A Study on Applying Parallelism for Construction of Steiner Tree Algorithms in VLSI Design

Shyamala G.
BMS College of Engineering
Bangalore, India

Latha N. R.
BMS College of Engineering
Bangalore, India

ABSTRACT

We present a survey of the different approaches that can be parallelized and also the parallel algorithms available today with special concern to Rectilinear steiner tree for VLSI Design and their appropriateness for high-performance computing. Thus, we review the parallel algorithms for solving the Stiener tree problem as it is of great importance for very large scale integration routing and wire length estimation. As the steiner problem in general is NP-hard, it is difficult to develop a polynomial-time algorithm to solve the problem exactly. This is why the most of research has looked at finding efficient heuristic algorithms. Additionally, many authors focused their work on utilizing the ever-increasing computational power and developed many parallel methods for solving the problem. Hence we are able to obtain better results in less time than ever before.The study shows that the accessibility of multi-core CPUs has given new impulse to the shared memory parallel programming approach., Hybrid parallel programming is the current way of harnessing the capabilities of computer clusters with multi-core nodes. On the other hand, high performance heterogeneous programming is found to be an increasingly well accepted paradigm, as a result of the availability of multi-core CPUs and GPUs systems. The use of open industry standards like OpenMP, MPI, or OpenCL, as opposed to proprietary solutions, seems to be the way to categorize and extend the use of parallel programming models. Here, we present a survey of the parallel methods for solving the stiener tree problem specifically for VLSI design

General Terms

Multicore Architecture, Parallel computing, VLSI

Keywords

RSMT, OARSMT, Multicore Architecture

1. INTRODUCTION

VLSI in Electronic design automation is a process of placing hundreds of thousands of electronic components on one chip. The efficient algorithms for global routing process in VLSI is in demand as the number of logic circuts and the memory capacity increased rapidly. Good positioning of the components is essential, because this can gradually reduce the power and lower the heating of the chip, in turn making the chips smaller and lower the production costs. The optimality criteria that must be examined include minimizing the total wire length, minimizing the total area, minimizing the cost of the chip. Most algorithms focus on minimizing the wire length. In the VLSI physical design global routing phase the appropriate locations for macro cells are determined. Macro cells are the logical units that perform the desired functions of the chip. Then the cells need to be assigned to a certain rectangular area and connected by wires which usually run rectilinearly on the chips. Hence, rectilinear Steiner trees are perfectly suited for solving this problem.

1.1 Rectilinear Steiner tree

One of the important purpose of Steiner tree in electronic design automation is in placement and routing stage of VLSI design process. The routing is categorized into two phases called global routing and detailed routing. In Global routing blocks are connected but does not consider the details of each wire and pin, whereas in detailed routing point-to-point connections is made between pins on each block is completed. For given set of input points, the Steiner tree problem (STP) is to find a minimum-length tree that connects all the input points, and new points are added to minimize the length of the tree. The new points that are added are called Steiner points. One of the types of Steiner tree problem is the Rectilinear Steiner tree that considers rectilinear or Manhattan distance between a pair of points. The rectilinear distance d between two points p_1 and p_2 is defined as: d $(p_1, p_2)=|x_1 – x_2| + |y_1 – y_2|$, where (x_i, y_i) are the coordinates of p_i. M.Hanan [1] is the first to have reduced the RST problem to the Graph Steiner Problem (GST) problem. Hanan's theorem proves that for any instance, an optimal RST exists in which every Steiner point lies at the intersection of two rectilinear lines that contain terminals. Hanan's theorem states that by depiction of a graph called Hanan Grid graph, an optimal RST may surely be obtained. F.K.Hwang[2] proves that the ratio of the cost of the rectilinear minimum spanning tree (RMST) to that of an optimum RST is no greater than 3/2. Therefore the rectilinear MST is a suitable starting point for deriving low cost RST. Ho,Vijayan and CK Wong [3] presented a new approach to construct an RST of a given set of points in the plane starting from a Minimum Spanning tree. The total minimum wire length of a RST is referred as the cost of RST. A heuristic approach is proposed that finds layouts for the edges of the MST so as to maximize the overlaps between the layouts and thus minimizing the cost (i.e. wire length) of the Rectilinear Steiner Tree

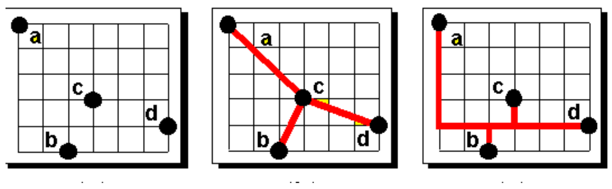

Fig 1. Rectilinear Steiner tree Construction

The original RSMT problem assume no obstacles in the routing region. In today' s VLSI design there can be many routing obstacles like macro cells, IP blocks and prerouted nets. Therefore, The RSMT problem with blockages, called OARSMT problem is widely studied. Let $P=\{p_1,p_2,p_3…p_n\}$ be the set of pins for m pin met. Let $B=\{b_1,b_2,b_3…b_k\}$ be a set of rectangular blockages. Let $V=\{v_1,v_2,v_3…v_m\}=P$ U {Corners in B} as the vertex set in the problem where each v_i has coordinates (x_i,y_i). The blockages which are rectangular has 4 corners, we have $n<=m+4k$.The rectilinear distance between

v_i & v_j is given as $|x_i-x_j|+|y_i-y_j|$. A OARSMT connects all pins through some extra points to achive a minimal total length, avoiding the intersection with any blockage in the design.

1.2 Obstacle Avoiding Rectilinear Steiner Tree

Most recent OARSMT heuristics use routing graphs to handle obstacles. Those routing graphs can be categorized into rectilinear graph and spanning graph. A rectilinear graph uses rectilinear edges to connect pin-vertices, obstacle corners, and other Steiner point candidates, while a spanning graph only contains pin vertices and obstacle corners. Compared with a spanning graph, a rectilinear graph usually contains better solutions, i.e., high effectiveness, but its space complexity is often (n2). Compared with a rectilinear graph, a spanning graph usually takes much smaller space, higher efficiency, while its solution quality is limited since it does not contain additional Steiner point candidates except obstacle corners.. To sum up, there exists a tradeoff between the effectiveness of rectilinear graphs and the efficiency of spanning graphs. For the OARSMT construction, it is desirable to balance the tradeoff, and to develop an excellent algorithm with the best practical performance in both the wirelength and the runtime.

1.3 Introduction to Multicore and Parallel computing

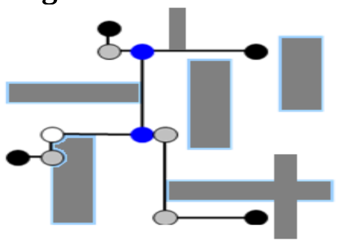

Fig 2 Obstacle Avoiding Rectilinear steiner tree

With increase in computer technology General-purpose multicore processors are being broadly accepted in all areas of the industry, including signal processing and embedded space, as the need for more performance efficiency for general-purpose applications has increased. Parallel processing increases performance considerably by adding more parallel resources while maintaining manageable power characteristics. The implementations of multicore processors are plenty and diverse. Designs range from usual multiprocessor machines to designs that consist of a many programmable arithmetic logic units (ALUs).

To develop efficient multicore algorithms, one must understand in depth multiple algorithmic and architectural aspects. The list of characteristics to develop a parallel algorithm for multicore architecture includes the following:

1. identifying the available and required dimensions of parallelism,
2. mapping parallel threads of activity to a potentially large pool of processing and functional units,
3. using simple processing cores with limited functionalities,
4. coping up with the limited on-chip memory per core,
5. coping up with the limited off-chip memory bandwidth (when compared to the rich resources available on chip)
6. performance on multicore systems:

In addition to the above features, according to [20] there is a need to consider other multicore issues which play very important role in parallel programming

Number of processing cores: Many existing systems have two to eight cores integrated on a single chip. Cores typically support features such as simultaneous multithreading (SMT) or hardware multithreading, which allow for greater parallelism and throughput. In recent designs, we have up to hundred cores on a single chiplike GPGPU.

Caching and memory bandwidth: Memory speeds have been increasing at a much slower rate than processor capacity. Memory bandwidth and latency are important performance criteria for several scientific and engineering applications. Caching severely affect the efficiency of algorithms even on single processor systems. In multicore systems, this will be even more important due to extra bandwidth constraints.

Synchronization: Implementing algorithms using multiple processing cores will require synchronization between the cores from time to time, which is an expensive operation in shared memory architectures.

Multi-core CPUs have become common, as they are widely used for high performance computations and also in consumer electronics. The idea of using the graphics processor for general purpose computation has also become popular, since this approach has yielded incredible performance when applied to suitable problems. Such increase of parallelism(multiple threads on a CPU or GPU) and heterogeneity (simultaneous use of a CPU and GPU) has succeeded in greatly improving the efficiency and performance of many traditional computation-intensive workloads, provided that their parallel or heterogeneous implementations are well analyzed. There are still some problems that demand fast computation but for whom efficient parallel or heterogeneous implementations have yet to be identified like RSMT or OARSMT construction VLSI design.Programming languages have been developed to utilize the cores available in multicore ot GPU. The CUDA language for programming GPGPU is designed to work with only NVIDIA’s GPUs, whereas OpenCL can be used with a different manufacturers of multi-core CPU and GPU devices, including NVIDIA’s GPUs. Despite CUDA’s hardware restriction, OpenCL share many similar syntax and other characteristics with CUDA.. The introduction of new parallel programming interfaces for general purpose computations, such as Computer Unified Device Architecture (CUDA), Stream SDK and OpenCL, have made GPUs powerful and attractive choice for developing high-performance numerical, scientific computation and solving practice engineering problems. However, programming on GPUs remains a challenging problem. The reason is that many contemporary GPUs exhibit complex memory organization with multiple low latency on-chip memories in addition to the off-chip memory. The access latencies and the optimal access patterns of each of the memories is significantly different, posing a significant challenge to develop techniques that optimally utilize the various memories to tolerate the latency and improve the memory thoughtful. The memory hierarchy along with the highly parallel execution model make optimization of the application difficult. The challenges increase many-times when the application to be optimized and parallelized are memory-intensive operations such as Sparse Matrix-Vector multiplication (SpMV), Graph algorithms or VLSI routing algorithms which are critical for most analysis and also simulation tasks for VLSI chip designs

2. PARALLELISM IN STEINER TREE CONSTRUCTION

2.1 The Iterated 1-Steiner (I1S) Approach

One of the earliest work where the idea given in this work was for sequential implementation but is suitable for parallel implementation is the Iterated 1-Steiner heuristic [5, 6], which repeatedly finds optimum single Steiner points for inclusion into the pointset. Given two pointsets A and B, we define the MST as: $MST(A, B) = cost(MST(A)) - cost(MST(A \cup B))$. Intersection points of all horizontal and vertical lines passing through points of P (as defined by Hanan's theorem [1]) Let H(P) denote the Steiner candidate set. For any pointset P, a 1-Steiner point with respect to P is a point $x \in H(P)$ that maximizes $\Delta MST(P, \{x\}) > 0$. the Iterated 1-Steiner (I1S) method repeatedly finds a 1-Steiner point x for $P \cup S$ and sets $S \leftarrow S \cup \{x\}$, starting with a pointset P and a set $S = \emptyset$ of Steiner points,. The $MST(P \cup S)$ cost will decrease with each added point, and thewhen there no longer exists any point x with $\Delta MST(P \cup S, \{x\}) > 0$ the construction terminates

For the Manhattan plane, to find a 1-Steiner point it suffices to construct an MST over $|P \cup S|+1$ points for each of the $O(n^2)$ members of the Steiner candidate set (i.e., Hanan grid points), and then pick a candidate which minimizes the overall MST cost. Each MST computation can be performed in O(n log n) time, yielding an $O(n^3 \log n)$ time method to find a single 1-Steiner point. Exploiting parallelism for this approach is simpler as each processor can independently construct the MST and add the steiner points in I1S method. Then there can be check on the length of each MST at every processor. Whichever MST with steiner point has smaller length can be choosen the process can be repeated for all the points in the plain.

Kahng and Robins [5, 6] proposed the iterated 1-Steiner algorithm where the best possible Steiner point is found iteratively and adds it to the solution set until no further improvement is possible. Barrera et al. [19] developed a straightforward and efficient implementation method for the iterated 1-Steiner algorithm and provided a parallel version of the algorithm. They concluded that the algorithm is very suitable for parallelization and can be generalized to arbitrary weighted graphs (principles like obstacle avoiding and congestion can be easily implemented).

2.2 The Batched 1-Steiner Variant

Even though a single 1-Steiner point may be found with time complexity of $O(n^2)$, the required computational geometry techniques are complicated and not easy to implement. To address these issues, a variant of I1S called batched I1S was developed [5,6], which amortizes the computational cost of finding 1-Steiner points by adding as many "independent" 1-Steiner points as possible in every round. The Batched 1-Steiner (B1S) variant computes $\Delta MST(P,\{x\})$ for each candidate Steiner point $x \in H(P)$ (i.e., the Hanan grid candidate points). Two Steiner points x and y are independent candidates if: $\Delta MST(P, \{x\}) + \Delta MST(P, \{y\}) \leq \Delta MST(P, \{x, y\})$, where each of the two 1-Steiner points does not reduce the potential gain in MST cost relative of the other 1-Steiner point. Given pointset P and a set of Steiner points S, each round of B1S greedily adds into S a maximal set of independent 1-Steiner points. When a round fails to add any new Steiner points termination occurs . The total time required for each round is O(n2logn).

Since each processor can independently compute the MST savings of different candidate Steiner points the I1S and B1S algorithms are extremely parallelizable. The Iterated Steiner approach is therefore very agreeable to parallel implementation on grid computers [7, 6]. The time complexity and practical runtime of B1S can be further improved using incremental / dynamic MST update techniques for I1S. Moreover, by exploiting tighter bounds on the maximum MST degree in the rectilinear metric, further runtime improvements can be obtained [7, 6, 8]

2.3 Parallel Scanline Heuristic

Techniques for producing Steiner trees for a large population of nets in parallel were discussed by Jayaraman and Rutenbar [9]. A scanline style heuristic [10] is proposed that computes the Steiner trees for an random number of nets, each with an arbitrary number of terminals. Given sufficient data parallel machine resources and a even distribution of net sizes this is done in constant time. This algorithm considers that every net terminal is alloted a unique processor and uses a vertical line segment, called a scanline that sweeps and connects the net terminals in lexicographic order (left to right on the x axis, then bottom up on the y axis) and following certain preferences. Only local information is required which gives the advantage for this technique. The terminals are processed in scanline order hence scanline heuristic is explicitly serial. However, parallelization can be easily achieved by identifying and processing all the scanlines simultaneously on data-parallel machines.

2.4 Parallel Heuristic for Macro Cell Routing

A two-phase parallel heuristic algorithm for the STP with usage in the routing of multi-terminal nets is proposed by Fobel and Grewal [11]. A single Steiner tree is constructed using a heuristic called Shrubbery in the first phase. Shrubbery grows the tree using a customized version of Dijkstra's shortest-path algorithm. After the initial solution is obtained a pool of dissimilar, high-quality tress is created from it, by running multiple instances of a local search in parallel. Their experimental results show a almost linear speedup in relation to the number of processors compared to the serial case. In addition, the formed trees are of high quality and dissimilar, allowing for numerous routing possibilities for each net.

2.5 Parallel GRASP

The parallel GRASP algorithm of Martins et al. [12] uses a randomized version of Kruskal's [13] algorithm for the Minimum spanning Tree in the construction phase of the algorithm. A local search method is used which is based on insertions and eliminations of the nodes to/from the current solution. A faster estimation for insertion and deletion moves was achieved by maintaining a list of promising moves of each type, which is periodically updated. This results in a faster execution of the local search. In order to improve the load balancing, the parallelization was made through the distribution of the GRASP iterations among the processors on a demand-driven basis. The local search proved to be the bottleneck of the algorithm and strategies for its improvement are mentioned. The authors continued their work by implementing some of those strategies in a parallel GRASP algorithm using a hybrid local search [14]. The iterations were equally distributed among the available processors. Increasing the number of processors improves the execution times in general. However, for some instances the speedup was linear, while for others the parallelization does not contribute much. The authors conclude that the largest speedups are achieved for the hardest instances, since very few repetitions of the constructed solutions occur.

3. PARALLELISM IN OARSMT CONSTRUCTION

Totsukawa et al. [15] devised a parallel algorithm that accepts a set of points S, the Euclidean MST and a set of obstacles in the space and returns a three-dimensional, rectilinear Steiner tree that avoids obstacles, has a minimum cost and has a bounded number of bends. The algorithm replaces each edge of the Euclidean minimum spanning tree by three rectilinear segments (bends). In order to avoid the obstacles more flexibly, one of the three segments is divided into two segments. The permutations of these segments are the genes that change during the execution of the algorithm. The parallelization was made by dividing the population into subpopulations.

[17] This article presents a parallel algorithm for constructing OARSMTs. The algorithm is based on Watanabe's Steiner tree construction algorithm [18], but there are differences between these two approaches. The algorithm is suitable for use on a shared-memory multi-core computer system, whereas Watanabe's algorithm is suitable for use on a computer system containing a two-dimensional array processor. Specifically, an array processor can have as many as tens of thousands of processing elements, while a shared-memory multi-core computer may have less than ten processor cores. Algorithm adopts two parallelized procedures, namely PARALLELCONNECT() and PARALLEL-CLEANUP(), and these two procedures can efficiently reduce the program execution time. Experimental results show that the implemented parallel program performs efficiently on a shared-memory multi-core workstation.

[16] In this paper, maze routing based approach is used which can also handle large scale OARSMT problems effectively. In this algorithm, in order to handle multi-pin nets, multiple candidates of the shortest path between the pins are kept until all the pins are reached. A graph that is composed of all candidates of the shortest path is formed and the MST of the graph is constructed to create an OARSMT. A post-processing step is then performed to further reduce the total wire-length. A maze router is implemented efficiently by using a heap data structure and propagating on the simplified Hanan grid. This simplified Hanan grid is formed by the original Hanan grid with all the intersection points lying inside an obstacle deleted. In addition, the obstacle-avoiding escape graph is created into a regular array-based data structure. A parallel approach is proposed with applying the multi-pin maze routing algorithm which can be efficiently executed on GPU. Although applying GPU in various computing applications has shown exciting speedup, the use of GPU in EDA is not as popular as the other computational fields.

4. ANALYSIS

Application of parallelism is observed to be very limited in OARSMT as the work carried out is very restricted. The Study of Steiner tree algorithms w.r.t RSMT and OARSMT algorithms have shown that only some of the techniques can be efficiently parallelized. Some techniques provide good results with out parallelizing them. Maze routing approach is the most commonly used for parallelism as it has the inbuilt capability of incorporating parallel implementation of A maze router efficiently by using a heap data structure and propagating on the simplified Hanan grid. This simplified hanan grid is formed by the original hanan grid with all the intersection points lying inside an obstacle deleted. In addition we facilitate the obstacle-avoiding escape graph into a regular array-based data structure. A parallel approach is proposed with applying the multi-pin maze routing algorithm which can be efficiently executed on GPU. In depth study of the routing algorithms which are implemented on Multicore or GPU has also given rise to a hope where OARSMT can be constructed using some of the routing algorithms on GPU or Multicore.

Table 4 The table presents the characteristics of the algorithm

Authors	Method	Heuristics Cost	Ref.
A B Kahng and Robins Barrera, T., Griffith,	repeatedly finds optimum single Steiner points for inclusion into the pointset All processors send their best candidate to a master processor,which selects the best of these candidates for inclusion into the pointset. This procedure is iterated until no improving candidates can be found.	MST computation can be performed in O(n log n) time, yielding an O(n^3 log n) increases with the problem size, reaching about 7.2 for ***n*** = 250 on 10 processors.	[5, 6] [19]
Jayaraman, R., Rutenbar, R.A	considers that every net terminal is alloted a unique processor and uses a vertical line segment, called a scanline that sweeps and connects the net terminals in lexicographic order	Efficient for Data parallel Machines	[9,10]
Fobel and Grewal	A single Steiner tree is constructed using a heuristic called Shrubbery the grows the tree using Dijkstra's shortest-path algorithm. A pool of dissimilar, high-quality tress is created from it, by		[11]

	running multiple instances of a local search in parallel		
Totsukawa et al.	The algorithm replaces each edge of the Euclidean minimum spanning tree by three rectilinear segments (bends)	Considerable performance improvement	[15]
Wing-Kai Chowa,n,	parallel implementation of A maze router efficiently by using a heap data structure and propagating on the simplified Hanan grid	Efficient for Large Graphs	[16]

5. CONCLUSION

The steiner tree construction plays a significant role EDA design. Many mathematicians devoted their attention to the problem whose solution is now recognized as the Steiner tree for three points in a plane. But as the problem is NP-hard, finding a polynomialtime algorithm for solving the problem exactly is difficult. This is why many non-exact solutions have been proposed. But with today's existing computing power, many parallel algorithms are introduced. These parallel concepts assist to obtain better results much faster than ever before.

In this paper a survey is done to analyze what existing sequential algorithms can be implemented in parallel and also analysis of limited number of existing parallel algorithms for constructing a steiner tree. combinatorial Also, many of the

The metaheuristic approaches proved to be among the most successful in serial algorithms and have proved their value once again in parallel computation.

Still, new generalizations are considered on a daily basis since the STP can be easily adjusted to a broad range of real-life problems. But, even after major accomplishments in serial algorithms, not much research is being done exploiting the parallel aspect other than the doing parallelism at the hardware level using FPGA.

6. ACKNOWLEDGMENTS

The work reported in this paper is supported by the college through the TECHNICAL EDUCATION QUALITY

IMPROVEMENT PROGRAMME [TEQIP-II] of the MHRD, Government of India.

7. REFERENCES

[1] Hanan, Maurice. "On Steiner's problem with rectilinear distance." SIAM Journal on Applied Mathematics 14.2 (1966): 255-265.

[2] Hwang, Frank K. "On Steiner minimal trees with rectilinear distance." SIAM journal on Applied Mathematics 30.1 (1976): 104- 114.

[3] Ho, J-M., Gopalakrishnan Vijayan, and C. K. Wong. "New algorithms for the rectilinear Steiner tree problem." ComputerAided Design of Integrated Circuits and Systems, IEEE Transactions on 9.2 (1990): 185-193.

[4] J. L. Ganley and J. P. Cohoon, "Routing a multiterminal critical net: Steiner tree construction in the presence of obstacle," in Proc. ISCAS, 1994, pp. 113–116.

[5] A. B. Kahng and G. Robins. A new class of iterative steiner tree heuristics with good performance. IEEE Transactions Computer-Aided Design, 11(7):893–902, July 1992.

[6] A. B. Kahng and G. Robins. On Optimal Interconnections for VLSI. Kluwer Academic Publishers, Boston, MA, 1995.

[7] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang. Closing the gap: Near-optimal steiner trees in polynomial time. IEEE Transactions Computer-Aided Design, 13(11):1351–1365, November 1994.

[8] G. Robins and J. S. Salowe. Low-degree minimum spanning trees. Discrete and Computational Geometry, 14:151–165, September 1995.

[9] Jayaraman, R., Rutenbar, R.A.: A parallel Steiner heuristic for wirelength estimation of large net populations. In: 1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers, pp. 344–347 (1991)

[10] Dunlop, A.E., Kernighan, B.W.: A procedure for placement of standard-cell VLSI circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 4, 92–98 (1985)

[11] Fobel, C.,Grewal, G.: A parallel Steiner tree heuristic for macro cell routing.In: IEEE International Conference on Computer Design, vol. 27–33 (2008)

[12] Martins, S.L., Ribeiro, C.C., Souza, M.C.: A parallel GRASP for the Steiner problem in graphs. In: IRREGULAR '98: Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel, pp. 285–297 (1998)

[13] Kruskal Jr, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am.Math. Soc. 7, 48–50 (1956)

[14] Martins, S.L., Resende,M.G.C., Ribeiro, C.C., Pardalos, P.M.: A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. J. Global Optim. 17(1), 267–283 (2000)

[15] Totsukawa, H., Senou, H., Ohmura, M.: A parallel genetic algorithm for 3-D rectilinear Steiner tree with bounded number of bends. In: 2008 51st Midwest Symposium on Circuits and Systems, pp. 89–92 (2008)

[16] Wing-Kai Chowa,n, LiangLi a, Evangeline F.Y.Young a, Chiu-WingSham b Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach

[17] A Parallel Algorithm for Constructing Obstacle-Avoiding Rectilinear Steiner Minimal Trees on Multi-Core Systems Cheng-Yuan Chang and I-Lun Tseng, Department of Computer Science and Engineering, Yuan Ze University, Taiwan

[18] Takumi Watanabe, Hitoshi Kitazawa, and Yoshi Sugiyama, "A Parallel Adaptable Routing Algorithm and Its Implementation on a Two-Dimensional Array

Processor," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 6, no. 2, pp. 241-250, 1987

[19] McKee, S.A., Barrera, T., Griffith, J., Robins, G., Zhang, T.: Toward a Steiner engine: enhanced serial and parallel implementations of the iterated 1-Steiner MRST algorithm. In: Proceedings of the Third Great Lakes Symposium on VLSI-Design Automation of High Performance VLSI Systems,vol. 2442, pp. 90–94 (1993)

[20] SWARM: A Parallel Programming Framework for Multicore Processors David A. Bader, Varun Kanade and Kamesh Madduri