A GPU based maximum common subgraph algorithm for drug discovery applications

P. B. Jayaraj, K. Rahamathulla and G. Gopakumar
Department of Computer Science and Engineering
National Institute of Technology Calicut, India

***Abstract*—The maximum common subgraph of two graphs, G_1 and G_2, is the largest subgraph in G_1 that is isomorphic to a subgraph in G_2. Finding the maximum common subgraph of two given graphs is known to be a NP-complete problem. An exact solution for the maximum common subgraph problem can be found by an algorithm that transforms the maximum common subgraph problem into a maximal clique enumeration problem. However, as the size of the graph increases, the solution space of the maximal clique enumeration problem increases combinatorially. A serial solution to the computationally intensive problem of complete maximal clique enumeration is tedious. This paper presents a parallel approach using Graphic Processing Unit to compute the maximum common subgraph of the given graphs. The parallel procedure achieves more than tenfold improvement in computational performance. As an application of the proposed parallel maximum common subgraph algorithm, two new tools; LIGANDMATCHER and GRAPHSCREEN are developed. These tools can be used to narrow down the large ligand search space to a small number in the screening phase of drug discovery process.**

***Index Terms*—Maximum Common Subgraph, Maximal Clique Enumeration, GPU, CUDA, Ligand Matching, Virtual Screening**

I. Introduction

GIVEN two graphs G_1 and G_2, the subgraph isomorphism [1] problem is to determine whether a graph G_1 contains a subgraph that is isomorphic to a subgraph of G_2. Subgraph isomorphism is a generalization of the graph isomorphism problem, which checks whether G_1 is isomorphic to G_2. The subgraph isomorphism problem can be reduced to problems like maximal clique enumeration problem and Hamiltonian cycle problem, and is therefore NP-complete.

The property of structural similarity between two graphs can be used for determining the similarity between structured objects. If the subgraphs of two graphs are isomorphic, then they are called common subgraphs. Maximum Common Subgraph (MCS) is a common subgraph between two graphs which has the maximum number of edges. Finding the maximum common subgraph of two graphs is important, as it has practical applications in many problems such as comparison of protein secondary structure, identification of changes in a chemical reaction, finding of co-expressed genes, pattern recognition, ligand matching and virtual screening [2], [3].

The worst case time complexity of finding MCS is exponential and is computationally difficult to solve. But because of its wide applicability, considerable effort was invested into finding algorithms and heuristics which could reduce the total search space [4]. The most common technique to find maximum common subgraph of two graphs is based on a backtracking search tree [5]. The search tree is constructed from a special root node. A traversal from the leaf node to the root of the tree will result in a common subgraph [6], [7]. In order to prevent the search tree from growing unnecessarily large, different refinement procedures and pruning techniques have been used [8].

Another method uses the reduction of maximum common subgraph problem into the Maximal Clique Enumeration (MCE). Existing parallel algorithms [9] to solve maximal clique enumeration problem involve decomposition of the search tree into search subtrees. These algorithms distribute the various section of the problem into different computing elements. Because of the high power utilization and huge investment cost involved in Multiple Instruction Multiple Data (MIMD) machines, Graphic Processing Unit (GPU) based parallel solutions are preferred for maximum common subgraph problem [10].

A. Contribution

The parallel algorithm proposed in this work efficiently manages the data-intensive nature of MCS and has got considerable speedup over an implementation on a SIMD machine. The algorithm developed here is the parallelization of the widely used method of enumerating maximal clique enumeration by Bron and Kerbosh (BK) [11]. The implementation uses CUDA threads to work on GPU. The software is freely available at http://ccc.nitc.ac.in/project/GPUMCS/.

The rest of the paper is organised as follows. Section 2 describes existing graph substructure matching methods and explains the Bron-Kerbosh algorithm for MCE problem in detail. Section 3 describes the proposed GPU based solution to MCE. Section 4 stands for results and discussions. Applications using parallel MCS procedure are also presented in section 4. Conclusions are given in section 5

/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.65

II. BACKGROUND

A. Existing solutions to MCS problem

There are existing works which have transformed the MCS problem into the maximal clique enumeration problem [4]. Consider two undirected labelled graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with n and m vertices, respectively. We can map parts of both graphs onto one another. In this case we say that certain vertices and edges of graph G1 are compatible with certain vertices and edges of graph G2. This compatibility information can be stored in a new graph called *product graph* or *compatibility graph*. In order to find the maximal common subgraph, edge product graph; a variant of product graph is utilized. The edge product graph uses the vertex set $V = E_1 \times E_2$, in which the edge pairs have to coincide with their edge labels and vertex labels. There exists an edge between two vertices (e_1, e_2) and (f_1, f_2), if

- $e_1 \neq f_1$ and $e_2 \neq f_2$
- either e_1, f_1 in G_1 are connected via the vertex of the same label as that of e_2, f_2 in G_2
- e_1, f_1 and e_2, f_2 are not adjacent in G_1 and G_2 respectively.

In order to have a common subgraph in G_1 and G_2, each edge pair has to be compatible with all those edge pairs which form this common subgraph. A clique in the product graph correspond to a maximal common subgraph in G_1 and G_2 [12]. In the edge product graph if $(e_1, f_1), (e_2, f_2), ..., (e_k, f_k)$ represents a clique, then $e_1, e_2, ..., e_k$ of G_1 and $f_1, f_2, ..., f_k$ of G_2 are common subgraphs.

B. Maximal Clique Enumeration

The problem of maximal clique enumeration is to find the clique with largest number of vertices. A simple way is to enumerate all maximal cliques in the graph and select the largest among them. The running time of the algorithm depends upon the number of maximal cliques in the graph. In the worst case the number of maximal cliques will be $3^{n/3}$, where n is the number of vertices in the graph [13]. Bron-Kerbosh is the fastest and commonly used algorithm to solve the maximal clique enumeration problem.

The BK algorithm [11], as in Algorithm 1 finds all cliques in a graph exactly once. The algorithm works on three sets C, P and S. Set C contains the set of vertices belonging to the current clique. Set P contains all vertices which can be used for the completion of C because they are adjacent to the vertex added last to C. In S, all vertices, which can no longer be used for the completion of C are collected, because all cliques containing these vertices are always generated. The algorithm starts with the empty sets C and S. Initially, P includes all vertices of the graph G. During execution, if P and S are empty, a clique is found and will be reported (line 03). Besides each vertex of P is considered in a loop (lines 05 to 11), where the arbitrarily chosen vertex u_i is eliminated from P. P and S are copied into P' and S', respectively (line 07 and 08) for the recursion. The neighbours of vertex u_i are generated and stored in N (line 09). Vertex u_i is added to C and it invokes a recursion with P and S as arguments.

All vertices in C are pairwise adjacent; i.e. C is a vertex set of a complete subgraph. Each vertex in G which is adjacent to all other vertices in C is either in P or in S. Each vertex u belonging to set P is adjacent to all vertices in C. The vertices from P are used for the extension of C. Once a vertex from P is used for the extension of C, it will be moved to C. Each vertex u belonging to set C is adjacent to all the vertices in C. The vertex sets of all cliques containing $C \cup \{u_i\}$ have already been enumerated once. If the function call ENUMERATE-CLIQUES() finishes, the vertex sets of all cliques containing C have been enumerated exactly once.

The recursion tree generated by BK Algorithm for a given graph is shown in Figure 1. The edges of the recursion tree labelled by vertex u, will be added to the current set C. The grey vertex is the root of the recursion tree. Paths from the root to a white leaf vertex describe cliques in G. Paths from the root to a black leaf vertex describe complete subgraphs in G which are not maximal, because they can be extended by at least one vertex from the nonempty set S.

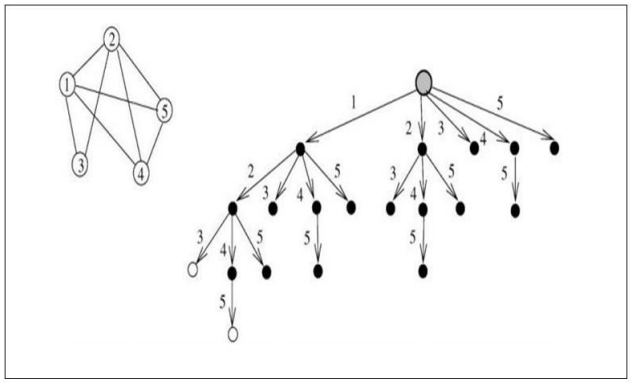

Figure 1: A graph and its recursion tree generated by BK algorithm

C. Existing parallel solutions for Maximal Clique Enumeration

The parallel algorithm presented in [9] is a parallelization of the above described BK algorithm. It includes a decomposition of the BK search tree into independent search subtrees. A

Algorithm 1 Bron-Kerbosh(BK) algorithm[4]

ENUMERATE-CLIQUES(C, P, S)
C: set of vertices belonging to current clique
P: set of vertices which can be added to C
S: set of vertices which cannot be added to C
N: set of neighbours of a node u in G

Let P be the set $\{u_1, u_2, ..., u_k\}$
if $(P = \phi)$ and $(S = \phi)$ **then**
    REPORT-CLIQUE;
else
    for $i \leftarrow 1$ to k **do**
        $P \leftarrow P \setminus \{u_i\}$
        $P' \leftarrow P$
        $S' \leftarrow S$
        $N \leftarrow \{v \in V | \{u_i, v\} \in E\}$
        ENUMERATE-CLIQUES $(C \cup \{u_i\}, P' \cap N, S' \cap N)$
        $S \leftarrow S \cup \{u_i\}$
    end for
end if

data structure called *candidate path structure* is used to store the vertex being visited, the level of the node in the search tree, set P (candidate-list) and set S (not-used-list) [11] in the process of decomposition. The set C (current clique) is stored as a global array in each computing element. The algorithm was implemented using POSIX threads for shared memory access and MPI for distributed memory access. It was tested on a Cray XT4 machine for different biological networks. The parallel algorithm internally decomposes the task of traversing the search tree into multiple threads.

Since the nature of the work in this MCE is more data-intensive, parallelizing the algorithm using SIMD machine will be beneficial. Implementing the parallelized version on a GPU is highly desirable, as it offers a large number of computational cores that are capable of doing floating point computations in parallel. The cost of installing and maintaining such a system is comparatively very low, as the power required for the operation of a GPU is very low. The computational performance obtained by running this algorithm on a GPU is very much comparable to that of a supercomputer [14].

III. The Proposed GPU based clique enumeration algorithm

The proposed GPU based parallel processing method constructs search tree separately at each level by utilizing the massive computing power of GPU. Since the use of recursion is difficult to implement in GPU kernels [15], iterative method is used in our method. The algorithm initiates from a root node, which denotes the lower bound of the optimal solution. In the branch and bound algorithm, the branching is done according to the search strategy adopted by the algorithm [16], [17], [18]. The aforementioned algorithm employs a Breadth First Search (BFS) technique for branching. The main data structure used in the algorithm is ***CurrentActiveSet*** which contains the list of nodes to be evaluated and which have not been branched yet. The bounding techniques are used for the evaluation [19], [20].

The *CurrentActiveSet* is filled serially until there is a sufficient number of nodes to generate considerable amount of threads. GPU can execute large number of threads parallelly where each thread performs simple operations. In this algorithm, each thread selects a node from the *CurrentActiveSet* and then evaluates a small portion of the total search space. Considering the given node as a root, each thread performs a BFS operation on the graph. Each thread must be well aware of the branching and evaluation rules for them. Hence these procedures must be implemented as device functions (kernel functions). The new nodes found by each of the threads will be added into the set of partial solutions. The newly generated nodes are then evaluated by simple procedures. The number of times the kernel is invoked depends upon the enormity of the problem. Each time the kernel is called, proportional number of threads will be launched and the sub problems being analysed will be more confining than the sub problem used in the previous searches.

When the kernel is invoked, each thread will perform operations based on its bounds there by returning the partial solutions to the *CurrentActiveSet*. This *CurrentActiveSet* will be processed by another set of threads in the next kernel invocation. This design has the advantage of segmenting the

search space. It allows the solution space to be evaluated in concurrent pieces and in less time. The serial portion of the implementation starts with a node at level 0, which is a special root node for the branch and bound tree. n Nodes are added to the tree at level 1 representing n one-vertex cliques, where n is the number of vertices of the graph. A node at level x represents an x-vertex clique.

Algorithm 2 presents the proposed data parallel MCE algorithm. The algorithm commences by reading the graph data and the features of the GPU. In the next step an initial *CurrentActiveSet* is populated, which contains all the nodes in the graph. The number of threads and blocks that are required to service the *CurrentActiveSet* is determined. The current active set and graph data are copied to device memory and adequate number of threads will be launched. Here each node i taken from the current active set will become a BFS root and its child nodes are generated by a thread T_i. The thread T_i will produce nodes which are contiguous to node i and then adds it into the search tree in the next level. In our implementation, these new nodes will be added to the current active set atomically (thread by thread). When the threads accomplish their task, a synchronization step is executed to update the tree stored in CPU. The newly generated nodes are then passed on to the device memory and the device functions are called again. Once the *CurrentActiveSet* is empty, the algorithm will terminate. The number of levels of the search space is required to be identified for populating the initial values of the *CurrentActiveSet*. If the number of levels used for the initial set is L, then there will be a maximum of N(N-1)(N-2)...(N-L) nodes present in the *CurrentActiveSet*. Figure 2 depicts the parallel evaluation of the BK search tree. All nodes in the same level are generated in the same level of iteration.

In this implementation we have used device memory for allocating the *CurrentActiveSet*. Considering the vastness of data and nature of parallelism, transferring the code to shared memory does not contribute speed improvement and hence have not been considered in this paper.

IV. Results and Discussion

A. Results: Comparison of Serial and GPU Implementations

The serial and GPU based parallel version of the algorithms have been implemented and an experiment was conducted for graphs of size 50 to 300 nodes. In the experiment each implementation has performed a complete enumeration. The input graphs are constructed using $rand()$ function. The execution time is measured using glibc's $gettimeofday()$ function, which provides a precision of microseconds.

The hardware configuration used for experimentation is shown in Table 1. An average speed up of 5 to 10 is achieved. Experiments are conducted with 6 randomly generated graphs (1-6 items in Table 2) and 7 DIMACS graphs (7 - 13 in Table 2. They are standard benchmark graphs used in Maximum Clique problem). The run time and size of the maximal cliques found are summarized in Table 2. The speed up of small size graph is less due to the CPU-GPU transfer overhead appearing in the execution flow.

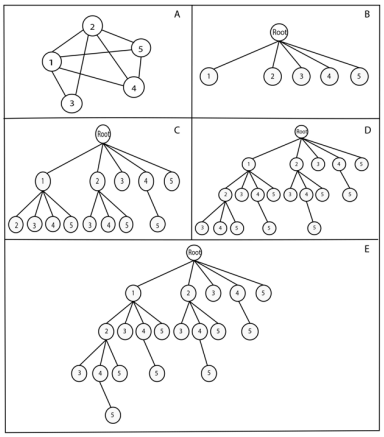

Figure 2: The parallel evaluation of BK search tree. A) The input Graph B) *CurrentActiveSet* is initiated C) First iteration: Kernel evaluated the first level nodes D) Second iteration of kernel evaluation E) Last iteration: Final BK search tree is built

B. Applications using the proposed GPU parallel MCE algorithm

The maximum common subgraph problem solution has many pragmatic applications in different areas like bioinformatics, cheminformatics, pattern recognition [2], [3], [21]. As applications of the newly developed GPU based maximum common subgraph algorithm, two useful tools for drug discovery were developed as follows:

1. **LIGANDMATCHER**: A ligand matching tool to compare two ligands for their substructure similarity.

2. **GRAPHSCREEN**: A virtual screening tool for selecting the lead molecules from a set of unknown compounds against a given active compound.

LIGANDMATCHER: According to similar property principle [3], compounds which have a similar structure also will have same biological activity and chemical similarity. So a compound which has a similar structure to that of a resistant drug, can also act as a drug to the same disease. So finding maximum common substructure will help to find the replacement for the resistant drug. Here this tool will

Algorithm 2 Proposed Parallel Maximal Clique Enumeration Algorithm

Input

G - Edge-Product Graph of two Graphs G1 and G2

Output

M - Maximal Clique in the Graph G

```
1: Read the graph G
2: v ← Number of vertices of Graph G
3: A ← Initiate the CurrentActiveSet
4: Copy G into the GPU
5: repeat
6:     p ← no of nodes in A
7:     for node i from 1 to p in parallel do
8:         for all neighbour node m of i do
9:             if node m is not a parent of node i in A then
10:                atomically add m to A
11:            end if
12:        end for
13:    end for
14: until No changes in A
15:
16: M ← Backtrack A to get the Maximal Clique
```

Particulars	**GPU1**	**GPU2**
GPU	NVIDIA GeForce GTX 780	TESLA K20
CUDA Cores	2304	2496
GPU Clock Speed	941 MHz	706 MHz
Graphic Memory	3072 MB	4800 MB
Memory Bandwidth	288.4 GB/Sec	208 GB/Sec
Peak Performance	4 TFlops	3.52 TFlops
Compute Capability	3.5	3.5
CPU	Intel Core i7	Xeon 2650

Table I: Hardware Configuration used

Sl	**Graph**	**Nodes**	**Max Clique**	**CPU (s)**	**GPU-GTX-780 (s)**	**GPU-TESLA K20(s)**
1	Graph1	50	5	0.012547	0.067277	0.033104
2	Graph2	100	9	1.563211	0.851731	0.297795
3	Graph3	150	10	6.43192	3.45219	1.671103
4	Graph4	200	11	186.516308	23.381524	4.956453
5	Graph5	250	10	2795.03886	285.12978	45.19026
6	Graph6	300	11	4792.518	514.815894	98.813951
7	johnson8-2-4	28	4	0.000321	0.009524	0.006102
8	johnson8-4-4	70	14	4.963601	2.612952	1.319193
9	hamming6-4	64	4	0.002371	0.044246	0.021785
10	cfat-200-1	200	12	0.179545	0.097485	0.058856
11	cfat-500-1	500	14	3.32982	0.414055	0.269897
12	keller4	171	11	857.2841	220.9887	71.033145
13	hamming8-4	256	16	21390.449	3752.1410	1339.1932

Table II: Runtimes and size of maximal cliques found by the serial and parallel algorithms.

help to find the lead molecules from which we can identify the replacement for the resistant drug. The ligand data files are obtained from RCSB protein data bank [22] in the macro molecular crystallographic information file format (mmcif). This file is parsed for obtaining atoms and bonds in the ligand molecule. This information is used to create labeled graph representation for matching. Atoms in ligand correspond to vertices in the graph and the bonds between atoms are represented as edges of the graph. The graphs were compared using the proposed parallel procedure for their common substructure similarity. The alignment of ligands 4-chlorobenzaldehyde and 4-nitrobenzaldehyde by this tool is shown in Figure 3. See the CUDA C implementation at http://ccc.nitc.ac.in/project/GPUMCS/LIGAND_MATCHER/.

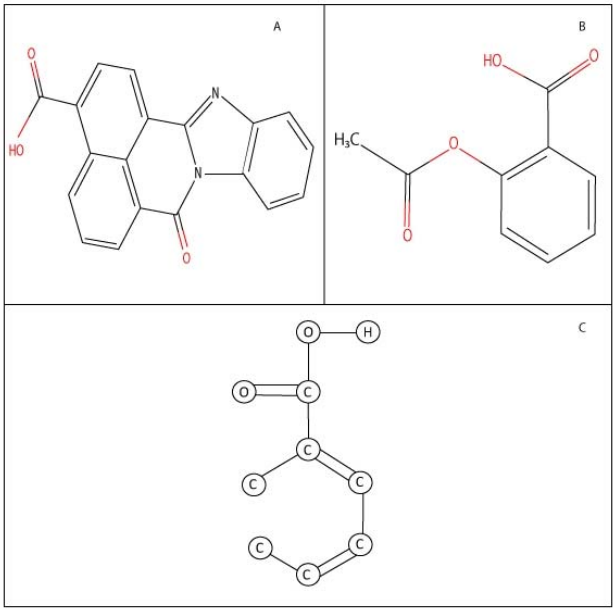

Figure 3: A) Ligand1 : 2-acetyloxybenzoic acid(AIN), B) Ligand2 7-oxo-7H-benzimidazo[2,1-a]benz[de]isoquinoline-3-carboxylic acid(609). C) Common substructure of AIN and 609

GRAPHSCREEN: The drug discovery process is a time consuming and expensive process which in general consumes 500 million dollars to billions of dollars [23] as development cost. The time for a potential drug candidate to be discovered and reach the pharmaceuticals stores can be more than 10 years. Here Virtual screening (VS), an in silico method can help us to reduce the time for developing a drug by identifying bio active compounds for the disease through computational methods. Ligand based virtual screening will be the best approach when the information about biological target is not available [21], [24]. A simple principle which can be used for this is that, compounds with similar structures will show similar biochemical properties. Many structural similarity measures have already been developed to accurately quantify the similarity between compounds. The size of the maximum common subgraph (MCS) between two compound graphs can also be used as a good metric for compound (structural) similarity [3]. So MCS method can also be employed for ligand based Virtual Screening. We have developed a tool named GRAPHSCREEN for virtually screening the chemical compounds using the newly proposed parallel MCS algorithm. The working of the tool and the parallel algorithm used are described in Algorithm 3. The ligand files (both active and unknown) obtained in the Structured Data File (SDF) format from PubChem are parsed to build the graph. The edge table describing number of elements, nature of bonds present between two atoms is built from the SDF file of the compound. To apply the MCE algorithm to find the common substructure, it is required to map parts of both the graphs onto one another. The Edge product graph (EPG) of the given graphs is created (by a procedure named *BuildEdgeProductGraph*) and used in GPU parallel BK algorithm for finding the maximal clique present in the graph. From the obtained clique, one can easily re-map the substructure which is common in the compounds.

A pool of unknown compounds downloaded from PubChem [25] for virtual screening is ranked according to the extent of similarity with the given active compound by cosine similarity rule. The source code of the PyCUDA implementation can be found at http://ccc.nitc.ac.in/project/GPUMCS/GRAPHSCREEN/.

Table 3 shows the time difference when the virtual screening tool was executed in serial and parallel. First two columns represent query and unkonwn (target) compounds. They are shown as pairs of atoms and bonds. It is clear from the table that the serial version cannot complete the execution due to serial exception error when the graph size becomes larger. This time difference shows the viability of this parallel tool for the ligand comparison for drug discovery applications.

Table 4 shows the Top 10 screened compounds when benzene was compared with a pool of 1,00,000 unknown compounds for their similarity using GRAPHSCREEN.

Table 5 shows the change in the execution speed with change in number of unknown compounds in virtual screening by the GRAPHSCREEN tool.

V. Conclusion

Maximum common subgraph problem has pragmatic applications in a myriad of areas like bioinformatics, chemistry, pattern recognition etc. As the size of the graphs increases the solution search space also escalates combinatorially. Therefore parallel solutions are endorsed over serial solutions for MCS problem. Considering the enormity of data, a parallelized version of MCS has a reduced running time and increased throughput. The cost of installation, power consumption and maintenance of a GPU based system is less than other multicore systems. Thus, the GPU based MCS is a viable alternative to quickly compare graphs at a comparatively lower cost. The parallel algorithm presented in this paper is able to handle the data-intensity

Algorithm 3 Parallel Algorithm for Virtual Screening using MCE using GPU Computing

Input

ACTIVE - Structured Data File(SDF) of Active compound

UNKNOWN - Pool of SDF files of unknown compound whose ACTVITY may be predicted

Output

Ranked pool of UNKNOWN compounds by sub structure similarity

```
GT ← Build Edge Table from ACTIVE input file
GQ ← Build Edge Tables from UNKNOWN input file
for each Graph G in GQ in CPU Parallel do
    EPG ← BuildEdgeProductGraph(G, GT)
    Add EPG to EPGBUFFER
end for
for each Graph G in EPGBUFFER in GPU Parallel do
        v ← number of vertices in G
    A ← Initiate the CurrentActiveSet
    Allocate GPU Memory
    Copy CurrentActiveSet into the GPU
    repeat
        p ← number of nodes in CurrentActiveSet
        for node i from 1 to p in parallel do
            for All neighbour node m of i do
                if node m is not a parent of node i in A then
                    Atomically add m to A
                end if
            end for
        end for
        Synchronise GPU threads
    until No Changes in A

        M ← Backtrack Active Set A to get the Maximal Clique
    Calculate the Common Subgraph and Count the nodes and Edges present in it
    Compute the similarity of common subgraph with the query graph in terms of number of edges
    Rank the compounds according to the obtained Consine Similarity value
end for
```

Query(atoms, bonds)	Unknown(atoms, bonds)	EP Graph Size	Maximal Clique Size	Serial Time(sec)	GPU time(sec)
(18,19)	(23,24)	89	15	356.70	5.33
(18,19)	(22,23)	103	15	924.26	4.56
(18,19)	(20,21)	85	13	210.89	3.09
(18,19)	(35,37)	136	12	Not Running	3.69
(18,19)	(38,40)	148	10	Not Running	4.56
(18,19)	(17,17)	53	10	93.40	3.08
(18,19)	(31,32)	109	7	1028.35	3.27
(20,21)	(31,32)	133	15	Not Running	5.68
(20,21)	(22,23)	125	10	Not Running	5.63
(25,26)	(23,24)	155	16	Not Running	17.4
(25,26)	(20,21)	144	14	Not Running	5.49
(31,31)	(17,17)	87	10	346.08	3.30

Table III: Comparison of CPU time and GPU time for various query compounds for screening out the lead molecules

Rank	Query	Unknown	Bonds in Query	Bonds in unknown	Similarity
1	Benzene	compound-94089	13	11	0.585369
2	Benzene	compound-2436	13	12	0.560449
3	Benzene	compound-32616	13	12	0.560449
4	Benzene	compound-39287	13	12	0.560449
5	Benzene	compound-55925	13	12	0.560449
6	Benzene	compound-843	13	13	0.538462
7	Benzene	compound-11906	13	13	0.538462
8	Benzene	compound-16672	13	13	0.538462
9	Benzene	compound-19285	13	13	0.538462
10	Benzene	compound-19487	13	13	0.526235

Table IV: First 10 Ranked elements of GRAPHSCREEN virtual screening

Sl.No	No of compounds	Execution time (sec)
1	1	0.10
2	5	0.53
3	10	1.06
4	100	10.59
5	1000	105.83
6	10000	1057.57
7	50000	5285.78
8	100000	10570.69

Table V: Variation in run time of GRAPHSCREEN with change in input compound set.

of the problem for large graphs. The algorithm uses the BK method of maximal clique enumeration for solving the common substructure problem. By using this newly developed parallel algorithm, two application tools for ligand based drug discovery are developed, which can speed up the drug discovery process.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

ACKNOWLEDGMENT

We would like to thank Central Computer Centre, National Institute of Technology Calicut, India and NVIDIA Corporation LTD for providing the CUDA based GPU infrastructure.

REFERENCES

[1] Ullmann and J. R, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, pp. 31–42, 1976.

[2] Y. Cao, T. Jiang, and T. Girke, "A maximum common substructure-based algorithm for searching and predicting drug-like compounds," *BIOINFORMATICS*, vol. 24 ISMB, p. 366 374, 2008.

[3] P. J. durand, R. Pasari, J. W. Baker, and C. che Tsai, "An efficient algorithm for similarity analysis of molecules." *Journal of Chemical information and Modeling*, vol. 2, 1999.

[4] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs." *Theoretical Computer Science*, vol. 250, no. 1-2, pp. 1–30, 2001.

[5] J. J. McGregor, "Backtrack search algorithms and maximal common subgraph problem." *Software-Practice and Experience*, vol. 12, pp. 23–34, 1982.

[6] H. Bunke, P.Foggia, C. Guidobaldi, C. Sansone, and M. Vento, "A comparison of algorithms for maximal common subgraph on randomly connected graphs." *Structural, Syntactic and Statistical pattern recognnition*, pp. 85–106, 2002.

[7] D. Conte, P. Foggia, and M. Vento, "Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs," *Journal of Graph Algorithms and Applications, http://jgaa.info/*, vol. 11, pp. 99 – 143, 2007.

[8] P. Ostergard, "A fast algorithm for the maximum clique problem," *Discrete Appl. Math. 120*, pp. 197–202, 2002.

[9] M. C. Schmidt, N. F. Samatova, K. thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration." *Journal of Parallel distributed computing*, vol. 69, pp. 417–428, 2009.

[10] B. Subramaniam, W. Saunders, T. ScogJand, W. chun Feng, and V. T. Department of Computer Science, "Trends in energy-efficient computing: A perspective from the green500," *A white paper*.

[11] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973.

[12] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs." *Calcolo*, vol. 9, pp. 341–352, 1972.

[13] E. Tomitaa, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science 363*, p. 28 422, 2006.

[14] D. B. Kirk and W. mei. W. Hwu, *Programming massively parallel processors, A hands on approach.* Morgan Kaufmann, 2010.

[15] J. Jenkins, I. Arkatkar, J. D. Owens, A. Choudhary, and N. F. Samatova, "Lessons learned from exploring the backtracking paradigm on the gpu," *Euro-Par 2011 Parallel Processing 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2*, Aug. 2011.

[16] T. C. Pessoa and M. J. N. Gomes, "Jurema, a new branch and bound anytime algorithm for the asymmetric travelling salesman problem," *In: Annals of the XLII Simpsio Brasileiro de Pesquisa Operacional*, p. 70690, 2010.

[17] T. Carnetro, A. E. Muritiba, M. Negreiros, and G. A. L. De Campos, "A new parallel schema for branch and-bound algorithms using gpgpu," *Computer Architecture and High Performance Computing*, vol. 1550-6533, pp. 41 – 47, 2011.

[18] T. Carnetro, M. Muritiba, A. E. De Campos, and G. A. L. Negreiros, "Solving atsp hard instances by new parallel branch and bound algorithm using gpgp," *Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE)*, 2011.

[19] T. C. R. N. M. N. G. A. L. de Campos, "Depth-first search versus jurema search on gpu branch-and-bound algorithms: a case study," *XXXII Congresso da Sociedade Brasileira de Computao,Brazil*.

[20] K. Kurowski and M. Mackowiak, "Parallel branch and bound method for solving traveling salesman problem using cpu and gpgpu volunteer computing and the xmpp protocol," *The 25th International Conference on Advanced Information Networking and ApplicationsBiopolis, Singapore, March 22-25*, 2011.

[21] H. Eckert and J. Bajorath, "Molecular similarity analysis in virtual

screening: foundations, limitations and novel approaches," *Drug Discovery Today*, vol. 12, no. 5/6, Mar 2007.
[22] "RCSB protein data bank," http://www.rcsb.org/pdb/home/home.do.
[23] C. P. Adams and V. V. Brantner, "Estimating the cost of new drug development: Is it really 802 million dollar?" *health affairs*, vol. 25, 2006.
[24] A. C. Schierz, "Virtual screening of bioassay data," *Journal of Cheminformatics*, vol. 21, pp. 1–12, December 2009.
[25] "NCBI PubChem," https://pubchem.ncbi.nlm.nih.gov/.