Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

Memory And I/O Efficient Rectilinear Steiner Minimal Tree Construction Under High Performance Computing Environment

Dr. Latha N.R[1] , Dr. Pallavi G B[2] , Dr. Shyamala G.[3] , Dr. G R Prasad[4]

[1,2,3]Assistant Professor, Dept. of CSE B.M.S. College of Engineering, Bangalore, India.

[4]Professor, Dept. of CSE B.M.S. College of Engineering, Bangalore, India.

Abstract—In modern VLSI circuit reducing the runtime as well as wirelength are considered to be the most preferred objectives. Thus, Rectilinear Steiner Minimal Tree (RSMT) construction is challenging. FLUTE (Fast Look-Up table) is the widely used method for fast and accurate RSMT construction. The FLUTE attained very good performance for smaller as well as higher degree nets, however, this system induces memory overhead. But it is important to utilize memory in an efficient manner. Therefore to overcome these persisting research problems, this work proposes a memory and I/O efficient RSMT (MIOERSMT) construction. In addition, by using high performance computing (HPC) environment like CPU and GPU further reduction in the execution time for constructing RSMT. But, GPU based model induces high deployment cost and requires efficient memory management method. Therefore, Performance and memory constraint parallel computation (PMCPC) algorithms are proposed that helps in efficient utilization of the computational capacities of these shared-memory multi-core HPC model. Investigations are performed on small to large nets using ISPD'98 benchmarks. The outcome achieved demonstrates that the suggested routing model reduces wire length, improves memory utilization, and also attains better processing time in comparison with the sequential and existing VLSI routing model.

Keywords—Graphical processing unit, HPC, Multi-core environment, Parallel computing framework, RSMT, VLSI.

I. INTRODUCTION

Electronic design Automation in Very Large Scale Integration (VLSI) is a scheme where hundreds to thousands of electronic modules are placed on a single chip. Designing competent algorithm for global routing process of VLSI is in demand with the increase in logic circuits numbers as well as the rapid increase in memory capabilities. Placing the components in good positions is crucial, as this could gradually decline the power and heating up of chip, which in turn results in reduced chip size and decrease in production cost. Thus the main criterions that must be considered are reducing the aggregate wire-length, space and chip cost. Majority approaches concentrate towards lessening the aggregate wire-length. In the VLSI physical design.

routing is a phase which involves global routing stage and then followed by detailed routing stage. First stage i.e., global routing does connections among blocks, however will not give details of each wire or pins. On the other hand second stage of routing performs point to point links amongst pins of every block. Further routing uses Rectilinear Steiner Trees (RST) in order to join all the pins of a net. With improvement in VLSI technology, the amount of pins that are to be connected is increasing to hundreds, and in some cases even to thousands. Finding RST's for problems of this size takes more execution time and leads to increased IC design process time and increased design cost.

The Main purpose of routing is finding Minimal RST's or constructing Rectilinear Steiner minimum tree (RSMT). Generating RSMT includes connecting the entire pins of a net in horizontal or vertical manner. For a group of given N points of a plane, Rectilinear Steiner Minimum Tree (RSMT) problem involves identifying the least distance rectilinear Steiner tree which joins these points accomplished by means of extra points known as Steiner points. Generating RSMT is of major concern in VLSI design phases namely interconnection, placement and floor planning. This is used in estimating, delay in transmission and delay in interconnection and also for computing workload. RSMT construction is applied during various global routing stages in constructing routing structure for all nets. This classical problem is proved as Non-deterministic polynomial problem [1]

The Generation of RSMT in VLSI is characterized as Non-deterministic polynomial problem [1], consequently rectilinear minimum spanning tree (RMST) are being used for maximum methods adapting dimensionality of space [2]. A fast and accurate look up table approach that provided optimal solution in constructing RSMT namely FLUTE was presented by [3] and [4]. The algorithm is built on the lookup table concept that is, for given a problem it finds the solution by referring to the lookup table instead of computing it from the scratch. But this approach has a drawback of applying it to nets of degree up to 9. For bigger nets, the net is to be broken down and then the FLUTE has to be applied, which increased runtime, wire-length and induced memory overhead. FLUTE is found to be very effective for bigger net having the runtime complexity $O(n \log n)$. The problem with this technique is that for bigger nets, the accuracy is affected severely mainly because of the error introduced while the nets are divided.

Further, the FLUTE is improved by presenting a net partitioning technique that is scalable [5]. Here the bigger nets are divided as subset of nets, later are merged by including Steiner nodes. Hence the approach handled the smaller as well as bigger degree net incurring minor reduction of accuracy that resulted in runtime complexity of $O(n \log^2 n)$. Later an enhanced lookup table based RSMT construction [6] was presented that resulted in a better balance between accuracy and runtime. The works [5] and [6] both does not consider memory limitations while generating the Look-up table during RSMT construction. Future design in VLSI consists of static blocks namely IP blocks and macros, FLUTE technique can be applied in the research works [7] [8] and [9]. In these designs the most preferred objective is to minimize wire length and reduce memory overhead. In [9] presented a technique named FOARS (FLUTE based Obstacle Avoiding RSMT construction (OARSMT)). This approach segments group of pins to several subgroups applying top-down method. While many algorithms were proposed for solving the RSMT construction problem as in [20] [21] [22] and[25], the majority of these algorithms are sequential, and are not parallel [15]. In [10] [28] [29] proved global routers normally decompose a net using RSMT. Thus, decreasing clogging as well as gaining flexibility relies on

generation of RSMT. Nevertheless, little improvement in wire length.In [11] as well as [12] [35] and [36] present a method that resolves global routing issues. The work [11] proposes a technique called as GRIP (Global Routing Technology using Linear Programming) and [12] propose an approach that adapts FLUTE technique and creates a fast Steiner tree that is congestion driven, but have not made efficient utilization of memory as well as CPU. In order to resolve congestion problem in global routing [13] as well as [14] proposed a model adopting game theory and clustering technique that improved the runtime complexity of routing in VLSI physical design. The work [15] proves maze routing could generate OARSMT on heterogeneous high performance computing (HPC) platform like CPU and GPU.

Similarly, [16] exploited heterogeneous computing to propose an approach for placement based on parallel clustering. The technique makes use of CPU as well as GPU cores to full extent. The result proves that this model achieve better runtime compared with its sequential strategy[19] and [23. But, using GPU results in additional deployment cost and also memory limitation is not taken into consideration by the model, which led to more I/O access time. In addition, it is observed processing job on high performance computing environment is constrained by memory bounds as utmost time jobs are waiting for memory availability for further execution. Therefore existing routing designs prompts high computation cost on shared memory environment.

For overcoming the above investigated problems, this research work proposes a Memory and I/O efficient RSMT (MIOERSMT) construction [17]. The MIOERSMT is designed to reduce wire length, processing time and improve memory utilization. Further, the MIOERSMT construction is accomplished adopting parallel approach using high performance computing platform. The Performance and memory constraint parallel computation (PMCPC) algorithms facilitates in exploiting the computational power from these shared-memory multi-core HPC systems.

The contributions of the research are as depicted below:

- Propose a novel parallel computing environment for constructing a memory and I/O efficient RSMT for VLSI circuit.
- The novel approach proposed minimizes wire length, the computation time as well as memory requirements.

This Research paper is organized as : Section I, comprises of introduction to parallel computing environment for constructing a memory and I/O efficient RSMT for VLSI circuit. Additionally, it projects the issues and challenges in the memory and I/O efficient routing design for VLSI circuit. Section II, presents the proposed novel parallel computing environment model for developing memory and I/O efficient routing model. Experimental result and analysis are presented in section III. Final Section concludes and proposed the future research direction of work.

I. INTRODUCTION TO PARALLEL COMPUTING ENVIRONMENT TO PERFORM MEMORY AND I/O EFFICIENT RSMT CONSTRUCTION FOR VLSI CIRCUIT

With the increase in computer technology general-purpose multicore processors are being largely adopted in different areas of the industry. These include signal processing and embedded space as there is

increased need for additional performance efficiency for these general-purpose applications. Parallel processing amplifies the performance significantly by enhancing the parallel resources, without increasing the power requirements. There are abundant and diverse implementations of multicore processors and these Designs vary from the usual multi-processor systems to that which consists of many programmable Arithmetic Logic Units (ALUs).

In order to develop an effective multicore execution framework, we much understand the system with respect to architectural and algorithmic perspectives. There are different characteristics that have to be considered while designing methodologies for HPC systems like understanding prerequisite for accomplishing good parallelization, map the parallel threads to group of functional and processing unit, exploiting processor core comprising limited functionalities, familiarizing on constrained on-chip memory and off-chip memory capacity and the performance on multicore systems.

In addition to the above features, there is a need to consider other multicore issues which play a very important role in parallel programming like Number of processing cores, caching, memory bandwidth and synchronization

Multi-core processors today are gaining popularity as it finds extensive application in high performance computing and in user electronics. Adopting graphics processor for general purpose computation is becoming common, as this result in a steep increase in performance when used to particular applications. Such increase in parallelism (multiple threads on a CPU or GPU) and heterogeneity (simultaneous use of a CPU and GPU) have significantly improved the productivity and overall performance of many traditional compute intensive systems [26] [30] and [33].

Today, there still exists certain applications that demands faster execution but efficient parallel or heterogeneous executions are not designed till now like RSMT construction. Programming languages have been developed to utilize the cores available in multicore of GPU

The CUDA language for programming GPGPU are modeled to execute on only NVIDIA's GPUs, while Open CL are designed to work on different manufacturers of multi core CPU and GPU devices, including NVIDIA's GPUs. In spite of CUDA's hardware constraint, Open CL follows similar syntax and other features of CUDA. GPU's have become dominant as new parallel programming interfaces for general purpose computations are being introduced like Computer Unified Device Architecture (CUDA), Stream SDK and Open CL. This makes GPUs to be selected as the best option for solving high-performance numerical application, scientific problems and also engineering applications.

But with all the above said features designing programs to work on GPUs is still a challenge. The main reason for this is contemporary GPUs involves composite memory organization comprising multiple low latency on-chip memory other than the off-chip memory. Further, the access latency and optimal access pattern of each memory is considerably different, this poses a major challenge to design methods which consume in an optimal way the various memories, endure their latencies and enhance the memory considerations [27] [31] [32] and [34]. Optimizing an application becomes difficult because of the memory

hierarchy and the extremely parallel execution model[24]. These challenges surge exponentially if the applications that needs optimization and parallelization are memory intensive processes like Sparse Matrix-Vector multiplication (Sp MV)[33], Graph algorithms or VLSI routing algorithms which are critical in most analysis and simulation tasks of VLSI design

High performance computing environment such as CPU and GPU does not take memory limitation into consideration and therefore, it is observed processing job on high performance computing environment is constrained by memory bounds as utmost time jobs are waiting for memory availability for further execution. Therefore existing routing designs prompts high computation cost on shared memory environment. For overcoming these research issues and challenges, it is important to design a VLSI routing model that minimizes wire length, runtime as well as memory utilization and develop memory constraint High performance computing environment for constructing RSMT with reduced wire length, memory usage, and improved speedup.

II. High Performance Computing environment Model for Establishing Memory and I/O Efficient Rectilinear Steiner Minimum Tree Construction

The segment proposes a High performance computing structure towards generating memory and I/O efficient RSMT (MIOERSMT) routing technique. In the beginning, the work specifies the method of accomplishing the memory and I/O efficient RSMT. Later it describes the process of parallel computation of MIOERSMT on high performance computing environment. At the end parallel computation model is presented for generating sub-graphs/sub-tree that utilized resources in a better way (i.e., decreasing processing time along with improved memory management).

A. Memory and I/O efficient RSMT routing design

This division presents generation of memory and I/O efficient RSMT (MIOERSMT). This MIOERSMT design is intended to reduce memory requirement, computation time as well as wire-length. This approach initially divides the tree as sub trees by considering available memory, as analogous to [6] later, considers Memory and spanning tree to be the input. First, the model initializes one node as a root node by calculating the smallest overhead edges (i.e.) using memory optimized spanning graph. A parent node is the one which is nearer to a root node. Optimization of memory for larger net size is obtained, by realizing a child-parent association through the edges, and later applying depth-first search and divide and conquer technique. For the given graph $H(N, M)$, in which M and N indicates a group of ordered pairs of edges and nodes correspondingly. Let $m = |E|$ depicts group of edges and $n = |V|$ represent node set. Initially a spanning graph H is constructed and then modified by adding Steiner nodes α which denotes that it is connected to all nodes in H as shown in **Algorithm 1**. Later, the divide-conquer technique is applied for generating Memory optimized tree of graph H using **Algorithm 2**. Finally, a merging algorithm is presented in **Algorithm 3**.

Algorithm 1: Memory Optimized based Rectilinear Steiner Minimum Tree Construction

Step 1. Start

Step 2. Input a graph H and memory S
Step 3. For all nodes $n \in N(H)$ using parallel computing environment
Step 4. **Cumulate** edge (α, v) in G where α is a Steiner nodes
Step 5. Get Divide & Conquer(H, G, S)
Step 6. Process Divide & Conquer (graph H, Memory S and tree G)
Step 7. If$|H| \leq = S$
Step 8. **Get** Memory Optimized Tree G using Memory optimization requirement
Step 9. **Set** dividend to **false**
Step 10. While (dividend = **false**)
Step 11. **Update** spanning tree (G, update) to obtain Memory optimized tree (H, G, S)
Step 12. **If** Update spanning tree is **false**
Step 13. **Obtain**G
Step 14. **Obtain** legal dividend of memory optimized spanning graph
$\left((H_1, H_2, \dots, H_p), (G_0, G_1, G_2, \dots, G_d), \alpha\right)$
Step 15. **If** $d > 1$
Step 16. **Set** dividend to **true**
Step 17. For $q = 1; q \leq d; q++$ using parallel computing environment
Step 18. **Set**G_q using Divide & conquer (H_q, G_q, S)using **algorithm 2**
Step 19. Compute Memory optimized G by merging $((G_0, G_1, \dots, G_d), \alpha)$ using **algorithm 3**
Step 20. ObtainG
Step 21. Stop

Algorithm 2: Memory Optimized Division Algorithm

Step 1. Start
Step 2. Input a graph H, tree G and memory S
Step 3.$\overline{G}_0 =$ a partitioned tree of G
Step 4. $\forall$ edge$(a, b) \in M(H)$ in serial order on disk
Step 5.$u =$ the $\mathcal{A}$ of x and y in G
Step 6. **If** (x, y) is a cross-edge &u is a non-leaf node in $N(\overline{G}_0)$
Step 7.$(u_x, u_y) = \bar{\mu}(x, y)\&M(\mu) = M(\mu) \cup \{(u_x, u_y)\}$
Step 8. If $(\mu! = \mathcal{D})$
Step 9. $\forall\, \mathcal{T}$ in μ
Step 10. **If**$|\mathcal{T}| > 1$
Step 11. **Adjust**G&μ adopting node contraction process w.r.t. $\mathcal{T}$
Step 12.$a_0 =$ root of G, $G_0 = \emptyset \& \mathcal{Q} = \emptyset \ \& \mathcal{Q}.\text{add}(a_0)$
Step 13. While $\mathcal{Q}! = \emptyset$

Step 14. $x = Q.\textbf{remove()}$
Step 15. **If** $x! =$ Steiner node
Step 16. $\forall$ child node y of x in G
Step 17. **If** $y \in N(\overline{G}_0)$
Step 18. **Add** edge (x, y) into G_0 , $Q.\text{add}(y)$
Step 19. delete nodes that are not in $N(G_0)$ and their respective edges from μ
Step 20. $\{a_1, a_2, a_3, \dots, a_d\}$ = the leaf node of a_0 in G_0
Step 21. $\textbf{For}(q = 1; q < d; q++)$
Step 22. G_q = the subtree rooted at a_q in G
Step 23. $\textbf{For}(q = 1; q < d; q++)$
Step 24. H_q = the subgraph induced by nodes in G_q
Step 25. Obtain $(H_0, H_1, H_2, \dots, H_d), (G_0, G_1, G_2, \dots, G_d), \mu$
Step 26. Stop

Algorithm 3: Memory Optimized Merging Algorithm.

Step 1. Start
Step 2. Input a (subtree$G_0, G_2, G_3, \dots, G_d, \mu$)
Step 3. $G = G_0$
Step 4. Topological sort all nodes in μ
Step 5. Reorder all nodes in G based on reverse topological order of correspondent node in μ
Step 6. $\textbf{For}(q = 1; q < p; q++)$
Step 7. **Merge** G_q into G
Step 8. $\forall$ Steiner node $y \in N(G)$
Step 9. x = root node of y in G
Step 10. $\forall$ leaf node u of y in G
Step 11. Eliminate (y, u) from G& add(x, u) into G
Step 12. Obtain G
Step 13. Stop

The **algorithm 1** memory optimized divide and conquer technique considers as its input Memory S, Spanning graph G of H and graph H . The resultant tree G obtained happens to be a depth first search tree ofH and this tree G is stored back at memory then H placed on disk. This model initially calculates whether the graph Hwill fulfill memory optimization requirement, thus H can be accommodated to memory S. Later by applying the memory optimization scheme a Memory optimized tree G of H is evaluated so that G is obtained. In case if some G is not obtained, the approach further continues the computation of Memory optimized tree G of Hby partitioning Memory optimized tree G applying divide and conquer technique. Then, using **algorithm 2**, the model maximizes the amount of sub-graphs. In existing model the division is performed on the spanning tree G and graph H by making use of structure G_0 with same parent as G. It leads to less divided

sub-graph and degrades I/O efficiency, thus, affecting overall processing time. Finally, the **algorithm 3** takes input, divided tree $G_0, G_{1,}G_2, \dots, G_d$ and the corresponding μ and outputs a graph G. Following two issues must be solved to do merging operation in **algorithm 1**. First is how to arrange $G_0, G_{1,}G_2, \dots, G_d$ as the combined tree G, such that G is a tree of graph H. Second is the way to process the Steiner node in tree$G_0, G_{1,}G_2, \dots, G_d$. More details of algorithm explanation can be obtained from [17][37].

B. High performance environment for parallel job execution model

This segment demonstrates the design of novel parallel execution platform that works on multicore environment using cache memory. In this multicore parallel execution (MPE) framework, there are different modes in which the jobs may operate. Initially all the sub jobs in a tree or graph will be in locked mode. If the sub jobs dependency bounds are satisfied then it changes to accessible or available mode. After this the sub jobs enters into steady or processing mode as long as the memory limits are reached or satisfied. In case the memory prerequisite is not met then the sub job enters the waiting mode. Later, it stays in this mode until required memory becomes available, and then changes to accessible mode. Applying this procedure helps in ensuring that the memory prerequisite of sub-jobs are met and is executed by a core regardless of job size. During execution mode the sub-job basically reads an input data and then begins processing jobs till it is completed. Once the processing is completed the outputs are stored and it changes to completed mode. After this the memory occupied by the sub job is released that can utilized again by succeeding sub jobs.

C. Limits modelling in High perforamnce computing parallel job execution model

This division present a design of limit modeling for parallel makes pan times (MT) of job trees or graphs. First we distinguish the limit parallel job makes pan times based on processing cores. Foremost challenge is detecting the total active cores that are constrained by memory requirement during the course of execution and to utilize the maximum limits of the makes pan times of job trees. But, memory availability is not fixed and is dynamic in nature particularly in view of heterogeneous and composite nature of memory requirements in the job trees. In order to resolve the above specified problems, the proposed research calculated the bounds of parallel jobs (BPJ). This is accomplished by computing for an arbitrary tree or graph, a minimum and maximum bound to the peak possible number of parallel jobs which are processed at same instance of time within the estimated memory requirement. After this we compute the maximum bounds to the expected makespan time of the job trees. Finally a memory constrained model is designed that is cost-effective and also allows job trees to make efficient use of memory resource and thus increases the overall performance of job trees.

D. Bounding of parallel jobs with enhanced resource utilization

Since there are huge number of parallel sub-jobs in a job tree with each having different memory requirements, it essentially becomes very hard to schedule the tree so that they hold effective makespan time while ensuring effective usage of memory. Therefore an analysis is made for finding the way in which the bounding of maximal number of parallel sub-jobs for scheduling is optimized when there is constraint on memory. The algorithm then allocates the available memory immediately instead of preserving to the

subsequent sub-jobs. The bounding of parallel jobs help in improved exploitation of the resource, which results in obtaining improved tradeoffs amongst processing time and requirement of memory resource. Further for attaining BPJ we minimize and maximize limitations on peak number of parallel sub jobs while the job tree is being processed without dividing memory of size N. These two bounds are gained by analyzing each clique in the derived tree (DT) $H(W, F)$, represented by$H_e(W, F_e)$, which is constructed by supplementing/augmenting $H(W, F)$ (i.e., an edge is appended amongst node u and w if there are no directed path between them) then weight of node is reassigned or optimized by $Opt(w), w \in W$. Assume D as a clique in a well- linked component $\mathcal{D}$ of the DT, such that minimal for $\mathcal{D}$ are estimated by resulting equation:

$$i_{\downarrow}(N, \mathcal{D}) = \min_{D \in \mathcal{D}} \left\{ |T(D)| : T(D) = \arg \min_{T \subseteq D\downarrow} \left\{ \left| N - \sum_{j=1}^{|T|} n_j \right| \right\} \right\} \quad (1)$$

and maximal limits for $\mathcal{D}$ by applying the equation mentioned

$$i_{\uparrow}(N, \mathcal{D}) = \max_{D \in \mathcal{D}} \left\{ |T(D)| : T(D) = \arg \min_{T \subseteq D\uparrow} \left\{ \left| N - \sum_{j=1}^{|T|} n_j \right| \right\} \right\} \quad (2)$$

where $n_j = Opt(j)$ is memory prerequisite of job j in T, $T(D)$ depicts the selected subgroup of D, whose memory perquisite utilize within N. Nevertheless, the method for selection is dependent on bound computation.

The minimal and maximal bounds on BPJ for each connected component can be used to assess the BPJ within memory limits N for an established tree $H_e(W, F_e)$ using following equation

$$BPJ = [i_{\downarrow}(N), i_{\uparrow}(N)], \quad (3)$$

where $i_{\downarrow}(N) = \min_{\mathcal{D} \in \mathbb{E}} \{i_{\downarrow}(N, \mathcal{D})\}$ and $i_{\uparrow}(N) = \max_{\mathcal{D} \in \mathbb{E}} \{i_{\uparrow}(N, \mathcal{D})\}$, in which $\mathbb{E}$ is a cluster of connected component of the DT.

For obtaining optimum computation the proposed work calculates the maximal limit applying dynamic programming approach which means that this technique allocates the existing memory size N amongst parallel sub-jobs in the graph. Specifically, an assigned memory limit d, $0 < d \leq N$, have successive repetition in order to calculate the highest number of parallel sub-jobs that could be processed at same instance, specified as $\mathcal{O}[v, d]$, the course of developing the sub-graph positioned at node w is computed as shown

$$\mathcal{O}[v, d] = \max \left\{ q_v(d), \sum_{\sum_{w \in Out(v)} n_w \leq d} \mathcal{O}[w, n_w] \right\}, \quad (4)$$

where

$$q_v(n) = \begin{cases} 0 & \text{if } n \leq n_v \\ 1 & \text{Otherwise} \end{cases} \quad (5)$$

is a technique for finding the number of cores required for node v for a memory of size n. This proves that $\mathcal{O}[v, d]$ is performed by allocating the memory to sub-graphs, sub-trees or to node v, whichever provides more parallelization of sub-jobs.

For solving Eq. (5), the universal tree graph (TG) is transformed into binary representation. This transformation is done by cumulating two duplicate nodes $\bar{v}$ and $\bar{\bar{v}}$ iteratively using following equation

$$\begin{cases} \mathcal{O}[v, d] = \max\{q_v(d), \mathcal{U}[\bar{v}, d]\} \\ \mathcal{U}[\bar{v}, d] = \max\limits_{0 \le r \le d}\{\mathcal{O}[w, r] + \mathcal{U}[\bar{\bar{v}}, d - r]\} \\ \mathcal{U}[\emptyset, d] = 0 \end{cases} \quad (6)$$

Where w is v's and also$\bar{\bar{v}}$'s left most child sub-jobs in the TG. $\mathcal{U}[\bar{v}, d - r]$ Depicts maximal amount of parallel sub-jobs within entire sub-graph of node v kept by enduring memory capability (i.e., size) of $d - r$. Preliminarily, left-most sibling of v is allocated size r. Post that its entire child sub-graph, which connects duplicate node to utilize leftover memory of $d - r$. For easing algorithm execution the duplicated nodes are introduced. Further, they don't use any memory. Instead, the algorithm evaluates two matrices $\mathcal{O}[v, d]$ and $\mathcal{U}[v, d]$, respectively, to finish the estimation, and lastly, $\mathcal{O}[root, N]$ is the intended strategy to $i_{\uparrow}(N)$ for the TG. Similarly, this work estimates $i_{\uparrow}(N)$ by searching least amount of parallel sub-jobs (i.e., big) in the graph to fully utilize the memory as possible, rather than searching for maximal amount of parallel sub-jobs (i.e., small).

Two matrices $\mathcal{O}$ and $\mathcal{U}$ is created alternatively for estimating Eq. (5) each possessing size of $o(nN)$. Every initialization of $\mathcal{O}$ requires a fixed cost while every initialization of $\mathcal{U}$ requires at least N cost because of $\max_{0 \le r \le d}$ computation when d is maximal of N. Therefore, the time complexity order of Eq. (5) is $O(nN^2)$. Thus, we attain optimal solution within polynomial bound for n-node TG.

E. Bounding schedule of parallel job execution

Given any job tree that is to be processed on multi-core framework having q cores, the makespan time of parallel processing the job tree remain limited. Now, consider the makespan time of a job tree on single core environment be U_1 (i.e., $q = 1$) and makespan time on multi-core framework is denoted as U_∞ (i.e., $q = \infty$). Hence, the parallel makes pan of a tree is constrained on available processing core $q \in [1 + \infty]$ which is as mentioned

$$U_q \le \frac{U_1}{q} + U_\infty \quad (7)$$

The above result is developed by considering workload that is of static size which means that total processor cores increases as the job size increases. In addition the memory size is also considered to be boundless (that is memory limitations are not considered). Besides, adapting BPJ helps in improving the result when memory requirement is constrained by N, the reason for this is because the number of processing core that can be used is around$[i_{\downarrow}(N), i_{\uparrow}(N)]$. Thus, we achieve subsequent result conforming to probable parallel makespan time of the job tree. Now consider a job tree that U_n and U_∞ describe corresponding MT with sequential memory specified by

$$N = n \quad (8)$$

and parallel MT without memory requirement constraint is as mentioned

$$N = +\infty, \quad (9)$$

Later, the model can perform bounding of predictable MT of the job tree when available memory is constrained by

$$N \in [n, N_{\uparrow}], \quad (10)$$

where $N_{\uparrow}$ is depicted as the minmax memory requirement of the tree calculated by the below mentioned equation

$$U_{\uparrow N} \leq \frac{\ln i_{\uparrow}(N)}{i_{\uparrow}(N) - i_{\downarrow}(N)} U_n + U_{\infty} \quad (11)$$

Now, consider that number of parallel sub-jobs is assigned in uniform manner within$[i_{\downarrow}(N), i_{\uparrow}(N)]$, for one of$q \in [i_{\downarrow}(N), i_{\uparrow}(N)]$. Thus the model possess $U_q \leq \frac{U_n}{q} + U_{\infty}$ as per Eq. (7). Thus,

$$\frac{\sum_{q \in [i_{\downarrow}(N), i_{\uparrow}(N)]} U_q}{i_{\uparrow}(N) - i_{\downarrow}(N)} \leq \frac{\sum_{q \in [i_{\downarrow}(N), i_{\uparrow}(N)]} \left\{\frac{1}{q} U_q\right\}}{i_{\uparrow}(N) - i_{\downarrow}(N)} + U_{\infty} \quad (12)$$

As the n^{th} frequency total is massive as the logarithm value of n, then for this case we say Eq. (11) holds. Further, the actual memory used during execution phase depend on the order it is processed where U_n is realized adapting maximal requirement of sub-job with certain contextual environment. In actual environment, it can be established that $i_{\downarrow}(N)$ is identical to $i_{\uparrow}(N)$ when N is small. Hence, we obtain$U_n \leq \frac{U_1}{i_{\downarrow}(N)} + U_{\infty}$.

Algorithm 4: Performance and memory constraint parallel computation (PMCPC) algorithm

Steps

1. Start
2. Execute PMCPC$(\mathbf{H}, \mathbf{N})$
3. $R \leftarrow \emptyset$ // queue R is created
4.$R.\,\mathrm{add}(w_o)$ // source is added to queue R
5. Event PMCPC:
6.while$(R \neq \emptyset)$**do**
7. $w \leftarrow R.\,\mathrm{remove}()$ //Element is remove from queue R and moved to w
8. **if**$(n_w \leq N)$**then** //w can be accepted for computing
9.$N \leftarrow N - n_w$ //Updating the existing N value
10. **else**
11.$w.\,\mathrm{resetq}(w, r)$ //Resetting the priority or selectivity
12.$R.\,\mathrm{add}(w)$ //Restricted parallelization
13. **end if**
14. **if** (items in R are not altered) **then**
15. **return** //Obtained N becomes inadequate
16. **end if**
17. **end while**
18. **event**w**completed:**
19.$N \leftarrow N + N_w$ //w freed
20. For$(\forall v \in w.\,\mathrm{outedge}())$**do** //assignment of out edges
21. $f.\,\mathrm{setidentifier}(d)$ // Every edge is defined an identifier
22. For$(\forall f \in V$ where $f \in v.\,\mathrm{inedge}())$ **do**
23. If$(\forall f \in v.\,\mathrm{inedge}()$ is identified$)$ **then**

24. v. $\mathrm{setq}(\alpha_v, u_v^f, u_v^u, \mathrm{degr}, \mathrm{cent})$
25. R. $\mathrm{add}(v)$
26. **End of if**
27. **End for**
28. **End for**
29. End process
30. Stop

The proposed novel algorithm not only solves the problem of memory limitations but also produces decreased makes pan time (i.e., find shortest scheduling lengths) by adopting depth first search (DFS) as well as breath first search (BFS) techniques. Applying, BFS aid in enhancing memory and parallel efficiency and DFS works in contrary. In addition, each sub jobs $w \in W$ has varied proportion $\alpha_w$that can be computed as follows

$$\alpha_w = \frac{\sum_{j \in In(w)} T_{in}(j)}{\sum_{j \in Out(w)} T_{out}(j)} \quad (13)$$

The objective of the novel parallel computing framework is to apply above mentioned data for selecting the sub jobs in the list that are prepared to initiate in such a way that fetches better tradeoff amongst memory efficiency and parallelization. This novel technique comprises of two main events. The PMCPC event consists of while loop, used for handling PMCPC event in which each ordered sub-job is added to queue R later is selected by a processing core only if the memory requirements are satisfied. If the memory requirements are not met then the sub job reenters the queue R by modifying the selectivity parameter. One of the salient features of this novel technique is the function v. $\mathrm{setq}(\alpha_v)$ and w. $\mathrm{resetq}(w, r)$, that can leverage various searching methodologies in the course of scheduling. In our novel design the selectivity parameter of sub-job nodes in R is set based on their α_w in a top-down manner (i.e., $\alpha_{w_1} \geq \alpha_{w_2} \geq \cdots$. Further, degr_w depicts the degree of node w within $H(W, F)$, cent_w represents the centrality metric of node w in $H(W, F)$. The centrality metric specifies the total time that node w acts as a connector in the shortest path amongst nodes. The technique basically is designed towards processing a job tree or scheduling the graph using memory limitations which satisfies the requirements of the jobs bounds. Also, the algorithm is hybrid in nature i.e., it can leverage both DFS and BFS aimed at attaining good parallelization with memory requirement. Once the sub job processing is complete, the memory n_w is freed and then added toN, also the output edges are assigned an identifier. Further, for each sibling of sub-job w, they are included into R with predefined selectivity parameter only if its entire input edges are assigned identifiers, so that the corresponding input data becomes accessible. A salient feature in **algorithm 4.4** is that the events runs, autonomously and coordinates among themselves, by creating and exploiting the N memory blocks by means of sub-jobs provision and de-provision.

$U_n \leq \frac{U_1}{i_\downarrow(N)} + U_\infty.$

III. EXPERIMETAL RESULT AND ANALYSIS

The segment depicts the experimental assessment of the outcome attained of the novel memory and I/O efficient RSMT generation technique on high performance computing environment compared with the existing routing technique. The design developed is realized adapting C++ programing language on eclipse IDE. The implementation framework used for evaluating the novel approach is Centos 7.0 operating system (OS), 2.3 Ghz, intel I-5 processor that comprises four logical core with 12 GB RAM. This section compares the performance evaluation of proposed model to existing model [6] with respect to wire-length and execution time. For evaluating, this model considers the IBM benchmark [18] the details of these benchmarks as shown in Table 1.

TABLE I. **BENCHMARK DETAILS OF VLSI CIRCUIT**

Benchmark Circuit Case	**Number of Net**	**maximum degree**	**Average Degree**
IBM1	14111	42	3.58
IBM2	19584	134	4.15
IBM3	27401	55	3.41
IBM4	31970	46	3.31
IBM5	28446	17	4.44
IBM6	34826	35	3.68
IBM7	48117	25	3.65
IBM8	50513	75	4.06
IBM9	60902	39	3.65
IBM10	75196	41	3.96
IBM11	81454	24	3.45
IBM12	77240	28	4.11
IBM13	99666	24	3.58
IBM14	152772	33	3.58
IBM15	186608	36	3.84
IBM16	190048	40	4.10
IBM17	189581	36	4.54
IBM18	201920	66	4.06
Average	106299	134	3.92

A. Wire length and memory perforamcne evaluation

As discussed in previous section the experiments are evaluated using ISPD 98 benchmark as specified in Table I. The Table II shows the resultant wire length and memory usage by the novel technique and existing FLUTE routing model. From outcome attained it is observed that an average 0.026% reduction in wire length is accomplished by proposed MIOERSMT compared to existing FLUTE routing model. Similarly, an average of 77.71% reduction in memory usage is attained by proposed MIOERSMT as compared to existing FLUTE routing model. Therefore, the proposed novel approach is proved scalable in accordance to minimizing wire length and decrease in memory usage when compared with standard FLUTE VLSI routing design.

TABLE II. **WIRELENGTH AND MEMORY PERFORAMNCE EVALUATION**

Benchmark Circuit	MIOERSMT		FLUTE [6]	
	Wire length	Memory usage	Wire length	Memory usage
IBM1	444307	109264	444553	398908
IBM2	527382	168457	527641	713451
IBM3	761993	180996	762276	711893
IBM4	855986	211003	856273	723607
IBM5	2809615	224661	2810816	1188898
IBM6	494144	244577	495969	970098
IBM7	994978	337674	995265	1248799
IBM8	944096	414,185	944382	2078590
IBM9	1260914	411154	1261199	1595039
IBM10	3190871	524740	3191615	2239228
IBM11	1898961	531886	1899367	1789611
IBM12	2914884	543178	2915521	2407032
IBM13	2450087	659237	2450577	2551928
IBM14	3180260	1030486	3180777	3801149
IBM15	2922395	1284495	2922778	5603960
IBM16	3500272	1348313	3500776	5732675
IBM17	5368916	1414806	5369659	6804778

IBM18	2145856	1627262	2146128	6982462
Average	**2036995.389**	**625,910**	**2037531.778**	**2,641,228**

B. Runtime perforamcne evaluation

Experimentations are performed on ISPD 98 benchmark as specified in Table I. The resultant runtime achieved by the novel routing design and existing routing design [6] is shown Table III. From outcome attained it is observed that average of 32.63% runtime reduction is attained by proposed MIOERSMT compared to existing FLUTE routing model. Further, experiment is conducted under HPC computing framework by varying the number of parallel computing cores. From outcome attained, it is observed that an average of 49.34% and 79.38% runtime reduction is attained by novel MIOERSMT design compared to existing FLUTE routing design considering 2 processing core and 4 processing core, respectively. Therefore, the proposed novel approach is scalable using parallel processing core in terms of reducing runtime

TABLE III. **RUNTIME TIME PERFORAMNCE EVALUATION**

Benchmark Circuit Case	**FLUTE [6]**	**MIOERSMT**		
		Single core	**with 2 core**	**With 4 core**
IBM1	180000	90000	48000	28900
IBM2	220000	130000	69150	39400
IBM3	267000	163700	80100	45925
IBM4	269000	161100	83400	39275
IBM5	870000	410000	232000	112500
IBM6	277000	183000	89000	53750
IBM7	298800	193000	122500	44250
IBM8	320000	250000	143000	71500
IBM9	322000	219000	101000	64750
IBM10	347000	260000	123000	77000
IBM11	390000	298000	158000	70500
IBM12	510000	390000	205000	99900
IBM13	570000	410000	195150	122500
IBM14	640000	520000	240300	149000
IBM15	651000	517000	277000	120250

IBM16	690000	587000	312000	149750
IBM17	1780000	1190000	512000	247500
IBM18	1880000	1090000	395000	212500
Average	**582322.2222**	**392322.2222**	**188088.8889**	**97175**

IV. CONCLUSION

This paper conducted a deep rooted analysis for addressing problem of RSMT for modern VLSI circuit. From analysis it is seen that existing method used FLUTE for routing in PD construction. However, usage of FLUTE induced memory and I/O overhead resulting in increase of overall processing cost and processing time. Therefore, it is important to minimize I/O and memory overhead while constructing RSMT for which this paper presented a memory and I/O efficient RSMT construction. The MIOERSMT uses depth-first search and divide and conquer approach to build a Memory and I/O optimized tree. Further, a parallel computing environment design is presented to reduce runtime of RSMT construction in parallel manner under HPC environment (i.e., both CPU and GPU). However, GPU based model induces high deployment cost and requires efficient memory management method. Hence, the work presents a Performance and memory constraint parallel computation (PMCPC) algorithm. The model allows us to exploit the computational power from these shared-memory multi-core HPC systems. Experiments are conducted on small and long nets using ISPD 98 benchmarks for evaluating performance of proposed and existing VLSI routing method. The performance is evaluated in terms of wire length (WL), memory utilization and runtime in sequential (i.e., with single core) as well as in parallel (i.e., using 2 cores and 4 cores). The experimental outcome shows that the proposed VLSI routing model reduces wire length and memory utilization by 0.026% and 77.71% when compared to the existing VLSI routing model, respectively. Further, the proposed VLSI routing model reduces runtime by 32.63% when compared with existing VLSI routing model. Then, the proposed parallel VLSI routing model attain a computation speedup of 49.34% and 79.38%considering 2 cores and 4 cores when compared with proposed sequential (i.e., with single core) VLSI routing model, respectively. From overall results attained it can be seen that the proposed routing model is robust irrespective of wire length, memory and computation performance when compared with existing VLSI routing model. Future work would consider performance evaluation considering higher number of cores and changing the memory size.

References

[1] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: Freeman, 1979.

[2] H. Chen, C. Qiao, F. Zhou, and C.-K. Cheng, “Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction,” in Proc. ACM Int. Workshop Syst. Level Interconnect Prediction, 2002, pp. 85–89.

[3] Chris Chu. FLUTE: Fast lookup table based wirelength estimation technique. In Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design, pages 696–701, 2004.

[4] Chris Chu and Yiu-Chung Wong. Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design. In Proc. Intl. Symp. on Physical Design, pages 28–35, 2005.

[5] Wong, Yiu-Chung, and Chris Chu. "A scalable and accurate rectilinear Steiner minimal tree algorithm." VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on. IEEE, 2008.

[6] Chu, Chris, and Yiu-Chung Wong. "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27.1 (2008): 70-83.

[7] Hao Zhanga, Dong-yi Yea, Wen-zhong Guo ”A heuristic for constructing a rectilinear Steiner tree by reusing routing resources over obstacles” Volume 55, Pages 162–175, 2016.

[8] P. P. Saha, S. Saha and T. Samanta, "An efficient intersection avoiding rectilinear routing technique in VLSI," 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, 2013, pp. 559-562.

[9] G. Ajwani, C. Chu and W. K. Mak, "FOARS: FLUTE Based Obstacle-Avoiding Rectilinear Steiner Tree Construction," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 2, pp. 194-204, Feb. 2011.

[10] K. Ma, Q. Zhou, Y. Cai, C. Zhang and Z. Qi, "A Steiner tree construction method for flexibility and congestion optimization," 2013 International Conference on Communications, Circuits and Systems (ICCCAS), Chengdu, 2013, pp. 519-523.

[11] T. H. Wu, A. Davoodi and J. T. Linderoth, "GRIP: Global Routing via Integer Programming," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 1, pp. 72-84, Jan. 2011.

[12] M. Pan, Y. Xu, Y. Zhang, and C. Chu, “FastRoute: An efficient and high-quality global router,” VLSI Design, vol. 2012, 608362, 2012.

[13] Umair F. Siddiqi, Sadiq M. Sait, and Yoichi Shiraishi, “A Game Theory-Based Heuristic for the Two-Dimensional VLSI Global Routing Problem,” Journal Of Circuits Systems And Computers, vol. 24, no. 6, 2015.

[14] Umair F. Siddiqi, and Sadiq M. Sait, “A Game Theory Based Post-Processing Method to Enhance VLSI Global Routers,” IEEE Access, vol. 5, pp. 1328–1339, 2017.

[15] Chow, Wing-Kai & Li, Liang & F.Y. Young, Evangeline & Sham, Chiu-Wing. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach. Integration, the VLSI Journal. 47. 105–114. 10.1016/j.vlsi.2013.08.001, 2014.

[16] A. Momeni, P. Mistry and D. Kaeli, "A parallel clustering algorithm for placement," Fifteenth International Symposium on Quality Electronic Design, Santa Clara, CA, 2014, pp. 349-356.

[17] Latha, N.R & Prasad, G.R.. Wirelength and memory optimized rectilinear steiner minimum tree routing. 1493-1497, 2017. 10.1109/RTEICT.2017.8256846.

[18] C. J. Alpert, “The ISPD98 circuit benchmark suite,” in Proc. Int. Symp. Phys. Des., 1998, pp.80–85. [Online]. Available: http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html

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

[20] Y. Cai, C. Deng, Q. Zhou, H. Yao, F. Niu and C. N. Sze, "Obstacle-Avoiding and Slew-Constrained Clock Tree Synthesis With Efficient Buffer Insertion," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 1, pp. 142-155, Jan. 2015.

[21] C. H. Liu, C. X. Lin, I. C. Chen, D. T. Lee and T. C. Wang, "Efficient Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Geometric Reduction," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 12, pp. 1928-1941, 2014.

[22] Held S, Spirkl S T. A fast algorithm for rectilinear Steiner trees with length restrictions on obstacles. Proceedings of the 2014 on International symposium on physical design. ACM, 37-44, 2014.

[23] M. Wang and Z. Li, "A Spatial and Temporal Locality-Aware Adaptive Cache Design With Network Optimization for Tiled Many-Core Architectures," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 9, pp. 2419-2433, Sept. 2017.

[24] S. He, Y. Wang, X. Sun and C. Xu, "Using MinMax-Memory Claims to Improve In-Memory Workflow Computations in the Cloud," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 1202-1214, 1 2017.

[25] Wang, Run-Yi & Pai, Chia-Cheng & Wang, Jun-Jie & Wen, Hsiang-Ting & Pai, Yu-Cheng & Chang, Yao-Wen & Li, James & Jiang, Jie-Hong. Efficient Multi-Layer Obstacle-Avoiding Region-to-Region Rectilinear Steiner Tree Construction *. 1-6, 2018.

[26] Panda S.K., Panda D.C. (2018) Developing High-Performance AVM Based VLSI Computing Systems: A Study. In: Pattnaik P., Rautaray S., Das H., Nayak J. (eds) Progress in Computing, Analytics and Networking. Advances in Intelligent Systems and Computing, vol 710. Springer, Singapore.

[27] Jiangong Song, Qinyong Li and Shilong Ma, " Toward Bounds on Parallel Execution Times of Task Graphs on Multicores with Memory Constraints" in IEEE Access , Volume 7, April 2019.

[28] Y. Han, K. Chakraborty, and S. Roy, "A global router on GPU architecture," in IEEE International Conference on Computer Design (ICCD), pp. 1–6, 2013.

[29] H. Shojaei, A. Davoodi, and J. Linderoth, "Congestion analysis for global routing via integer programming," in IEEE International Conference on Computer-Aided Design (ICCAD), pp. 256–262, 2011.

[30] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra. PaRSEC: Exploiting heterogeneity for enhancing scalability. Computingin Science & Engineering, 15(6):36–45, 2013.

[31] E. Agullo, P. R. Amestoy, A. Buttari, A. Guermouche, J. L'Excellent, and F. Rouet. Robust memory-aware mappings for parallel multifrontal factorizations.SIAM J. Scientific Computing, 38(3), 2016.

[32] G. Aupy, C. Brasseur, and L. Marchal. Dynamic memory-aware task-tree scheduling. In Proceedings of the International Parallel and DistributedProcessing Symposium(IPDPS), pages 758–767. IEEE, 2017.

[33] M. Jacquelin, L.Marchal, Y. Robert, and B. Uçar. On optimal tree traversals for sparse matrix factorization. In Parallel & Distributed Processing Symposium(IPDPS), 2011 IEEE International, pages 556–567. IEEE, 2011.

[34] M. Sergent, D. Goudin, S. Thibault, and O. Aumage. Controlling the memory subscription of distributed applications with a task-based runtime system. In Proceedings of the International Parallel and Distributed Processing SymposiumWorkshops, pages 318–327. IEEE, 2016

[35] K. Ma, Q. Zhou, Y. Cai, C. Zhang and Z. Qi, "A Steiner tree construction method for flexibility and congestion optimization," 2013 International Conference on Communications, Circuits and Systems (ICCCAS), Chengdu, 2013, pp. 519-523.

[36] Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F.M., Weikum, G.: STAR:Steiner-tree approximation in relationship graphs. In: Proc. of IEEE InternationalConference on Data Engineering, Washington DC, USA, 868{879, IEEE (2009).

[37] Latha N.R., Dr. G.R. Prasad, “Performance and Memory Efficient High Performance Computing Framework for VLSI Routing” Progress in Computing, Analytics and Networking, pp.369-381, March 2020(springer)