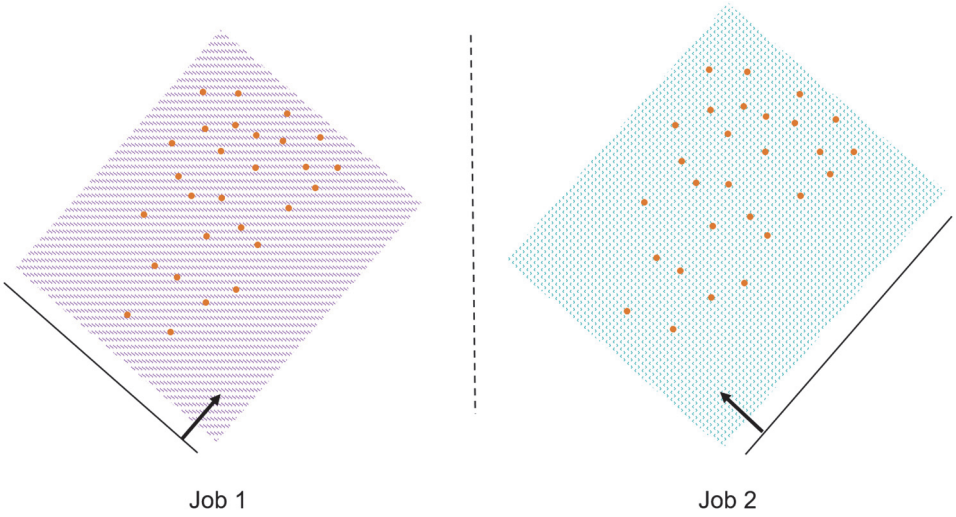

Fig. 1. The sweep line jobs parallel computation

The second part of our coarse-grained parallelization strategy is the nearest neighbor search in each of the sweeping line phases. For the $x + y$ phase, these regions are R1 and R2 and for the $x - y$ phase, R3 and R4 respectively. In Fig. 3, the regions for $x + y$ sweep line phase are illustrated.

Since these regions have separate active sets and the points in active sets do not overlap, the computation can be parallelized to run independently. The same idea is applied in this part too. As each thread works with its own active set, and these data structures are not overlapping with coordinates, again we pass edge lists as local data structures for each process. By this, we avoid synchronization across threads and carry out the merging process efficiently in $O(1)$ time.

Algorithm Parallelized Rectilinear Spanning Graph (PRSG)

1. In thread T1: Sort points by (x + y) for regions R1, R2
2. In thread T2: Sort points by (x - y) for regions R3, R4
3. Initialize two active sets for each thread.
4. Initialize local edge lists for each thread.
5. Parallel processing:

a. In thread T1:
    i. For each point p in sorted (x + y) order:
        1. Create a separate thread T1_R1:
            a. Query active set for points s where p lies in their R1 region.
            b. Find the nearest point in the queried subset.
            c. Add edge (p, the nearest point) to the local edge list.
            d. Remove s from the active set.
            e. Add p to the active set for future queries.
        2. Create a separate thread T1_R2:
            a. Query active set for points s where p lies in their R2 region.
            b. Find the nearest point in the queried subset.
            c. Add edge (p, the nearest point) to the local edge list.
            d. Remove s from the active set.
            e. Add p to the active set for future queries.
        3. Merge the local edge lists.
b. In thread T2:
    i. (Symmetric logic for regions R3 and R4)
6. Merge the local edge lists.

Fig. 2. Algorithm Parallelized Rectilinear Spanning Graph (PRSG)

By applying this coarse-grained parallelization strategy, we outperformed the sequential algorithm by 40-60 percent depending on the data size making it useful for huge data sets especially routing algorithms where circuit design sizes have millions of pins. Since this is a coarse-grained approach, it scales well for larger datasets but not for a high number of threads. The required number of threads or tasks for this approach is 6.

For the future work, we plan to extend this method to higher dimensions, such as 3D space, where parallel sweeping and neighbor searches can be applied across more dimensions and threads.

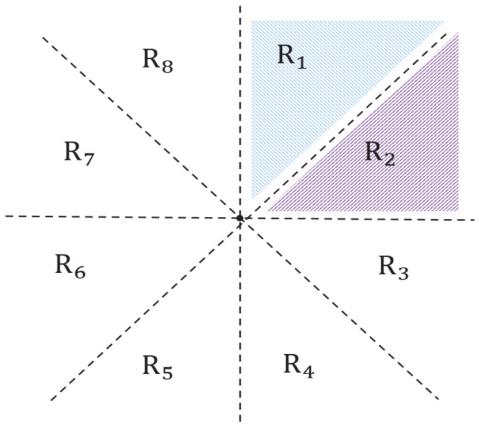

Fig. 3. The Nearest neighbor search jobs parallel computation

Experimental results. The program was implemented in C++ language and compared with the original sequential algorithm. Our experiments show that the parallelized spanning graph construction algorithm outperforms the sequential one on average by 40.32 percent for 10.000-100.000 points and 62.01 percent, for 100.000-600.000-point data set. The performance measurement plots are shown in Fig. 4 and 5. All experiments were conducted on a Linux server running Alma Linux 8.4 (kernel 4.18.0-425.3.1.el8.x86_64) with an AMD EPYC 9754 processor (32 cores, 1 thread per core), 255 GiB of RAM, and g++ 8.4.1 compiler using the C++17 standard.

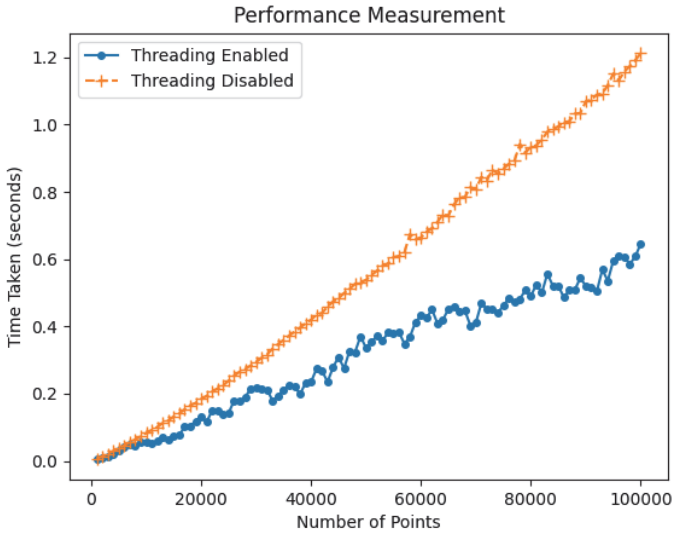

Fig. 4. Performance measurements of parallelized and sequential RSG algorithm. First data set

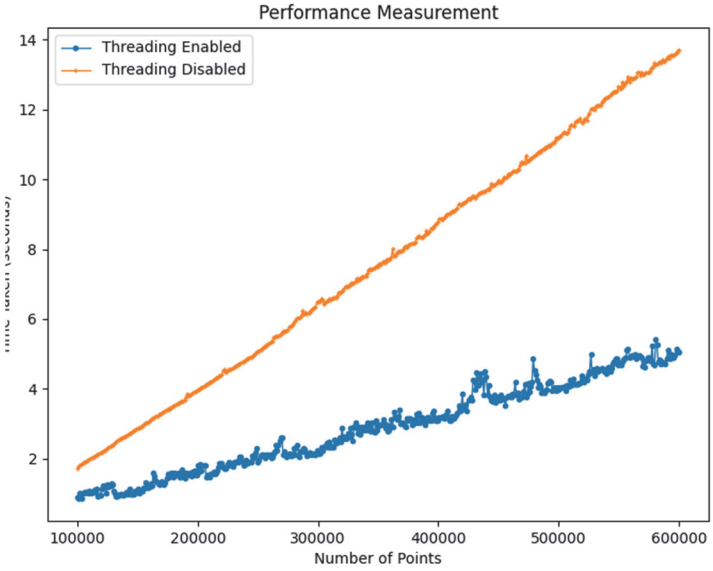

Fig. 5. Performance measurements of parallelized and sequential RSG algorithm. Second data set

Conclusion. In this paper, we present the parallelized version of the Rectilinear Spanning Graph (RSG) construction algorithm, which is used as a core algorithm in problems like RMST construction, efficient Steiner tree construction, and obstacle-avoiding Steiner tree construction. Our approach outperforms the original sequential algorithm by 40-60% using 6 threads, which corresponds to about 23–43% parallelization efficiency. This efficiency is not very high for small data sets because coarse-grained parallelization does not try to scale with the number of threads. It only parallelizes the parts of the algorithm that are independent. The main advantage is that the implementation is simple, assign independent sweep-line and region-finding jobs to threads, split the edge lists for each thread, and merge edge lists in $O(1)$ time. These changes do not modify the core algorithm structure, so the method is easy to integrate while still giving a useful performance boost for large datasets. We parallelized sweep-line operations and nearest-point connection processes, avoiding shared resource synchronization by using independent data structures. This algorithm uses only 6 threads instead of other parallelization algorithms for graphs instead other MST parallelization algorithms which require huge CPU or GPU computational power. For the future work we want to extend this algorithm for RMST construction algorithm in 3D space and 3D Steiner tree construction based on that, also do fine-grained parallelization of this algorithm.

REFERENCES

1. **Hai Zhou, Narendra Shenoy, William Nicholls.** Efficient Minimum Spanning Tree Construction without Delaunay Triangulation // Proceeding.- P. 192-197.
2. **Hai Zhou.** Efficient Steiner Tree Construction Based on Spanning Graphs // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.- May 2004. Vol. 23, no. 5.- P. 704–710.
3. Spanning Graph-Based Nonrectilinear Steiner Tree Algorithms / **Qi Zhu, Hai Zhou, Tong Jing, et. al** // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.- 2005.- Vol. 24, no. 7.- P. 1066–1075.
4. **Yung-Tai Chang, Ya-Wen Tsai, Jun-Cheng Chi, Mely Chen Chi.** Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction // Proceedings of the 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT). 2008. P. 35–38.
5. **Jieyi Long, Hai Zhou, Seda Ogrenci Memik.** EBOARST: An Efficient Edge-Based Obstacle-Avoiding Rectilinear Steiner Tree Construction Algorithm // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.- 2008.- Vol. 27, no. 12.- P. 2169–2182.
6. **Osipov, Vitaly, Peter Sanders, and Johannes Singler.** The Filter-Kruskal Minimum Spanning Tree Algorithm // Proceedings of the Meeting on Algorithm Engineering and Experiments.- 2009.- P. 52–61.
7. **Sun Chung, A. Condon.** Parallel Implementation of Boruvka's Minimum Spanning Tree Algorithm // Proceedings of the International Parallel Processing Symposium.- 15 April 1996.
8. **Vasconcellos J.F. de Alencar, Cáceres E.N., Mongelli H., Song S.W.** A Parallel Algorithm for Minimum Spanning Tree on GPU // Proceedings of the 2017 International Symposium on Computer Architecture and High-Performance Computing Workshops (SBAC-PADW).- Campina- Brazil, 2017.- P. 67–72.
9. **Vasconcellos J.F. de Alencar, Cáceres E.N., Mongelli H., Song S.W.** A New Efficient Parallel Algorithm for Minimum Spanning Tree // Proceedings of the 2018 30th International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD).- Lyon, France.- 2018.- P. 107–114.
10. **Alex Fallin, Andres Gonzalez, Jarim Seo, Martin Burtscher.** A High-Performance MST Implementation for GPUs // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23).- 11 November 2023.- Article No. 77.- P. 1–13.

National Polytechnic University of Armenia. The material is received 30.07.2025.

Ա.Գ. ՀԱՐՈՒԹՅՈՒՆՅԱՆ, Ռ.Ն. ԽԱՉԱՏՐՅԱՆ

ԿՄԱԽՔԱՅԻՆ ԳՐԱՖԻ ԿԱՌՈՒՑՄԱՆ ԱԼԳՈՐԻԹՄԻ ԽՈՇՈՐԱՀԱՏԻԿ ԶՈՒԳԱՀԵՌԱՑՈՒՄԸ

Տրված կետերի բազմության վրա որոշված մետրիկայով նոսր գրաֆ կառուցելու խնդիրը, որը պարունակում է առնվազն մեկ նվազագույն կմախքային ծառ, կոչվում է կմախքային գրաֆի կառուցման խնդիր: Այս խնդիրը հիմք է հանդիսանում մի շարք ալգորիթմների համար, ինչպիսիք են ուղղանկյուն նվազագույն կմախքային ծառի կառուցումը, Սթայների ծառի արդյունավետ կառուցումը, խոչընդոտներից խուսափող Սթայների ծառի կառուցումը և այլն: Այս ալգորիթմները կիրառվում են համակարգչային գիտության բազմաթիվ ոլորտներում, հատկապես՝ ծրագծման ալգորիթմներում: Քանի որ այս խնդիրներից շատերը NP-լրիվ են, և դրանցից մի քանիսը օգտագործում են կմախքային գրաֆի կառուցման մոտեցումը, ապա այդ պրոցեսի զուգահեռացումը կարող է օպտիմալացնել նաև մյուս խնդիրները: Ներկայացվում է կմախքային գրաֆի կառուցման ալգորիթմի բարձր մակարդակի զուգահեռացման մոտեցումը: Առաջարկվող ալգորիթմը, համեմատած առկա հաջորդական տարբերակի հետ, ապահովում է միջինում մոտավորապես 40-60% արտադրողականության աճ:

Առանցքային բառեր. նվազագույն կմախքային ծառ, ուղղանկյուն հեռավորություն, զուգահեռացում, գերմեծ ինտեգրալ սխեմաների նախագծում, սահող գծի ալգորիթմ:

А.Г. АРУТЮНЯН, Р.Н. ХАЧАТРЯН

КРУПНОЗЕРНИСТАЯ ПАРАЛЛЕЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ КАРКАСНОГО ГРАФА

Задача построения каркасного графа заключается в создании разреженного графа на заданном множестве точек таким образом, чтобы граф содержал хотя бы одно минимальное остовное дерево в соответствии с заданной метрикой расстояния. Эта задача лежит в основе множества алгоритмов, таких как построение прямоугольного минимального остовного дерева, эффективное построение дерева Штайнера, построение дерева Штайнера с учетом препятствий и др. Эти алгоритмы применяются во многих областях информатики, особенно в алгоритмах трассировки в проектировании VLSI. Поскольку многие из этих задач являются NP-полными и некоторые из них используют подход построения каркасного графа, параллелизация этого этапа может также оптимизировать и другие задачи. В данной работе представлен подход к высокоуровневой параллелизации алгоритма построения каркасного графа. Предлагаемый алгоритм, по сравнению с существующим последовательным вариантом, обеспечивает в среднем около 40…60% прироста производительности.

***Ключевые слова*:** минимальное остовное дерево, прямолинейное расстояние, распараллеливание, крупнозернистый параллелизм, проектирование сверхбольших интегральных схем, алгоритм скользящей линии.