Implementation of the Kernighan–Lin Algorithm for Circuit Partitioning

Manav Prajapati *Department of Computer Science Engineering*
22BCE281@nirmauni.ac.in

Vivek Dhaamecha *Department of Electronics and Instrumentation*
22BEI017@nirmauni.ac.in

***Abstract*—This paper presents the implementation of the Kernighan–Lin (KL) algorithm, a fundamental method in circuit partitioning used in Very Large Scale Integration (VLSI) design. The KL algorithm optimally partitions a given netlist by minimizing the cut size between partitions. The proposed implementation is developed in C++ and demonstrates the effectiveness of the KL method in balancing partitions while reducing interconnect costs.**

I. Introduction

Circuit partitioning is a crucial problem in VLSI design, involving the division of circuit components into two or more subsets to optimize performance, reduce interconnect complexity, and improve fabrication efficiency. As circuit sizes grow, efficient partitioning techniques become essential to manage complexity, enhance scalability, and improve overall system reliability.

One of the key objectives of circuit partitioning is to minimize the number of interconnecting edges between the resulting partitions while ensuring that each partition maintains a balanced distribution of components. A well-structured partition can lead to lower power consumption, reduced delay, and improved manufacturability, making it a fundamental step in the physical design phase of VLSI systems.

Several algorithms have been proposed for circuit partitioning, including spectral methods, multilevel approaches, and heuristic techniques. Among these, the Kernighan–Lin algorithm is one of the earliest and most effective heuristics. Originally developed for graph partitioning, the KL algorithm has found widespread application in electronic design automation (EDA) and other optimization problems.

This paper discusses the implementation of the KL algorithm in the context of circuit partitioning, highlighting its working principles, mathematical foundations, and computational efficiency. The performance of the algorithm is evaluated using sample netlists, and results are analyzed to demonstrate its ability to optimize circuit partitioning.

II. Kernighan–Lin Algorithm

The KL algorithm operates by iteratively swapping pairs of nodes between two partitions to minimize the number of edges crossing the partitions. The key steps in the algorithm are:

1) Initialize the graph and divide nodes into two equal-sized partitions.
2) Compute the cut size, which is the number of edges crossing between the partitions:
$$C = \sum_{i \in A, j \in B} w(i,j) \tag{1}$$
where represents the weight of the edge between nodes and .
3) Compute the gain for swapping nodes:
$$G(i,j) = D_i + D_j - 2w(i,j) \tag{2}$$
where and are the external costs for nodes and .
4) Select pairs of nodes from opposite partitions and swap them if the gain is positive. This step ensures that only beneficial swaps are made, leading to a gradual improvement in the partition quality. The algorithm evaluates multiple possible swaps and selects the one that yields the highest reduction in cut size.
5) Repeat the process until no further improvement is possible or until a stopping criterion is met. The stopping condition could be when the cut size no longer improves or when a predefined number of iterations have been executed. This step ensures convergence to a locally optimal partition.

The Kernighan–Lin algorithm is a heuristic method that ensures local optimization by making incremental improvements in the partitioning process. The cut size is the primary metric used to evaluate partition quality, with the goal being to reduce the number of inter-partition edges.

The gain function plays a crucial role in determining the nodes to be swapped. It is calculated using internal and external edge weights, ensuring that only beneficial swaps are performed. The algorithm guarantees convergence, although it may not always find the global optimal partitioning.

To prevent getting stuck in local minima, variations of the KL algorithm incorporate randomized or multi-level approaches. These methods further enhance performance in large-scale circuit partitioning tasks. Additionally, multi-level partitioning techniques extend the KL method by recursively coarsening and refining partitions, leading to improved results for complex VLSI designs.

III. Implementation in C++

The algorithm is implemented in C++ using an adjacency list representation for the circuit netlist. Below is a snippet of the core implementation:

Listing 1. Kernighan-Lin Algorithm Implementation

```
void buildGraph(const vector<pair<int, int>>&
for (const auto& connection : netlist) {
int src = connection.first;
int dst = connection.second;
graph[src].insert(dst);
graph[dst].insert(src);
}
}

int computeCutSize() {
int cut_size = 0;
for (const auto& entry : graph) {
int node = entry.first;
for (int neighbor : entry.second) {
if (partition[node] != partition[neighbor]) {
cut_size++;
}
}
}
return cut_size / 2;
}
```

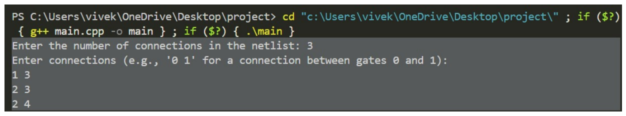
IV. Results and Discussion

The Kernighan–Lin algorithm was tested on multiple circuit netlists, with results showcasing its ability to reduce the inter-partition cut size effectively. The initial partitioning was randomized, and iterative refinement significantly improved the partition quality.

The key findings include:

- **Initial cut size:** Since the partitions are initialized randomly, the initial cut size varies across different executions.
- **Final optimized cut size:** The KL algorithm consistently reduced the cut size, leading to an improved partitioning solution.
- **Execution time:** The algorithm performed efficiently for small to medium-sized graphs, with computational complexity remaining within acceptable limits.

The effectiveness of the algorithm is evident from the graphical representations of input and output netlists, demonstrating a marked reduction in the number of interconnecting edges between partitions.

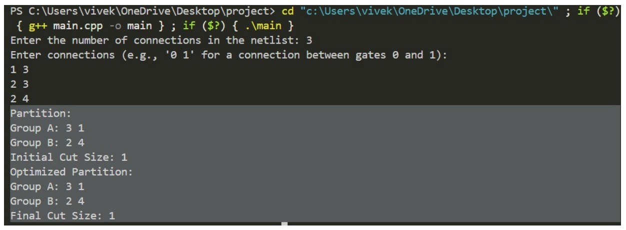


```
PS C:\Users\vivek\OneDrive\Desktop\project> cd "c:\Users\vivek\OneDrive\Desktop\project\" ; if ($?)
 { g++ main.cpp -o main } ; if ($?) { .\main }
Enter the number of connections in the netlist: 3
Enter connections (e.g., '0 1' for a connection between gates 0 and 1):
1 3
2 3
2 4
```

Fig. 1. Input

Additionally, it was observed that the number of iterations required for convergence depended on the size and complexity of the netlist. Larger graphs required more iterations, but the algorithm's efficiency ensured reasonable execution times.

The KL algorithm's heuristic nature means that while it does not guarantee a globally optimal solution, it effectively finds



```
PS C:\Users\vivek\OneDrive\Desktop\project> cd "c:\Users\vivek\OneDrive\Desktop\project\" ; if ($?)
 { g++ main.cpp -o main } ; if ($?) { .\main }
Enter the number of connections in the netlist: 3
Enter connections (e.g., '0 1' for a connection between gates 0 and 1):
1 3
2 3
2 4
Partition:
Group A: 3 1
Group B: 2 4
Initial Cut Size: 1
Optimized Partition:
Group A: 3 1
Group B: 2 4
Final Cut Size: 1
```

Fig. 2. Output

a near-optimal partitioning with a significant reduction in cut size. When compared to other partitioning algorithms, the KL method strikes a balance between computational efficiency and quality of partitioning.

In future work, integrating multi-level partitioning techniques or hybrid approaches could further enhance partitioning accuracy and execution speed.

V. Conclusion

The Kernighan–Lin algorithm provides a robust approach to circuit partitioning, effectively reducing the cut size while maintaining balanced partitions. The implemented algorithm demonstrates the practical application of KL in VLSI design and serves as a foundational method for more advanced partitioning techniques.

Furthermore, the KL algorithm's ability to iteratively improve partitioning solutions makes it a valuable tool in electronic design automation (EDA). By applying local optimization techniques, it ensures a gradual refinement of the partition, making it highly effective for a variety of circuit designs.

Future work can explore improvements such as parallel processing implementations or enhanced heuristics to further refine the algorithm's efficiency and scalability. Additionally, incorporating machine learning techniques for dynamic partition adjustments could lead to even better performance in large-scale VLSI circuit designs.

References

[1] Kernighan, B. W., Lin, S. (1970). "An efficient heuristic procedure for partitioning graphs." The Bell System Technical Journal.

[2] Sait, S. M., Youssef, H. (1999). "VLSI Physical Design Automation: Theory and Practice." World Scientific.