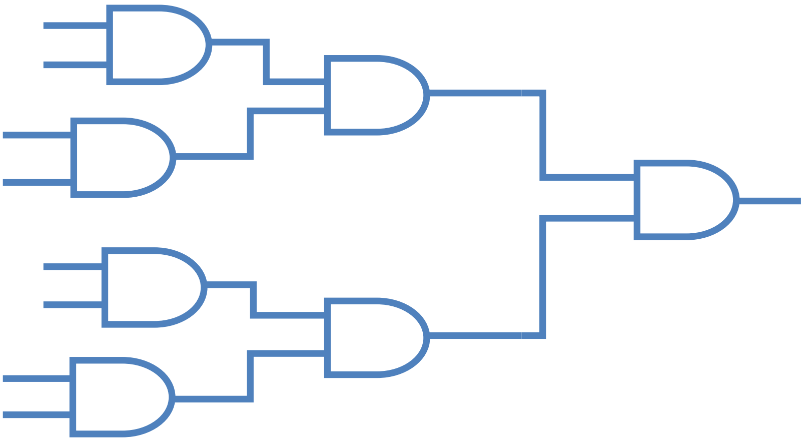


VLSI Circuit Partitioning

ESE 556 VLSI Physical and Logic Design Automation

Abstract

To implement and experiment the Fiduccia-Mattheyses partitioning algorithm implemented for gate-level designs

Project Members

Kunal Wadhwa	111518025
Sukrutha Jade	111635216

Stony Brook University

Fall 2017

CONTENTS

1. Problem Statement:

The main aim of the project is to implement and experiment the Fiduccia-Mattheyses partitioning algorithm implemented for gate-level designs and to this minimize the cut-set size while meeting the given area constraints.

2. Introduction:

2.1. Need for Partitioning

As the data sets become more and more complex and vast, the decomposition of the sets becomes necessary. Here, the need of partitioning the data set arises. Partitioning of data into smaller sets helps define the structure in a more modular way. It does not alter the basic functionality of the structure, but gives a more simplistic approach to tackle and understand the set better.

Partitioning plays a crucial role in reducing the design complexity of VLSI chips. When the chips contain a multitude of transistors (usually in millions), partitioning the transistor set helps aid the designing process. VLSI partitioning takes place at several levels (system level, board level, chip level) but inputs of the problem remain similar (analogous to the level), and the output desired is always some sub-circuits that are same in functionality to the original set.

2.2. Overview of the Problem statement

When a circuit is partitioned in multiple sub-circuits, often there exists a large number of interconnections between the various sub-circuits. These interconnections between the sub-circuits is called the cut-set and the total number of these interconnections is called the cut-set size. The main objective of the project is to first partition the given set into two subsets, and then minimize the cut-set size while also adhering to the given area constraints. Fiduccia-Mattheyses partitioning algorithm is to be used.

Kernighan-Lin Algorithm

Kernighan-Lin algorithm or simply KL algorithm is a heuristic algorithm to partition graphs into two subsets of equal sizes. An initial partition is made, and the cut-set size for that partition is determined. Then one vertex from each partition is picked and interchanged in the two sets to reduce the gain. This process is repeated until the cut-set size cannot be reduced any further. Choosing the vertex pair depends on which pair gives the maximum benefit when their places are interchanged. KL algorithm forms the basis of our understanding of the Fiduccia-Mattheyses partitioning algorithm.

Fiduccia-Mattheyses Algorithm

Fiduccia-Mattheyses algorithm or FM algorithm is essentially an upgrade to the KL algorithm. Differences between the two are:

1. Instead of moving a pair of nodes across each partition, only one node is moved at a time from one partition to the other. This enables to handle partitions that are not essentially equal in size (when number of nodes are considered) and could also vertex weights (to meet the area constraints)

2. FM algorithm, unlike KL algorithm, is not limited to graphs. Hyper-graphs can also be partitioned.

3. Since only one vertex is chosen to move instead of choosing the best pair of vertex to move, FM algorithm is much faster than KL algorithm.

3. Related Work:

Since partitioning of nets is critical in the VLSI industry, a lot of research work is available with respect to improvement of the partitioning [1] [2] using the ratio factor as a criterion or extending it to multiway partitioning [3] from bi-partitioning algorithm which requires the use of efficient data structures for moving the nodes like a LIFO instead of a random queue or FIFO queues. Caldwell,Khang and Markov have proposed Multilevel FM Hypergraph partitioning which exploits the fixed nodes in the partitioning[4].As directed graphs are used in modeling of data flow in streaming applications the partitioning algorithms can be utilized for parallelizing computation for multiprocessor architectures as proposed recently in the Evolutionary Acyclic Graph Partitioning[5].

4. Implementation:

The implementation of the Fiduccia Mattheyses algorithm for minimizing the cut-set size while meeting the area constraints can be split into the following:

4.1. File Parsing:

- Partitioning needs to be done on VLSI networks containing millions of cells and the benchmark for the circuit is provided by [6] THE ISPD98 CIRCUIT BENCHMARK SUITE, where the IBM files .net, .are, .netD files which serve as the input for the partitioning can be obtained.
- Various data like the area of each cell, the name of the cell, the connections to other nodes, source cell information, direction, etc are obtained after parsing these files and the data is populates in two structures which is the bucket structure and the cell structure (cell is called vertex in hypergraps).

4.2. Generation of initial partition:

- The initial bi partition for the given netlist can be obtained either randomly or in a specific way to aid the cut set size reduction.
- If the total area is calculated and the ratio factor is fixed based on this the part A sectioning can be decided. But it is a trade off in time as randomized function can be used to divide the bucket structure into 2 groups.

$$\boldsymbol{Ratio\ Factor = Area\ A/(Area\ A + Area\ B)}$$

  - Area A→ Area of the first partition (A)
  - Area B→Area of the second partition (B)
  - Total Area→ Area A + Area B

4.3. Bucket Structure:

- The bucket structure is implemented using a two-dimensional linked list as the number of nodes is dynamic the data of the cell and the network cannot be stored in a static array.

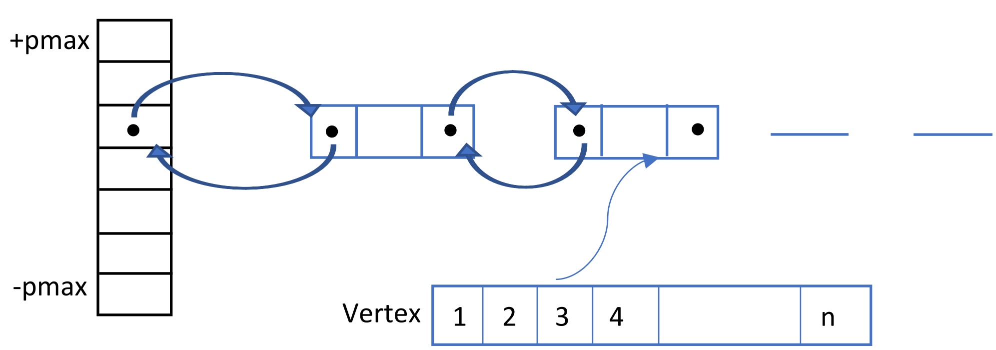


Fig 1.

4.4 Fiduccia Mattheyses:

Implement the FM algorithm:

- Calculate the gain of all the nodes after checking the connections and critical nets using the display cell structure.

$$\Delta g(c) = FS(c) - TE(c)$$

Where the "moving force" FS(c) is the number of nets connected to c but not connected to any other cells within c's partition, i.e., cut nets that connect only to c, and the "retention force" TE(c) is the number of uncut nets connected to c.

The higher the gain Δg(c), the higher is the priority to move the cell c to the other partition.

- Compute the balance criterion for area constraint:

$$[r.area(V) - area_{max}(V)] \leq area(A) \leq [r.area(V) + area_{max}(V)]$$

- Choose the cell with maximum gain and move it to the other partition and fix it.

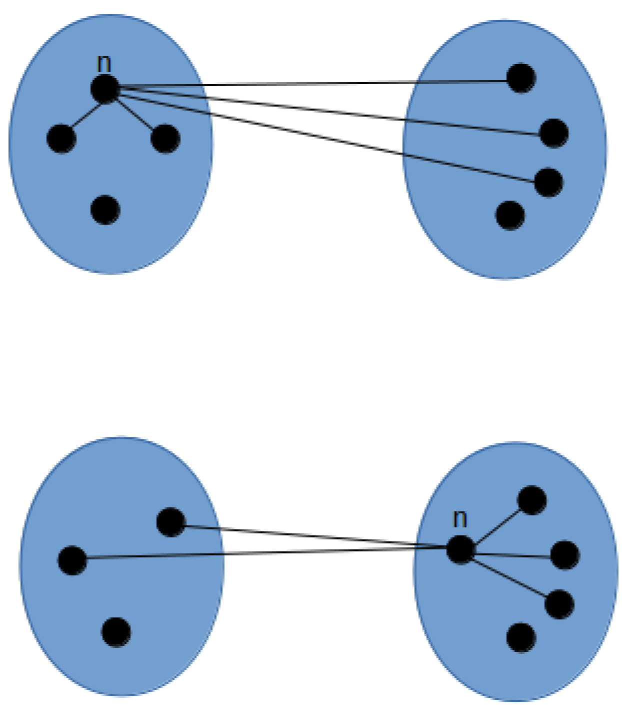
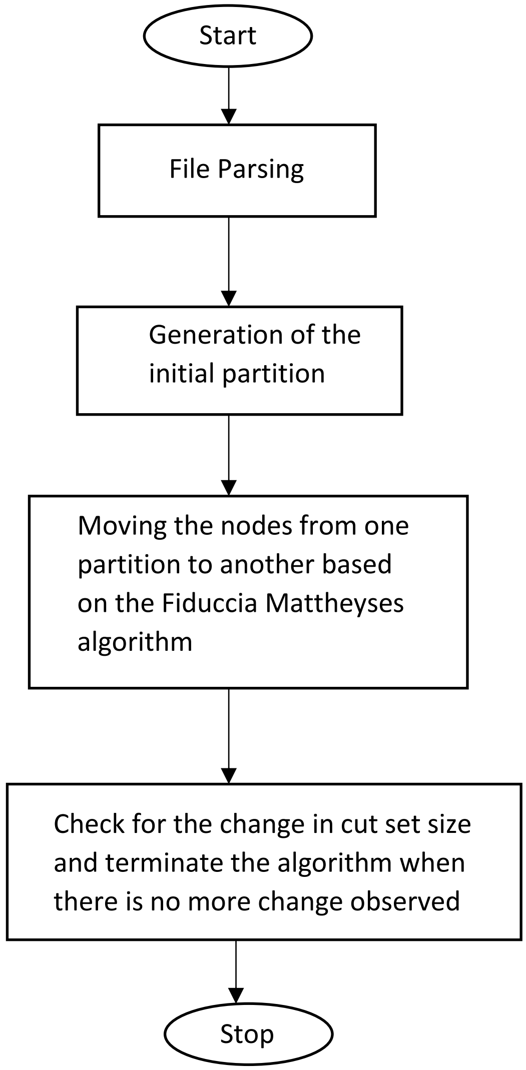


Fig 2.

- Update all the cell that are connected to critical nets via the base cell
- Repeat the previous two steps until all cells are locked or until athe minimum cut size with maximum gain is obtained.
- Figure out the best sequence to move all the cells.

4.5. Flowchart:



5. Experimental results:

The Fiduccia Mattheyses algorithm implemented for circuit partitioning was tested for the IBM networks which are provided in the benchmark webpage http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html

The initial cut set size and the final cut set size after the finalized partition are calculated and the percentage reduction is also calculated.

The figures attached show the runtime, reduction in cut set size for all the 13 IBM files and the table shows the percentage reduction.

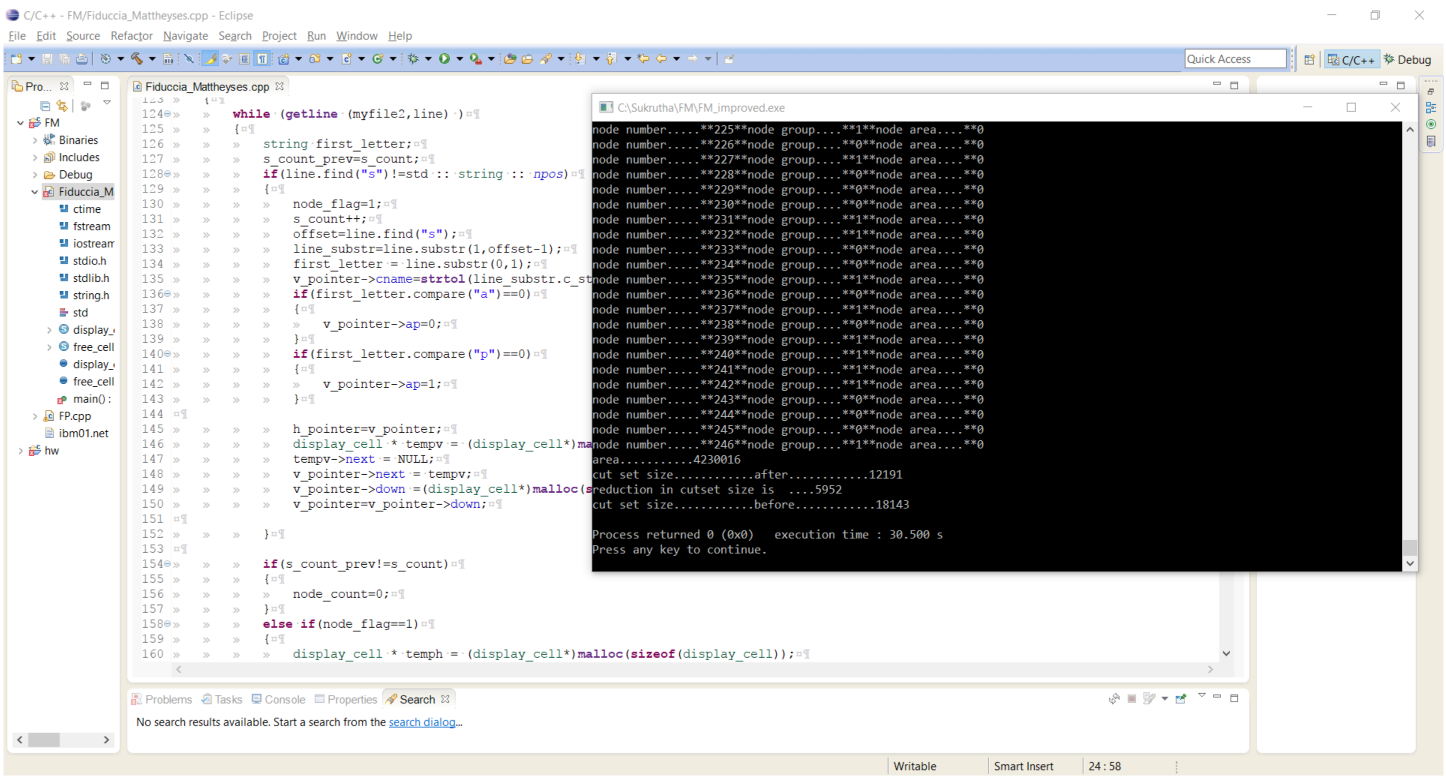


Fig 3. Output terminal results for ibm01

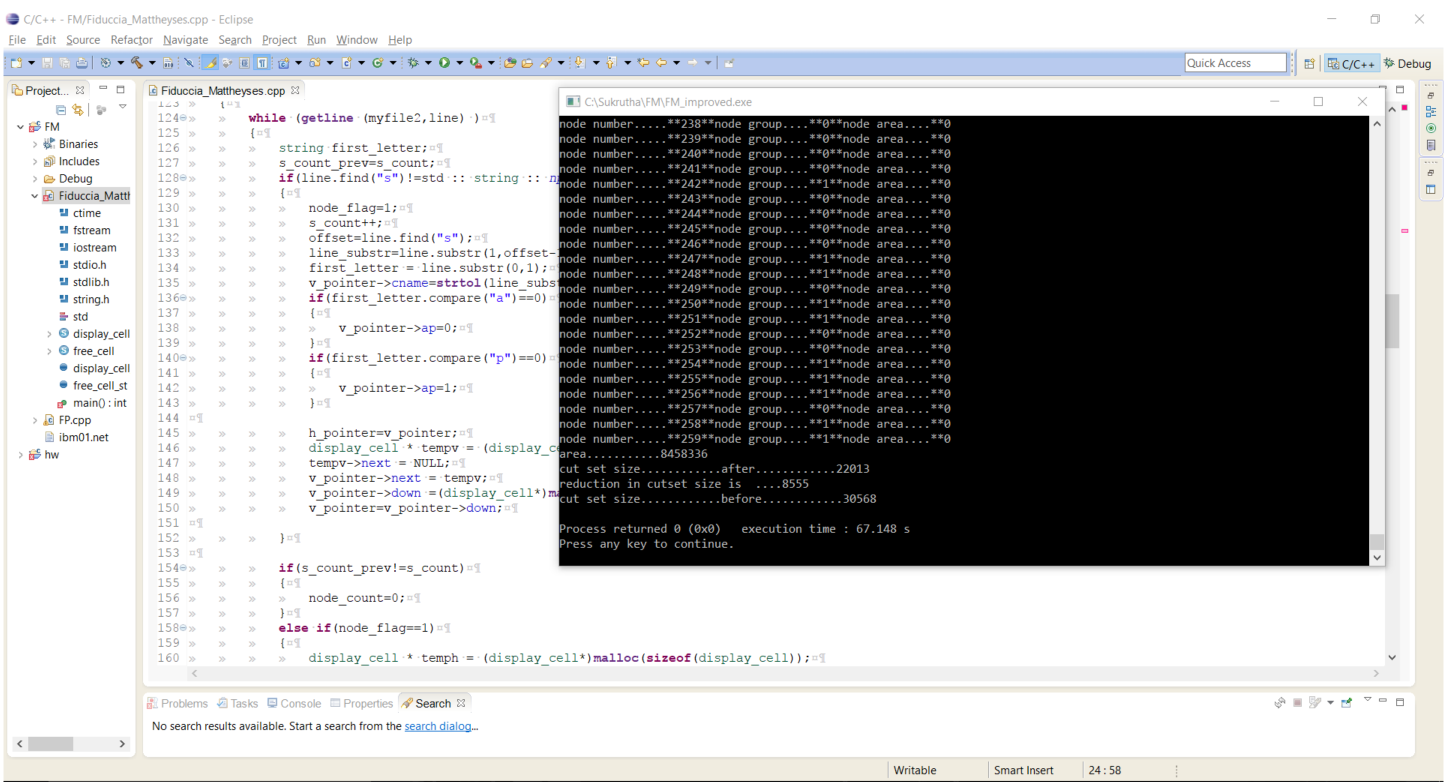


Fig 4. Output terminal results for ibm02

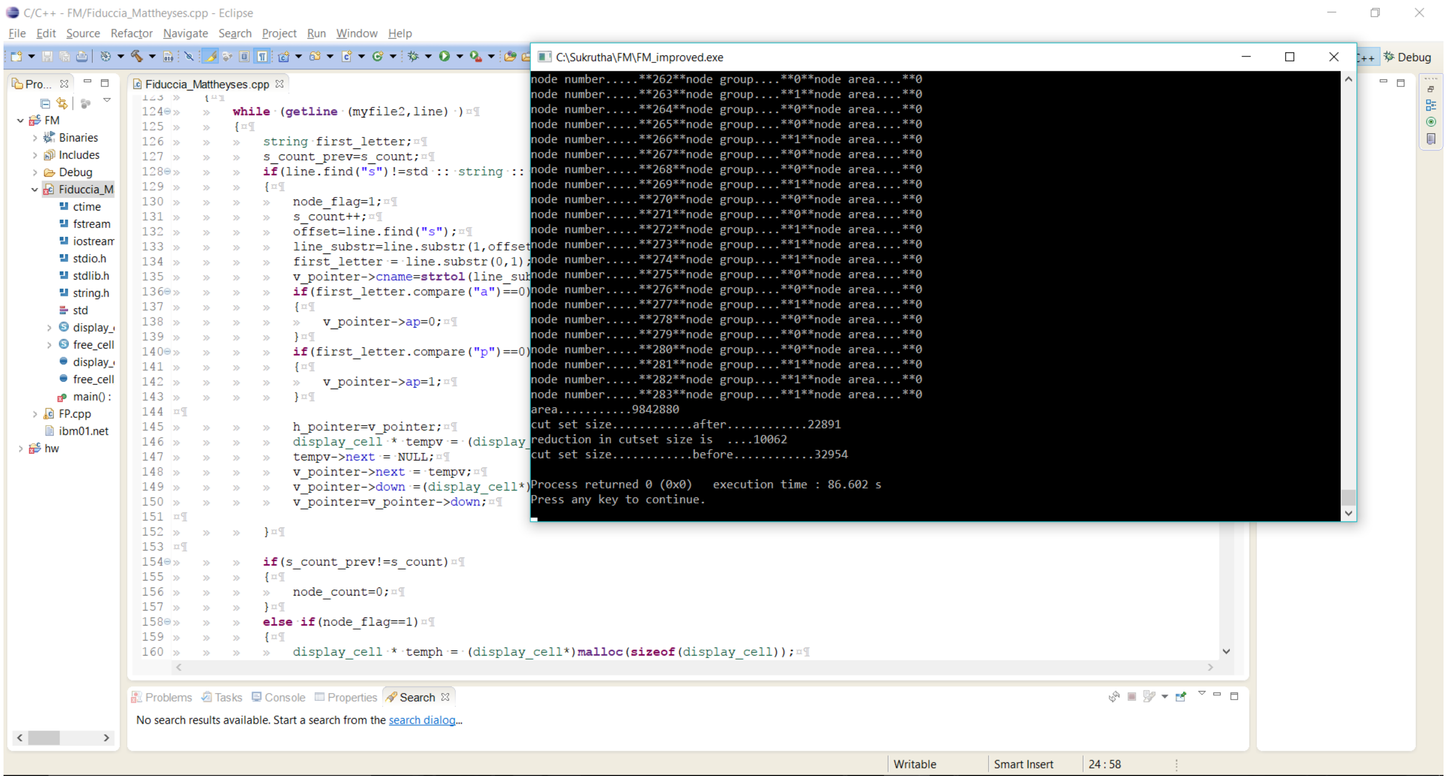


Fig 5. Output terminal results for ibm03

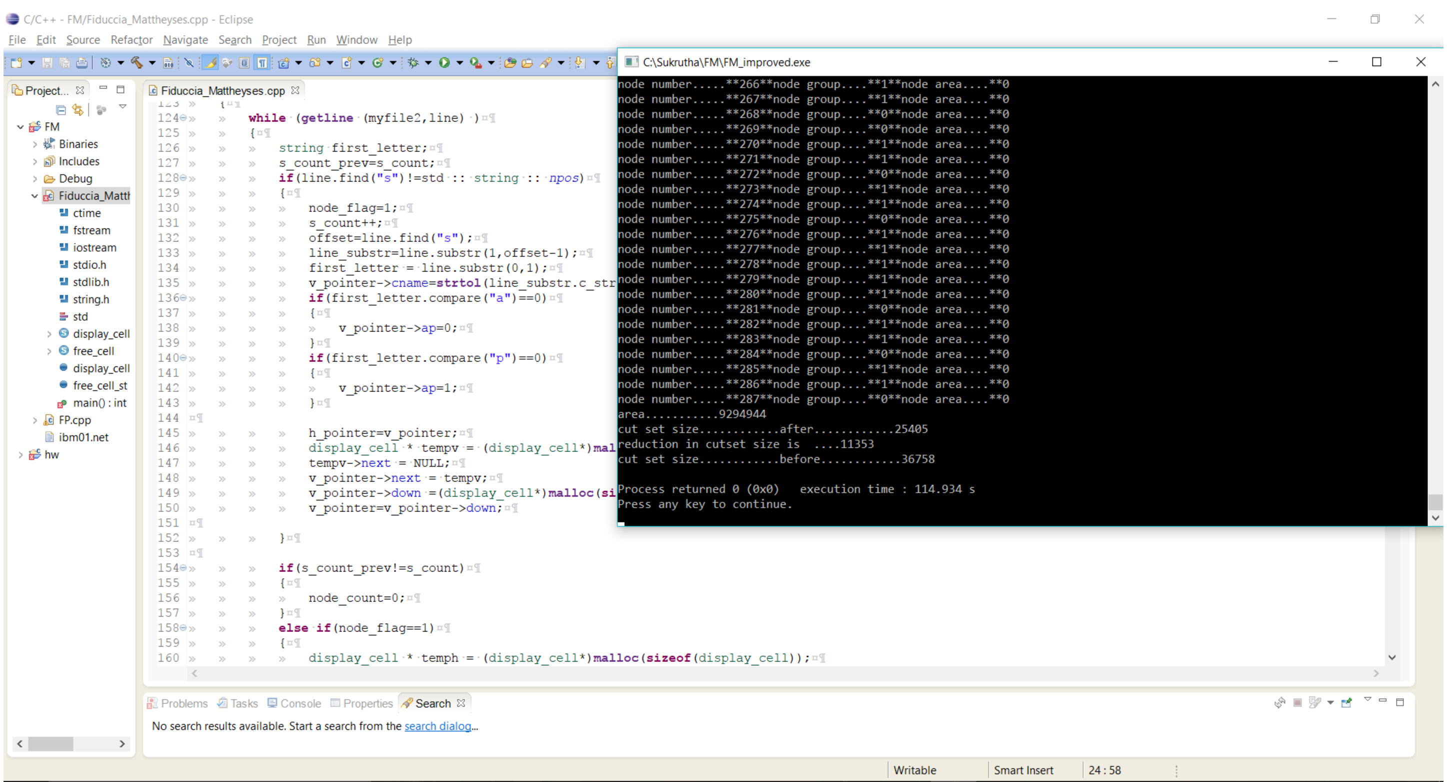


Fig 6. Output terminal results for ibm04

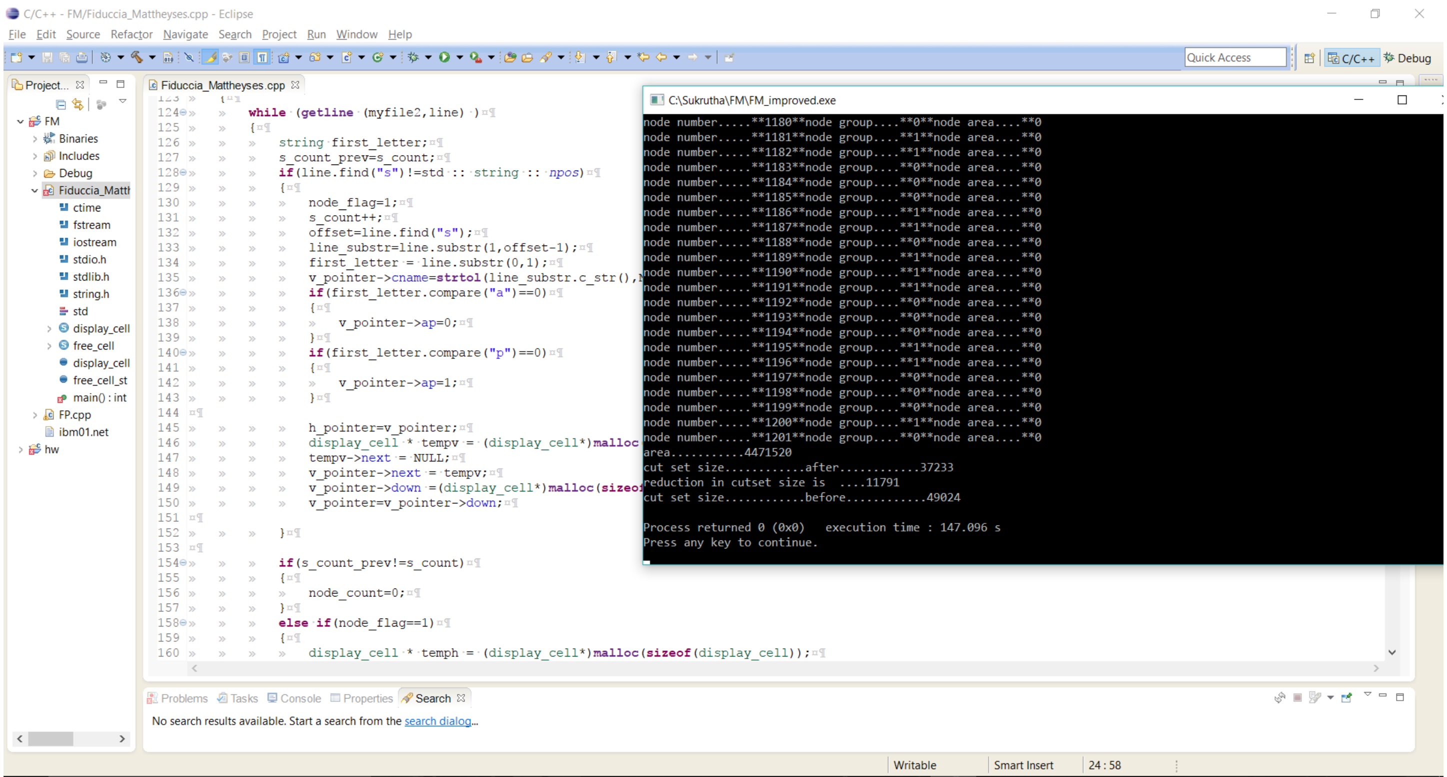


Fig 7. Output terminal results for ibm05

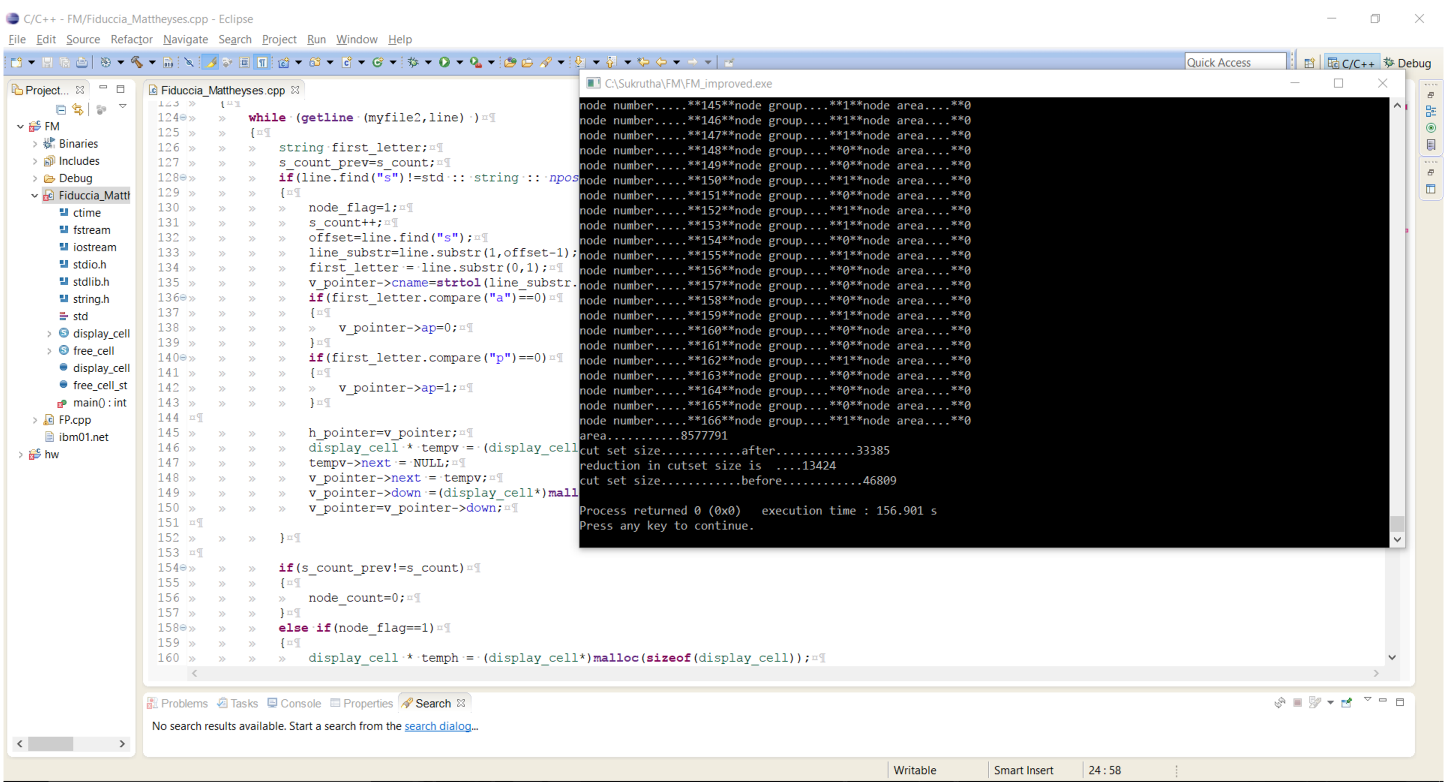


Fig 8. Output terminal results for ibm06

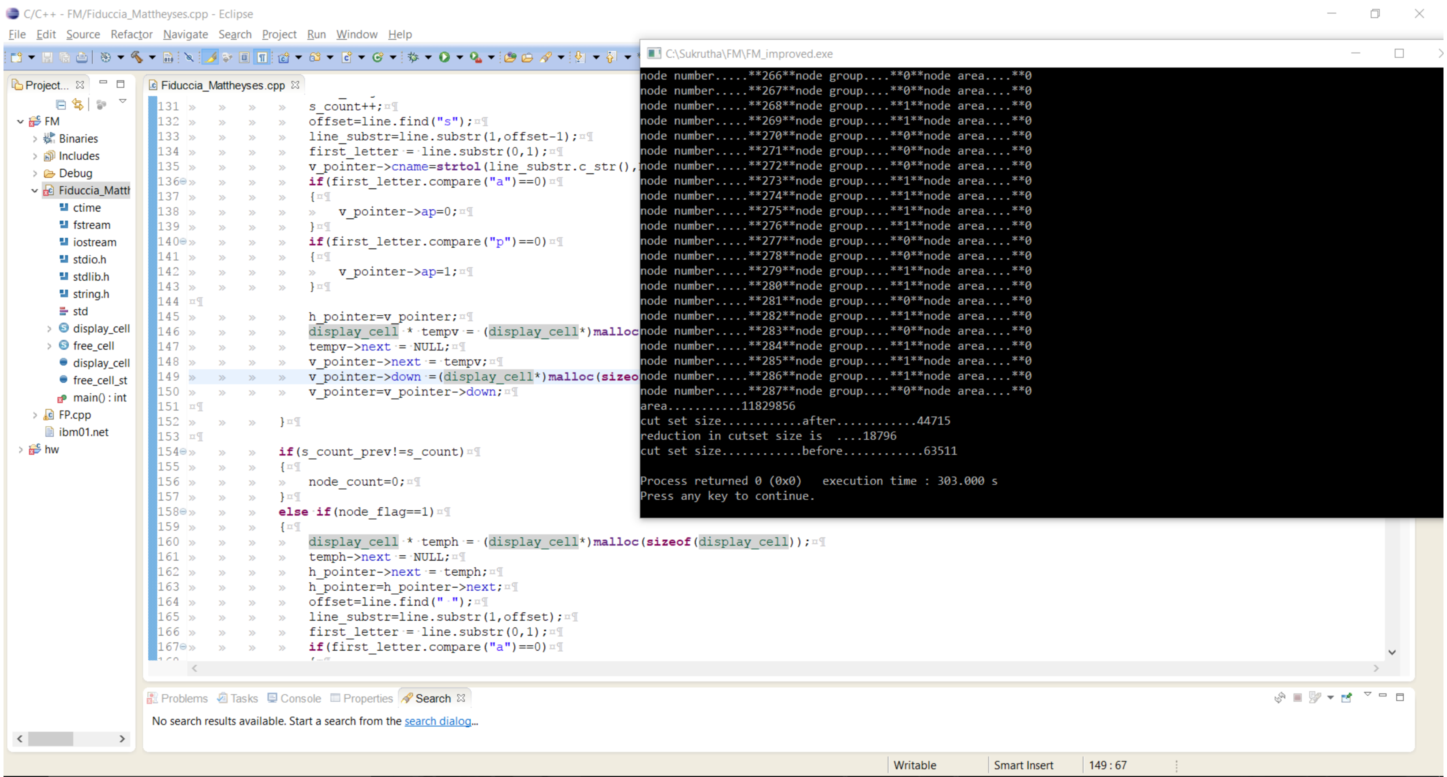


Fig 9. Output terminal results for ibm07

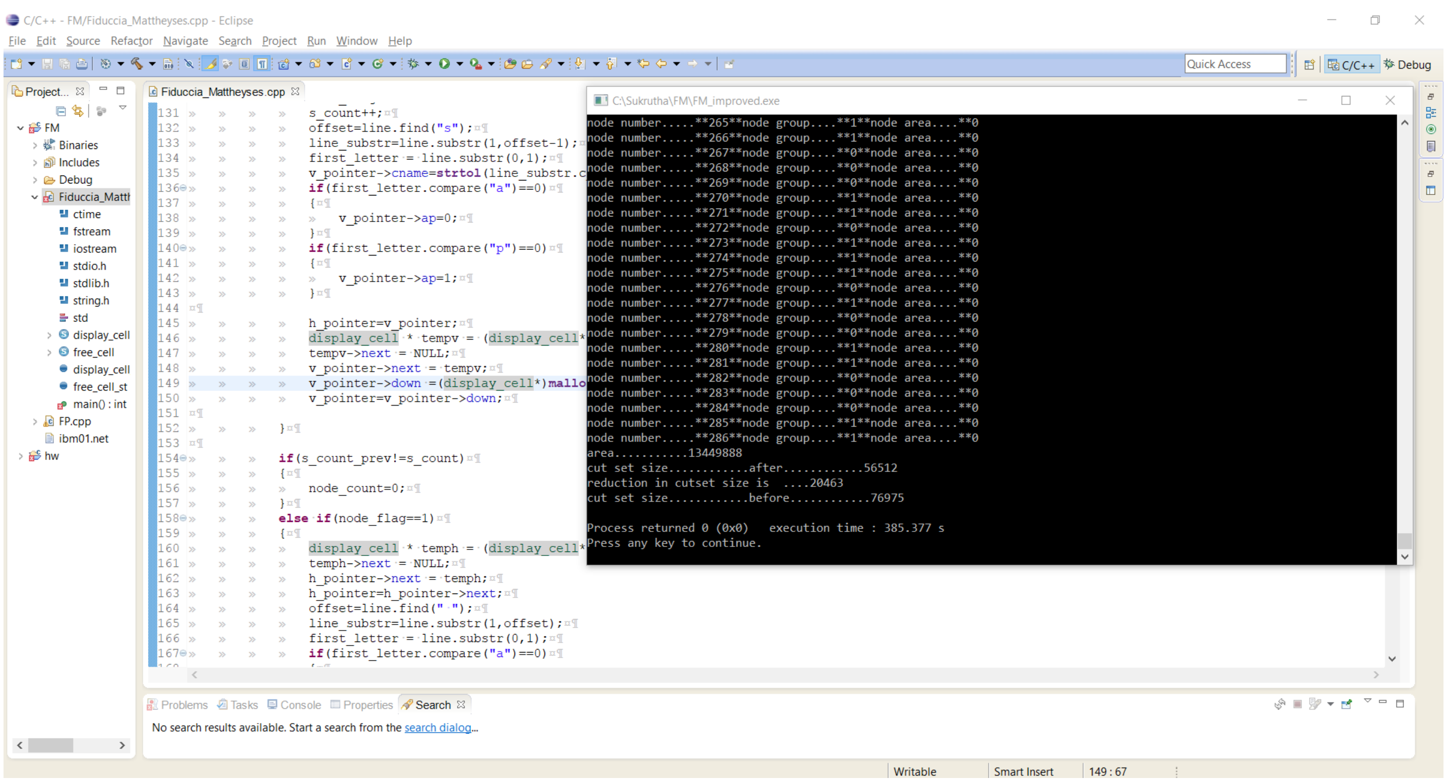


Fig 10. Output terminal results for ibm08

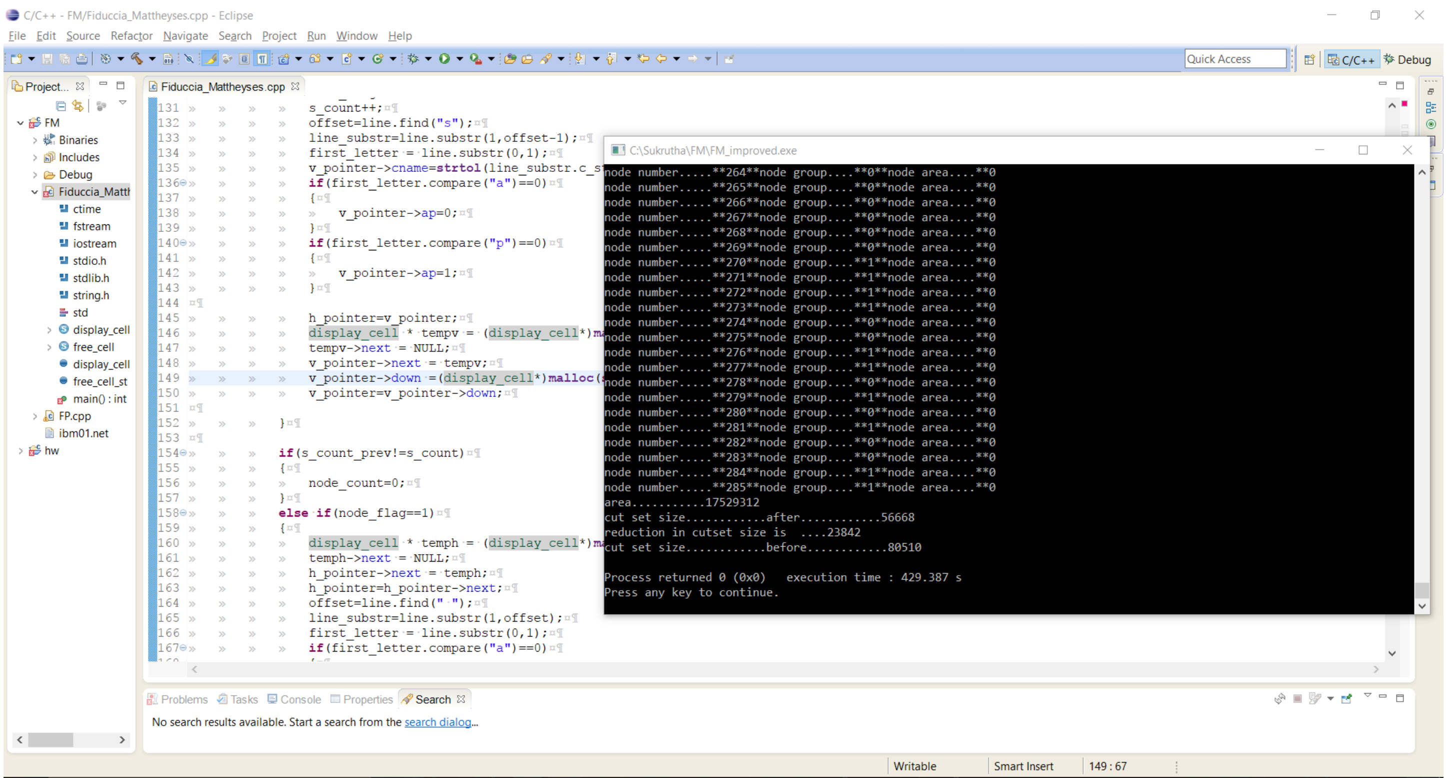


Fig 11. Output terminal results for ibm09

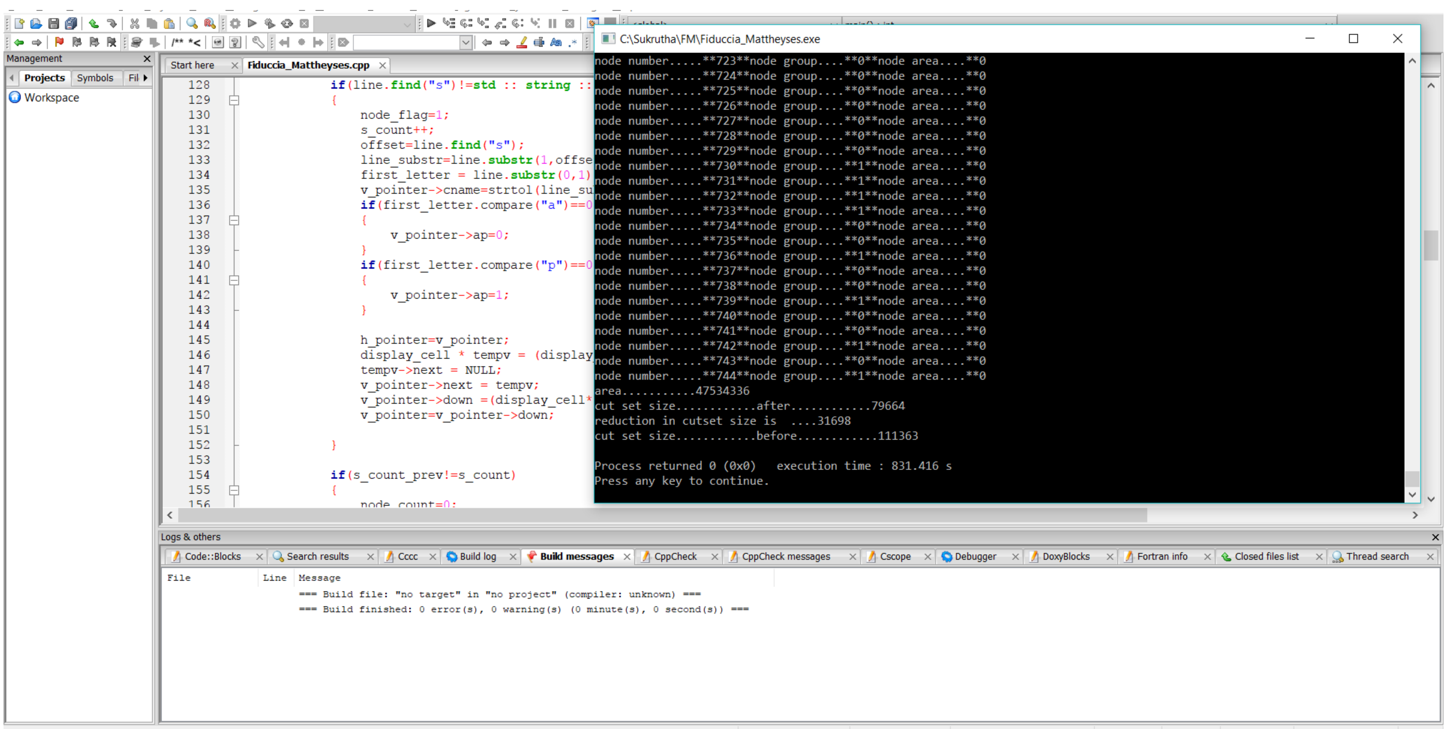


Fig 12. Output terminal results for ibm10

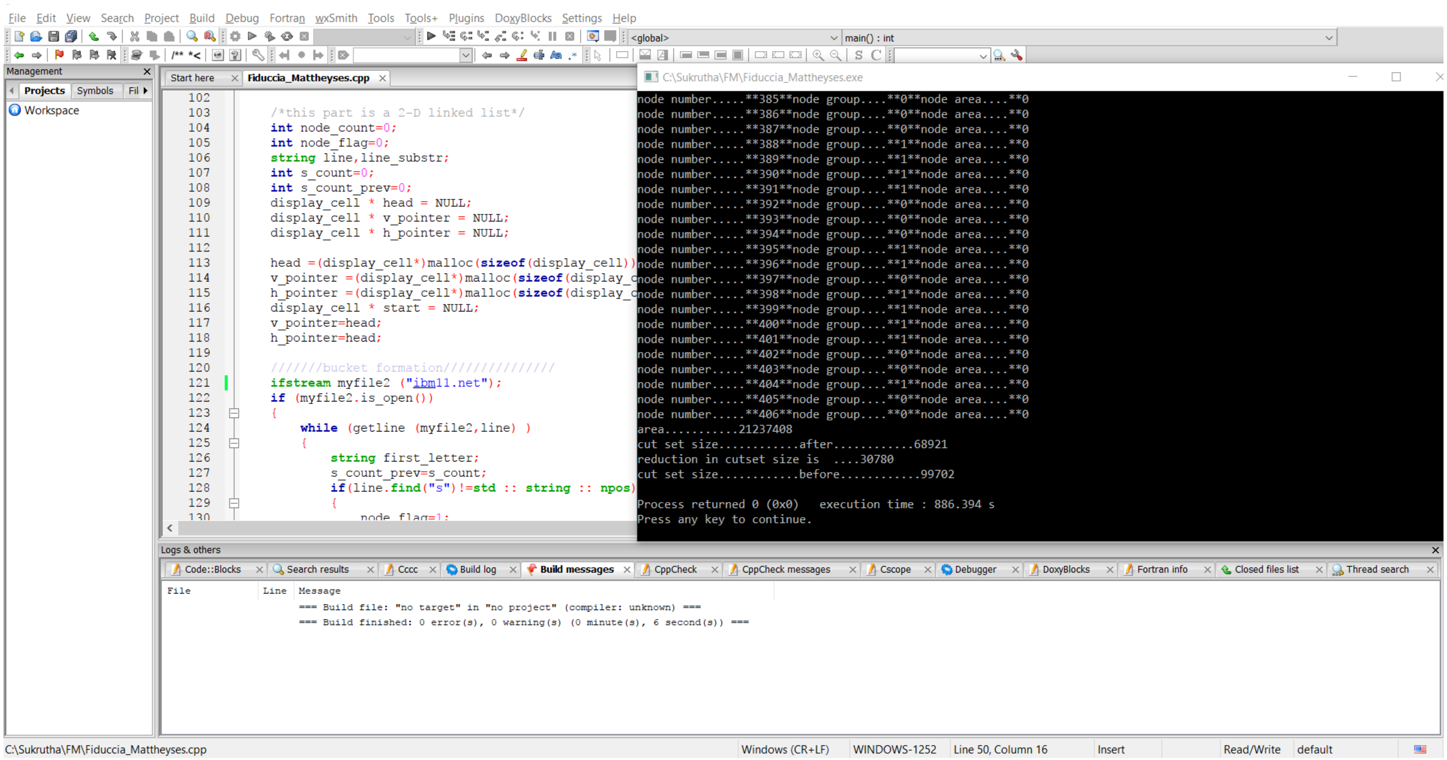


Fig 13. Output terminal results for ibm11

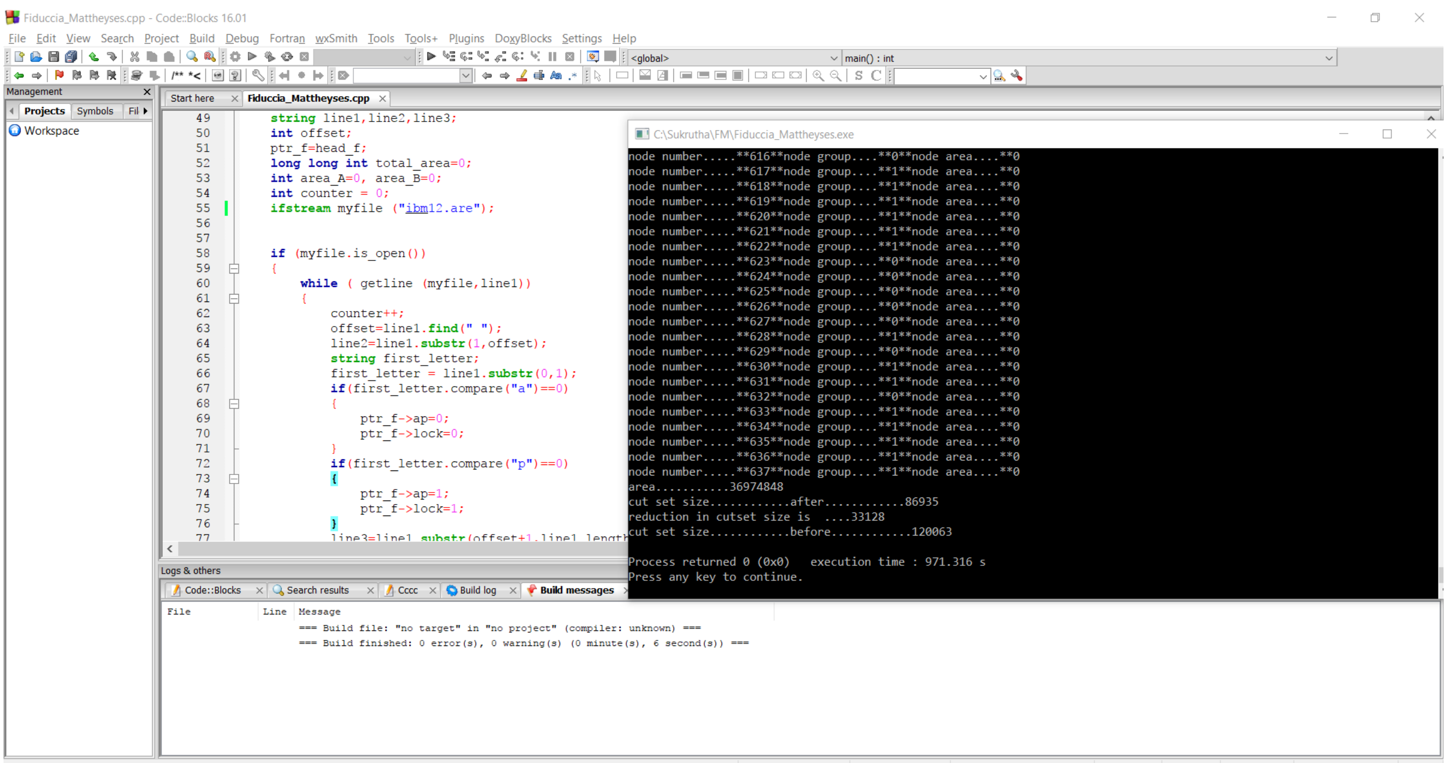


Fig 14. Output terminal results for ibm12

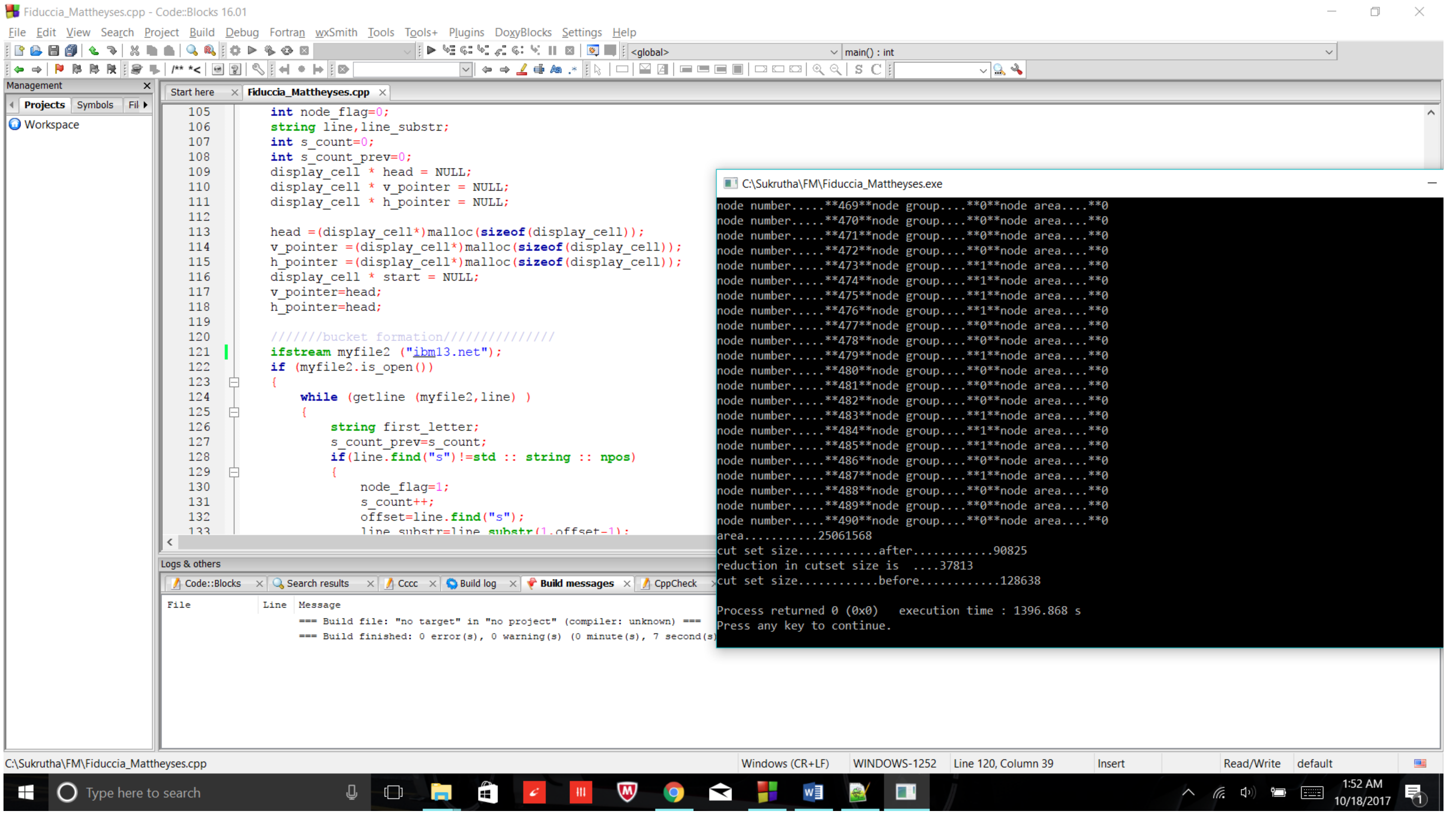


Fig 15. Output terminal results for ibm13

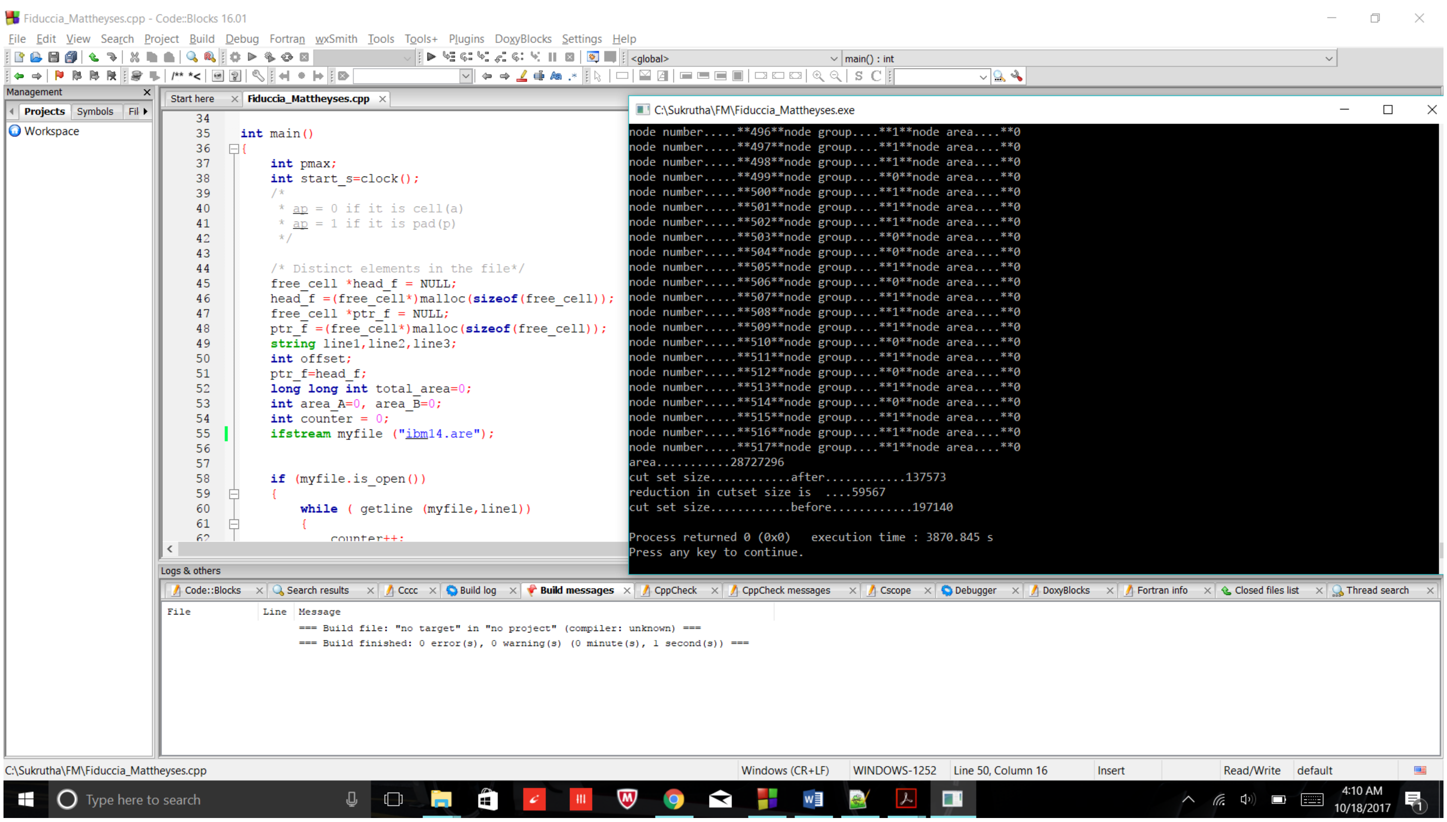


Fig 16. Output terminal results for ibm14

Table for percentage reduction in cutsize:

IBM File Number	Percentage Reduction in cut set size	Running time
ibm01	32.8%	30.5s
ibm02	28%	67.148s
ibm03	30.53%	86.602s
ibm04	30.88%	114.934s
ibm05	24.5%	147.096s
ibm06	28.7%	156.901s
ibm07	29.6%	303s
ibm08	26.6%	385.377s
ibm09	29.61%	429.387s
ibm10	28.5%	831.416s
ibm11	30.9%	886.394s
ibm12	27.6%	971.316s
ibm13	29.4%	1396.868s
ibm14	30.2%	3870.845s

6. Conclusion:

As the runtime is comparably big for the fast-paced embedded era there can be a few improvements added which will help run the partitioning algorithm faster. The possible improvements are:

- Optimization of the code to reuse the existing code of the bucket structure.
- Run coverage tests at unit level to eliminate redundant modified conditions.
- Extend the two-way partitioning to multiway partitioning and check if the algorithm is consistent
- Feasibility of implementing a different data structure for the bucket instead of the linked list.

7. Bibliography:

[1] Michael T. GoodrichCatherine C. McGeoch, Algorithm Engineering and Experimentation, International Workshop ALENEX'99 Baltimore, MD, USA, January 15–16, 1999 Selected Papers

[2] Chung-Kuan Cheng, Member, IEEE, and Yen-Chuen A. Wei, An Improved Two-way Partitioning Algorithm with Stable Performance, IEEE Transactions on Computer-Aided Design, Vol. 10, No. 12, December 1991

[3] L.W. Hagen, D.J.-H. Huang, A.B. Kahng on implementation choices for iterative improvement partitioning algorithms, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Volume: 16, Issue: 10, Oct 1997)

[4] Andrew B. Kahng and Igor L. Marko V, Improv edAlgorithms for Hypergraph Bipartitioning, Andrew E. Caldwell, Proceedings of the 2000 Asia and South Pacific Design Automation Conference

[5] Orlando Moreira, Merten Popp, Christian Schulz, Evolutionary Acyclic Graph Partitioning, arXiv:1709.08563 [cs.DS]

[6] Charles J. Alpert ,THE ISPD98 CIRCUIT BENCHMARK SUITE, IBM Austin Research Laboratory

[7] Naveed A Sherwani Algorithms for VLSI