A High Performance Global Routing Algorithm on Julia Parallel Computing Platform

[1]MEENAKSHI AGARWALLA, [2]MANASH PRATIM SARMA, [3]KANDARPA KUMAR SARMA
[1,2,3]Department of Electronics and Communication Engineering, Gauhati University, INDIA

Abstract—To keep pace with the design requirements of Integrated Circuits (ICs), parallel processing is adopted. The path to be routed between two nodes may or may not be dependent on the previously routed paths. The solution requires careful attention in distributing the nets to be routed to different processors. Previous work on allocating the tasks to processors has been quite successful, reporting upto 3x improvement on 4 cores and 5x improvement on 8 core machine. The advantage of increasing the number of cores diminishes with each added processor and the challenge lies in being able to maintain the improvement per added core. The existing techniques of distributing the nets cannot provide additional advantage of using more than 8 cores. This paper improves the work on parallelizing global routing using a technique of balancing the load on the processors for better utilization of the resources. A relatively new budding platform Julia has been used which provides the ease of programming while maintaining the performance of the C language. Technique used in this paper has enabled to use 16 cores with routing solutions obtained in 0.8 minutes achieving 12.5 times speedup compared to sequential processing on a single core.

Keywords : Wirelength, Runtime, Congestion, MPI, RMST, RSMT, RRR, MR-GR, MRB-GR, HBLMR- GR

Received: March 3, 2021. Revised: July 28, 2021. Accepted: July 30, 2021. Published: August 10, 2021.

1. Introduction

The miniturization of Integrated Circuits (ICs) has greatly increased the complexity in all phases of the IC design flow especially routing. Routing is analogous to finding the roads between the pins of circuit elements to be connected. It is a two-fold process that involves global routing followed by detailed routing. Global routing simply specifies the tracks to be used for interconnections without abiding the design rules required for fabrication [1], as shown in Fig: 1. Also, it needs to be performed multiple times and demands for exploring the strength of parallel processing using efficient programming languages. Parallel processing involves the use of multicomputers or multiprocessors. The increased number of cores for parallelism reduces the runtime but it has not been possible to utilize the advantage of adding cores successively. Programming language Julia has the capability to handle parallel problems in an easy to use manner with simplified coding styles compared to other languages. A fast routing algorithm on parallel grounds under the shield of high-performance and dynamic programming language is the key to optimization of the problem of global routing.

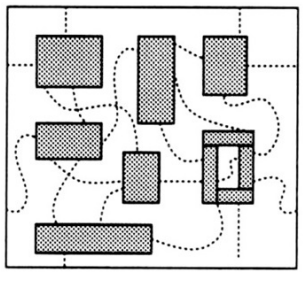

Fig. 1. Global Routing [1].

Programming languages come in two flavours, viz,-

- System languages which are hard to use but are fast, and
- Scripting languages which are easy to use but are slow.

Attempts to get the best of both the worlds have given birth to Julia [2], [3] which enjoys the speed comparable to C and dynamic capability of Python. The credit of fast execution in Julia goes to it being a Just-In-Time (JIT) compiled language unlike C, Fortran where codes are compiled before execution. Julia uses On-the-fly code generation called metaprogramming, multiple dispatch, dynamic typing and readable code as a paradigm, making it the cunningest player of all high-performance languages.

Parallelizing the routing process requires identification of independent nets and distribution of these nets to different processors. Related works [4], [5] have parallelized routing using upto 8 cores. The utilization of increased number of cores cannot be justified with the previous reported methods. This paper presents a technique of extending parallelism using more than 8 cores. A fast global routing algorithm is implemented using Julia on High-Performance Computing (HPC) machine with 12.5x speedup achieved in 0.8 minutes with balanced load compared to the sequential processing.

The paper is organized into six sections of which Section II highlights the steps involved in the conventional global routing algorithms. Section III deals with parallelization techniques and Section IV discusses the implementation steps of Global routing algorithms using Julia. The results are presented in Section V and finally, Section VI concludes the paper with future directions.

2. Global Routing and Heuristics

2.1 Overview of Global Routing Process

Global Routing accounts for its time consuming nature as it is performed multiple times. It is a complex process with steps involved shown in Fig: 2. Initially, paths are searched in between the pins to be connected by allowing congestion. Congestion is the traffic on the layout where the number of paths through a route exceeds its capacity. The used paths are given weighted values to identify congested areas. The overflowing areas undergo Rip-up and Re-route (RRR) stage [6], [7] for nullifying the effect of congestion. RRR comes into play after the initial routing solutions are obtained and involves the following two steps:

- Areas with congestion are found and a few nets under violation are ripped.
- Nets considered for rip-up are rerouted in a pre-defined order.

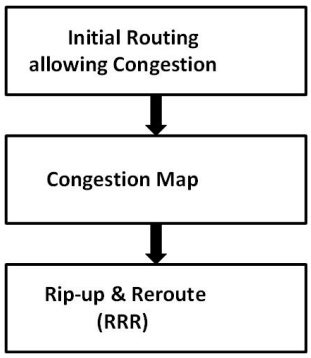

Fig. 2. Steps in Global Routing.

2.2 Two-pin Net Routing

A net is a path connecting two points. In IC design, two-pin net routing involves connection between the source and the target. Some of the routing algorithms are:

Lee's Algorithm: Lee's algorithm [8] is based on the principle of Breadth First Search (BFS) technique considering a gridded-graph layout as shown in Fig: 3.

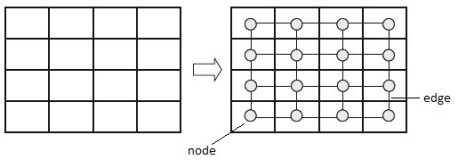

Fig. 3. Global-Routing Graph

It is a two-stage process:

- Forward wave propagation
- Backtracing

Lee's algorithm, also known as maze algorithm, enjoys the advantage of finding the shortest path between two pins to be connected. But it has some disadvantages that include:

- Runtime is more as it searches in all directions from the source.
- It consumes a lot of memory space for storing a huge amount of unwanted data.
- It cannot be applied to multi-pin nets as it is meant for two-pin nets.

The work [4] proposes a variation of maze routing algorithm that is applicable to multi-terminal

nets. Defects of Lee's algorithm are handled using bounding box [9], [10] which reduces the search space as well as the processing time. Use of bounding box comes at the expense of long wirelengths. The processing time for routing using maze algorithm is minimized by applying constraints to the wirelength and the resources used for routing [11].

A Search Routing:* The A* search algorithm is based on the principle of Depth First Search (DFS) technique where the search for the path is always directed towards the target. The direction is searched using a cost function f(n) given by -

$$f(n) = g(n) + h(n) \tag{1}$$

where *n* represents the path connecting the initial node and the final node, *g(n)* denotes the weight of the path connecting the initial node and the intermediate node and *h(n)* denotes the weight of the path connecting the intermediate node to the final node. The search is carried out directing towards the target by selecting the successive nodes with lowest cost f(n).

A negotiation-based A* search routing algorithm [10] helps routing of nets through areas with low values of congestion. Global routers [5], [9] propose to obtain routing solutions very fast using bounding box but the weakness lies with long routing wirelengths and occupation of more routing resources. A variation of A star algorithm with bounding box expansion method [11] reduces the processing time of maze routing algorithm by applying constraints to the wirelength and the routing resource. ***Han et al.*** [12] proposes to obtain a fast hand initial routing solutions using A star algorithm.

The A* search routing has advantages of being fast and requires less memory relative to maze routing.

2.3 Multi-pin Net Routing

Nets with more than two pins to be connected are called multi-pins and the tracks connecting these pins form the multi-pin routed path. Steiner trees play a major role in this regard by forming subnets of two-pins, followed by routing through either Rectilinear Minimum Spanning Tree (RMST) [13] or Rectilinear Steiner Minimum Tree (RSMT) algorithm. RSMT algorithm uses Steiner points [14] to alleviate the problem of increased wirelength and rectilinear lines which cannot help in areas under congestion.

Liu et al. [15] proposes to obtain cost saving routing solutions in terms of wirelength by using a combination of RSMT and RMST unlike the global routers [16] [17] that use RMST.

3. Proposed Parallelization Approach

Parallelism is the key to gear up the process of global routing. Global routing can be parallelized by two startegies:

- Partitioning based concurrency approach [18] and
- Task based Concurrency approach (TCS) [11].

In the former approach, parallelization is possible by allowing processors to route nets in allocated subregions. In the latter approach, processors can find routing solutions for nets in parallel without any restriction in the search space [11]. Routing in parallel the nets with overlapping bounding areas [4] leads to congestion.

Another approach is to route nets in parallel with non-overlapping areas from the formed group of nets based on wirelength [19] and use Graphics Processing Unit(GPU) with multi-core processors [12]. This approach suffers from load imbalancing and hinders the advantage of using more cores. Our proposed method involves identification of as many non-overlapping nets as are processors available. This technique has enabled to extend parallelism upto 16 cores efficiently with balanced distribution of these nets to processors for routing.

4. Implementation

4.1 Sequential and Parallel Processing Blocks

The steps involved in sequential and parallel processing are shown in Fig: 4

1) Grouping of Nets: The process of routing differently sized nets by processors hinders efficiency. Hence, groups of same - sized nets are to be formed based on some parameters like wirelength [19].

Grouping of nets offers the following advantages:

- Different routing algorithms can be applied to different groups.

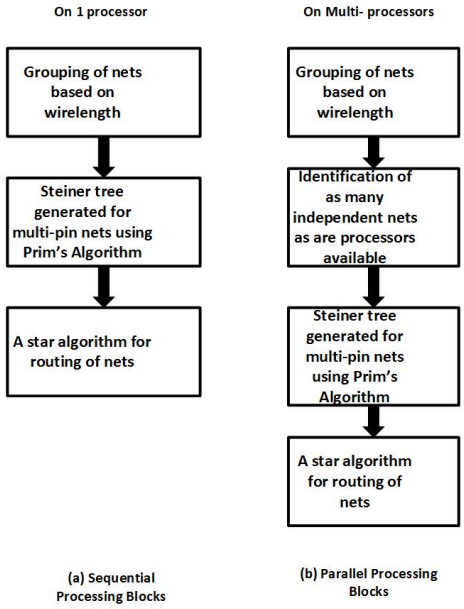

Fig. 4. Sequential and Parallel Processing Steps.

- Nets within a group have similar wirelengths and hence, take the same time for processing.
- Parallelism can be applied efficiently to obtain routing solutions in minimum time.

2) Identification of Independent Nets within each cluster: Nets with non-overlapping bounding boxes are said to be independent and can be considered for parallel routing.

- Non-overlapping nets are identified using an efficient technique over [19]. Non-overlapping nets in a group must not exceed the worker count as demonstrated in Fig: 5 and Fig: 6.
- The initial experiments are carried out iteratively in a loop by identifying four non-overlapping nets using four cores.
- On an HPC machine, eight and sixteen cores are used with the subsequent number of independent nets in each iteration within a cluster.

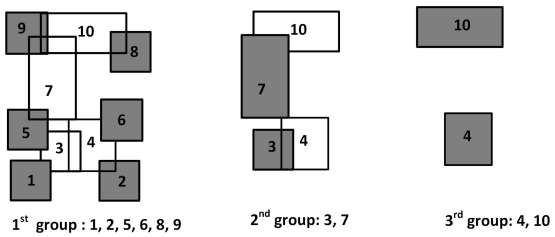

Fig. 5. Grouped Nets for Parallel Processing as in [19]

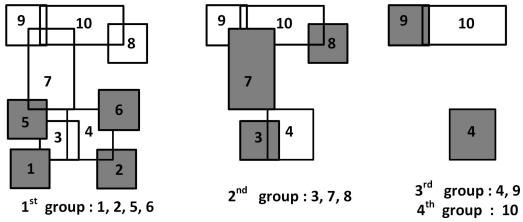

Fig. 6. Grouped Nets for Parallel Processing used in the current work.

Our proposed technique enjoys the following advantages:

- Balanced load.
- Overhead on processors is reduced.
- Runtime is improved.

3) Prim's Algorithm and A star algorithm: A spanning tree with minimum cost that connects all the pins of a multi-pin net is obtained using Prim's algorithm. This tree consists of a subset of edges with minimum weight.

A star algorithm is used for routing the nets. It uses the lowest cost function to find the path between two pins. The cost functions are the distances in between the nodes in consideration. The formula used for calculating the weights are the well-known Manhattan distance [20] and Euclidean distance [21] equations. Manhattan distance between two nodes with co-ordinates (x1,y1) and (x2,y2) is given by

$$Manhdist = |x1 - x2| + |y1 - y2| \qquad (2)$$

Euclidean distance between two nodes with co-ordinates (x1,y1) and (x2,y2) is given by

$$Eucldist = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \qquad (3)$$

Using the Manhattan distance for cost calculation posed a problem as it gave almost similar results for all neighbors corresponding to a source node, thereby creating a perplexed situation for selecting the next node. So, we used the Euclidean distance formula which solved the problem but it consumed a lot of runtime because of the square root function. Hence, we used a modified equation for our purpose, given by

$$Modfdist = (x1 - x2)^2 + (y1 - y2)^2 \qquad (4)$$

The use of square root function causes the data type to be changed from Integer format to Floating

format which is strictly prohibited in Julia environment for processing efficiency. Also, the distance calculated without square root doesnot affect the final solution.

5. Results and Discussion

Experiments are carried out on *adaptec1* with 219794 nets which is one of the benchmarks of ISPD 2008 [22] using the environment Julia. The results are obtained on a 64-bit quad-core processor with 7.7 GB of memory. Fig: 7 presents the results obtained after forming groups of nets based on the semiperimeter of their bounding boxes as in [19].

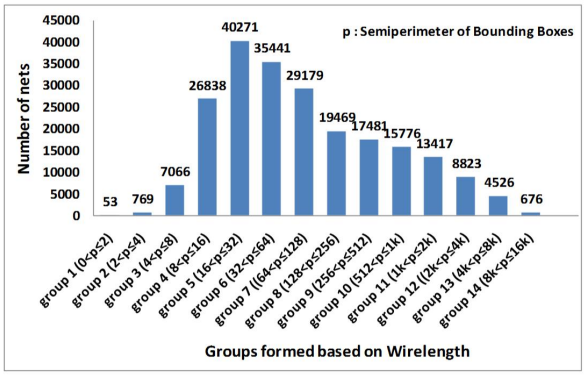

Fig. 7. Net Count in the Groups formed based on Wirelength.

Table I compares the simulation time for sequential processing and parallel processing using four cores.

TABLE I
PERFORMANCE COMPARISON ON SINGLE CORE AND QUAD CORE

Group Name	Net Count	Runtime on Single core (s)	Runtime on Quad core (s)	Speed-up
group1	53	0.002	0.002	1x
group2	769	0.04	0.02	2x
group3	7066	0.4	0.19	2.1x
group4	26838	2.1	0.84	2.5x
group5	40271	5.1	1.75	2.9x
group6	35441	7.5	2.47	3.03x
group7	29179	11.8	3.59	3.29x
group8	19469	14.8	4.25	3.48x
group9	17481	26.4	7.52	3.51x
group10	15776	49.2	12.88	3.82x
group11	13417	100.3	24.19	4.15x
group12	8823	196.7	32.8	5.99x
group13	4526	149.7	49.4	3.03x
group14	676	35.9	20.75	1.73x

Table II shows the performance on HPC machine using eight and sixteen cores with speedups compared to sequential results.

Table III shows runtime comparison with NTHU-Router 2.0 [23] and GPU-CPU [12] based router on 1 core and 4 cores. Also, simulation time

TABLE II
PROCESSING TIME USING VARIOUS NUMBER OF CORES ON HPC

Group Name	Parallel Processing (with 8 cores) (s)	Speedup	Parallel Processing (with 16 cores)(s)	Speedup
group1	0.0019	1.05x	0.0015	1.3x
group2	0.0139	2.88x	0.0092	4.35x
group3	0.098	4.08x	0.099	4.04x
group4	0.33	6.36x	0.236	8.89x
group5	0.61	8.36x	0.352	14.5x
group6	0.84	8.93x	0.519	14.45x
group7	1.17	10x	0.772	15.28x
group8	1.35	10.96x	0.97	15.26x
group9	2.57	10.27x	1.82	14.5x
group10	4.36	11.28x	3.44	14.3x
group11	9.19	10.9x	8.58	11.69x
group12	14.57	13.5x	11.66	16.87x
group13	17.56	8.53x	13.9	10.77x
group14	7.35	4.88x	5.76	6.2x

is compared with Maze Routing Global Router without bounding box (MR-GR) [11], Maze Routing with Bounding box Global Router (MRB-GR) [15] and Heuristic Bounded Length Maze Router (H-BLMR-GR) [15] on 8 cores. Quad-core processing results on Julia Router are 1.12 times better than NTHU Router2.0 and almost same as GPU-CPU Router. Results on 8 cores are 5.9 times,3.85 times and 2.65 times better on Julia Router than MR-GR, MRB-GR and H-BLMR-GR respectively. Also, the results obtained on 16 cores using Julia show 12.5x speed-up compared to processing on 1 core.

TABLE III
RUNTIME COMPARISON WITH NTHU ROUTER 2.0, GPU-CPU BASED ROUTER, MR-GR, MRB-GR, H-BLMR-GR

Number of cores	Name of the Router	Programming Language	Simulation Time (min)
1	NTHU 2.0 [23]	C/C++	9.95
	Julia Router[**This Work**]	Julia	9.9
4	NTHU 2.0 [23]	C/C++	2.9
	GPU-CPU [12]	C/C++	2.51
	Julia Router[**This Work**]	Julia	2.6
8	MR-GR [11]	C/C++	5.9
	MRB-GR [15]	C/C++	3.85
	H-BLMR-GR [15]	C/C++	2.65
	Julia Router[**This Work**]	Julia	1
16	Julia Router[**This Work**]	Julia	0.8

Table IV shows the performance comparison of an un-optimized Julia code which is equivalent to Matlab or Python Code with optimized Julia code. The results show that the performance is undoubtedly better than that obtained in other scripting language.

TABLE IV
PERFORMANCE COMPARISON OF UN-OPTIMIZED JULIA CODE (EQUIVALENT TO MATLAB, PYTHON CODE) WITH OPTIMIZED JULIA CODE

Number of Cores	Un-optimized Julia Code (min)	Optimized Julia Code (min)	Speedup
1	472.6	9.9	47.7x
4	167.35	2.6	64.36x

6. Conclusion

This paper presents implementation of global routing on parallel platform. Simulations are done on a single core, quad core and on High Performance Computing Machine (HPC) using eight and sixteen cores in Julia programming environment. It is observed that Julia offers the ease of programming and debugging of high-level languages and allows optimization of code giving performance of low-level languages. The solutions have improved by 12.5 times through massive parallelism. This parallelism has been possible due to the proposed technique of distributing tasks to processors. The achieved speedup will undoubtedly save the processing time in post routing phases. The parallelism can be stretched out to an extent limited by the number of independent nets. It is expected to open up new avenues to parallelize the global routing process efficiently with minimum processing time by using variations of routing algorithms.

References

[1] N. A. Sherwani, "Algorithms for VLSI Physical Design Automation", *Springer*, 3rd ed., Berlin, Germany, 1999.

[2] I. Balbaert, A. Sengupta and M. Sherrington, "Julia: High Performance Programming", *P*ackt, 1st ed., Birmingham, UK, 2016.

[3] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, "Julia: A fresh approach to numerical computing", *Society for Iand Applied Mathematics (SIAM)*, Vol :1, pp. 65-98, 2017.

[4] Y. Shintani, M. Inagi, S. Nagayama and S. Wakabayashi, "A Multithreaded Parallel Global Routing with Overlapped Routing Regions", *Euromicro Conference on Digital System Design(DSD)*, September 2013.

[5] M. D. Moffitt, "MaizeRouter: Engineering an Effective Global Router", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol: 27, pp. 2017-2026, Issue : 11, November 2008.

[6] J. R. Gao, P. C. Wu and T. C. Wang,"A new global router for modern designs", *Asia and South Pacific Design Automation Conference(ASP-DAC)*, pp. 232-237, 2008.

[7] M. M. Ozdal and M. D. F. Yong,"ARCHER: A history-driven global routing algorithm", *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 488-495, November 2007.

[8] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits", *Integration,The VLSI Journal, Elsevier*, Vol: 31, pp. 1-49, Issue: 1, November 2001.

[9] M. Cho and D. Z. Pan, "BoxRouter: A new global router based on box expansion and progressive ILP", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol : 26, pp. 2130-2143, Issue : 12, December 2007

[10] M. Cho, K. Lu, K. Yuan and D. Z. Pan, "BoxRouter2.0: Architecture and Implementation of a hybrid and robust global router", *International Conference on Computer-Aided Design (ICCAD)*, November 2007.

[11] W. H. Liu, W. C. Kao, Y. L. Li and K. Y. Chao, "Multi-threaded Collision-Aware Global Routing with Bounded-Length Maze Routing", *Proceedings of the 47th Design Automation Conference (DAC)*, pp. 200-205, June 2010.

[12] Y. Han, D. M. Ancajas, K. Chakraborty and S. Roy, "Exploring high-Throughput Computing Paradigm for Global Routing", *IEEE Transactions on VLSI systems*, Vol: 22, January 2014.

[13] J. A. Roy and I. L. Markov, "High performance Routing at the nanometer scale", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol: 27, pp. 1066-1077, Issue : 6, June 2008.

[14] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement", *IEEE/ACM International Conference on Computer-Aided Design*, 2006.

[15] W. H. Liu, W. C. Kao, Y. L. Li and K. Y. Chao, "NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing with Bounded-Length Maze Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol:32, pp. 709-722, Issue:5, May 2013.

[16] C. Chu, "FLUTE: Fast look-up table based wirelength estimation technique", *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 696-701, November 2004.

[17] C. C. N. Chu and Y. C. Wong, "Fast and Accurate Rectilinear Steiner minimal tree algorithm for VLSI Design", *Proceedings of International Symposium on Physical Design (ISPD)*, pp. 28-35, April 2005.

[18] T. H. Wu, A. Davoodi and J. T. Linderoth, "A parallel integer programming approach to global routing", *Proceedings of the 47th Design Automation Conference (DAC)*, pp. 6, June 2010.

[19] D. Tumelero, G. Bontorin and R. Reis, "Overhead for Independent Net Approach for Global Routing", *IEEE 6th Latin American Symposium on Circuits and Systems(LASCAS)*, February 2015.

[20] M. Sarrafzadeh, K. F. Liao and C. K. Wong, "Single-layer global routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, Issue: 1, 1994.

[21] Y. C. Hsu, Y. Pan and W. J. Kubitz, "A path selection global touter", *Proceedings of the 24th ACM/IEEE Design Automation Conference*, Pages - 641-644, USA, 1987.

[22] Cliff Sze. (2008), *ISPD 2008 Contest* [Online]. Available: http://archive.sigda.org/ispd2008/contests/ispd08rc.html.

[23] Y. J. Chang, Y. T. Lee and T. C. Wang, "NTHU-Route2.0: a fast and stable global router", *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 338-343, November 2008.