Partitioning Algorithms Implementation and Comparison

Introduction to Computer-Aided Design Term Project Report

Nguyen Tuan Nghia, Nguyen Duy Thanh
Computer Architecture and Parallel Processing Laboratory
Department of Electrical and Computer Engineering, College of Engineering, Seoul National University, Seoul, Korea
{nghiant, thanhnd}@capp.snu.ac.kr

***Abstract*—In this work, we survey three algorithms, including Kernighan-Lin algorithm (KL), Fiduccia-Mattheyses algorithm (FM) and simulated annealing (SA), for the partitioning problem in VLSI. The algorithms are implemented and tested on twelve different benchmark circuits. Their performances are compared in terms of net cut size and execution time in relation to the size of the circuit. The experiments show that all algorithms work equally well in general. Each of them has different dependencies to both satisfy a given constraint and achieve a good result.**

***Index Terms*—CAD, VLSI, Partitioning, Optimization**

I. Introduction

Partitioning is the first and important step in circuit layout process. For a circuit with thousands of cells, partitioning decreases the problem size and makes placement and routing easier to perform, thus improving layout in terms of size and performance. However, the partitioning problem in VLSI is intractable which requires a heuristic algorithms to solve.

Extensive research efforts have been made to solve the problem in a short time while still giving a good result. Most of the invented algorithms aim for bi-partitioning instead of multi-way-partitioning. Also, we usually consider a circuit as a graph or hypergraph, then try to divide it into two subgraphs by a smallest cut possible. The partitioning result may also come with some constraints on the number of cells or size ratio for each partition.

In this work, we survey and implement three state-of-the-art algorithms for the partitioning problem. The rest of this report is organized as follows. Section II briefly introduces the previous researches. We walk through our implementation in Section III, and then show the experimental results in Section IV. Finally, we conclude everything in Section V.

II. Previous Work

There are some algorithms to bi-partition the circuit. They usually carry out by iteratively improving an initial solution. Kernighan-Lin algorithm (KL) [1] starts with a random partitioning result. At each iteration, it makes a move that gives the highest possible gain by swapping a number of cells between two partitions, until there is no more improvement. Fiduccia-Mattheyses algorithm (FM) [2] is a modified version of KL that is able to generate a balanced result for a circuit with non-uniform cell sizes. This algorithm also considers a more accurate cost calculated on nets rather than edges. Instead of swapping a pair of cells at each iteration, only a single cell moves from one partition to the others. That cell is selected to maximize the gain when the move is made. Network flow based algorithm, proposed by Yang and Wong [3], makes use of max-flow algorithm for graph to generate the best cut. Another popular method [4] applies simulated annealing (SA) to the partitioning problem. SA can satisfy size ratio constraint as well as FM. However, it requires a careful tuning of parameters, such as cooling schedule and number of moves, to achieve a good result in a short time.

In order to see the advantages and disadvantages of each algorithm, we implement FM, KL and SA in C programming language, and then compare their performances on MCNC benchmark circuits. The criteria include net cut size and execution time, along with how well a given size ratio constraint is satisfied. In the next section, we walk through the implementation details, before discussing the experimental results in Section IV.

III. Implementation

A. Partitioning Result and Initialization

In our implementation, the partitioning result is represented by an array whose size equals to the number of cells. Indices of the array associate with cell index (cell id). The array is filled with 1 and -1 such that all cells that have the same value belong to the same partition.

Initialization is done by randomly permuting the array of indices. Then, we iterate over the array and assign each cell to a specific partition as described above. For FM and SA, we consider the size ratio at this step by continuously asserting whether adding a cell to a partition would violate the constraint or not. For KL, we simply fill the first half of the partitioning result with 1 and the others with -1. By observation, we find that there is case that the initialization fails the constraint for FM and SA. Thus, we add an option to re-initialize the partitioning result if bad case happens.

B. Fiduccia-Mattheyses Algorithm

FM is implemented as illustrated in Algorithm 1. In the source code, we also add a small improvement on area ratio when there is a tie between the minimum cut size and current

Algorithm 1: FM Algorithm

```
Result: partitioning result and cut size
initialize;
unlock all cells;
min_cut_size = inf;
repeat
    compute gains for unlocked cells;
    compute cut_size;
    if cut_size is smaller than min_cut_size then
        min_cut_size = cut_size;
    end
    sort the gains in descending order;
    find the cell with highest gain that satisfies constraint;
    if none is found then
        break
    else
        move that cell to its opposite partition;
        lock it;
    end
until all cells are locked;
update the final partitioning result;
```

one. That is we choose the move which generates an area ratio with less tolerance than the others. Bucket sort is used in sorting step.

C. Kernighan-Lin Algorithm

Algorithm 2: KL Algorithm

```
Result: partitioning result and cut size
initialize;
repeat
    unlock all elements in A and B;
    compute D values for all a in A and b in B;
    let gv, av, and bv be empty lists;
    for k ← 1 to |V|/2 do
        find a,b : g = D[a] + D[b] - 2*c(a,b) is maximal;
        lock a and b;
        add g to gv, a to av, and b to bv;
        compute G_k = G_(k-1) + g;
        update D values for other unlocked elements;
    end
    find k* that maximizes G_k;
    if G_k* > 0 then
        swap first k* pairs;
    end
until G_k* ≤ 0;
update the final partitioning result;
compute net cut size;
```

The implementation of KL is illustrated in Algorithm 2. Since KL works with graph instead of hypergraph, the cut size that takes part in the main computation is actually computed by edges. When the partitioning result is obtained, it is converted to net cut before the function returns. In case there is an odd number of cells, the last swap is accepted as a single-cell move.

D. Simulated Annealing

Algorithm 3: SA Algorithm

```
Result: partitioning result and cut size
initialize;
repeat
    select a random cell A;
    empty Options;
    compute the area of each partition after A moved;
    if constraint is not violated then
        add single-cell move to Options;
    end
    compute area bounds for the partner cell;
    find any opposite cell whose area is in the bounds;
    add the candidates to Options;
    if no option is available then
        increase move counter;
        continue
    else
        choose randomly among all options for the move;
        perform the move;
        if move is accepted then
            update cut size;
        else
            reverse the move;
        end
        increase move counter;
    end
    if max number of moves for each T is reached then
        if no update over the last T then
            break
        else
            update T;
        end
    end
until max number of moves is reached;
update the final partitioning result;
```

Algorithm 3 shows how SA is implemented in our work. The difference between our implementation and the work in [5] is that we consider area ratio as a strict rule. In other words, size ratio constraint always holds. The only cost is the difference of the current cut size and new one. In each move, a random cell A is chosen to move to its opposite partition. Next we choose a cell B in the opposite partition such that swapping these two cells does not break the constraint. In addition, we assert whether the size ratio constraint would be violated if A is the only cell that moved. If it does not, we also accept this move as a single-cell move. Finally, swapping or single-cell move is decided randomly. New cut size as well

as cost is computed afterwards, and this move is accepted if the following condition is satisfied:

$$r < \exp \frac{-dE}{T} \quad (1)$$

where r is a random real value in range [0,1], dE is the cost which equals to difference of the current cut size and new cut size, and T is the current temperature. If the cost is smaller than 0, that move is always accepted.

IV. Experimental Results

A. Testing Conditions

We perform the algorithms on twelve benchmark circuits of different sizes. The size ratio of one partition is set to 0.5 with a tolerance of 0.1 of the total size. In other words, the ratio of the size of one partition to the total size should be from 0.4 to 0.6. The test is repeated 10 times for each algorithm on each circuit. We report both the minimum cut size and the average cut size of 10 trials along with the execution time for one call. The reported cut sizes are calculated on nets of hypergraph (directly for FM and SA, and indirectly for KL). For SA algorithm, the temperature is initially set to 10,000, and then reduced by half at each update. At most 100 updates are allowed, and the number of moves at each temperature is fewer than 25% of the number of cells. These parameters are selected by observation. Generally, the outcome is better if the number of moves is large, but it will take more time to finish.

All three algorithms are implemented in C. Tests are done on a PC with the following specifications: OS Ubuntu 18.04, Intel(R) Core(TM) i5-4670 CPU @ 3.4GHz and 8GBs of RAM.

B. Results and Discussion

Our results are similar to the ones published in [6]. Based on our implementation, the size ratio constraint is always satisfied by FM and SA. KL may have this property depending on circuit specification. As shown in Fig.1, execution time of FM and SA has linear dependence on number of cells. KL seems to not only depend on number of cells however.

For the small circuits including 'test', 'test2' and 'test3', all three algorithms finish in less than 0.01s, giving the smallest possible cut size (this can be manually verified). However, FM and SA have corresponding average cut sizes higher than the minimum one. That means these algorithms are unstable and may get trapped in local minimum if being badly initialized.

FM algorithm always finishes in less than 1 second which is usually much faster than two others as shown in Table I. Size ratio constraint is always satisfied in FM. However, it only reports the smallest cut size for 2 circuits.

KL reports the smallest cut size and also satisfies the area constraint on 4 circuits. However, it fails the area constraint on 5 other big circuits. The reason is that in KL, both partitions should have the same number of cells or a difference of exactly 1 if the total number of cells is even and odd respectively. This specific constraint may conflict with the size ratio one if the standard deviation of cell sizes is high enough.

SA reports the smallest cut size for 3 circuits, but it is usually the slowest one. The reason for this may come from its parameters. In general, SA works well on the benchmarks circuit. It requires higher number of moves to converge as the circuit gets bigger.

V. Conclusions

We survey three common algorithms, including FM, KL and SA for the partitioning problem in VLSI. All algorithms are implemented and tested on twelve benchmark circuits to assert their performance in terms of cut size and execution time. The results show that they work equally well in general. FM algorithm runs faster than the others, but its resulting cut size highly depends on the initial partitioning process. KL usually reports the best cut size and also satisfies the area ratio constraint on the circuit which has uniform cell size distribution (low standard deviation). SA balances between constraint and performance, and may require a set of carefully-tuned parameters to achieve the best result.

References

[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[2] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th design automation conference*. IEEE Press, 1982, pp. 175–181.

[3] H. H. Yang and D. Wong, "Efficient network flow based min-cut balanced partitioning," in *The Best of ICCAD*. Springer, 2003, pp. 521–534.

[4] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning," *Operations research*, vol. 37, no. 6, pp. 865–892, 1989.

[5] C.-W. Yeh, C.-K. Cheng, and T.-T. Lin, "Optimization by iterative improvement: an experimental evaluation on two-way partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 2, pp. 145–153, 1995.

[6] J. Cong, W. J. Labio, and N. Shivakumar, "Multiway vlsi circuit partitioning based on dual net representation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 4, pp. 396–409, 1996.

TABLE I
COMPARISON OF FM, KL AND SA PERFORMANCE ON 12 BENCHMARK CIRCUITS

Circuit	#Cells	#Nets	Size	Min Cut Size			Average Cut Size			Execution Time (s)		
				FM	KL	SA	FM	KL	SA	FM	KL	SA
primary1	752	1266	12,027,000	129	114	124	161.9	180.8	146.8	0.04	0.22	0.53
primary2	2907	3817	34,197,000	589	463	480	614.6	659.6	545.4	0.62	4.36	11.69
primga1	752	1266	13,402,000	129	124	130	156.2	170.7	152.3	0.04	0.23	0.50
primga2	2907	3817	33,064,000	596	417	507	636.2	572.0	553.6	0.58	4.65	11.52
test	9	11	81,000	2	2	2	2.3	2.8	5.3	-	-	-
test02	1602	2308	194,114,576	164	533*	162	218.8	578.2	229.0	0.17	2.89	2.96
test03	1550	2338	75,437,360	171	227*	167	240.6	330.7	238.4	0.14	2.07	3.33
test04	1489	2189	143,110,460	44	520*	114	84.5	585.6	215.5	0.07	3.27	2.51
test05	2540	3488	244,677,324	42	829*	124	176.7	905.3	300.3	0.24	11.05	7.40
test06	1691	2048	57,483,800	280	554*	212	299.0	610.9	237.5	0.19	3.19	3.67
test2	16	28	144,000	2	2	2	2.3	2.0	10.1	-	-	-
test3	4	8	36,000	4	4	4	5.6	4.0	4.0	-	-	-

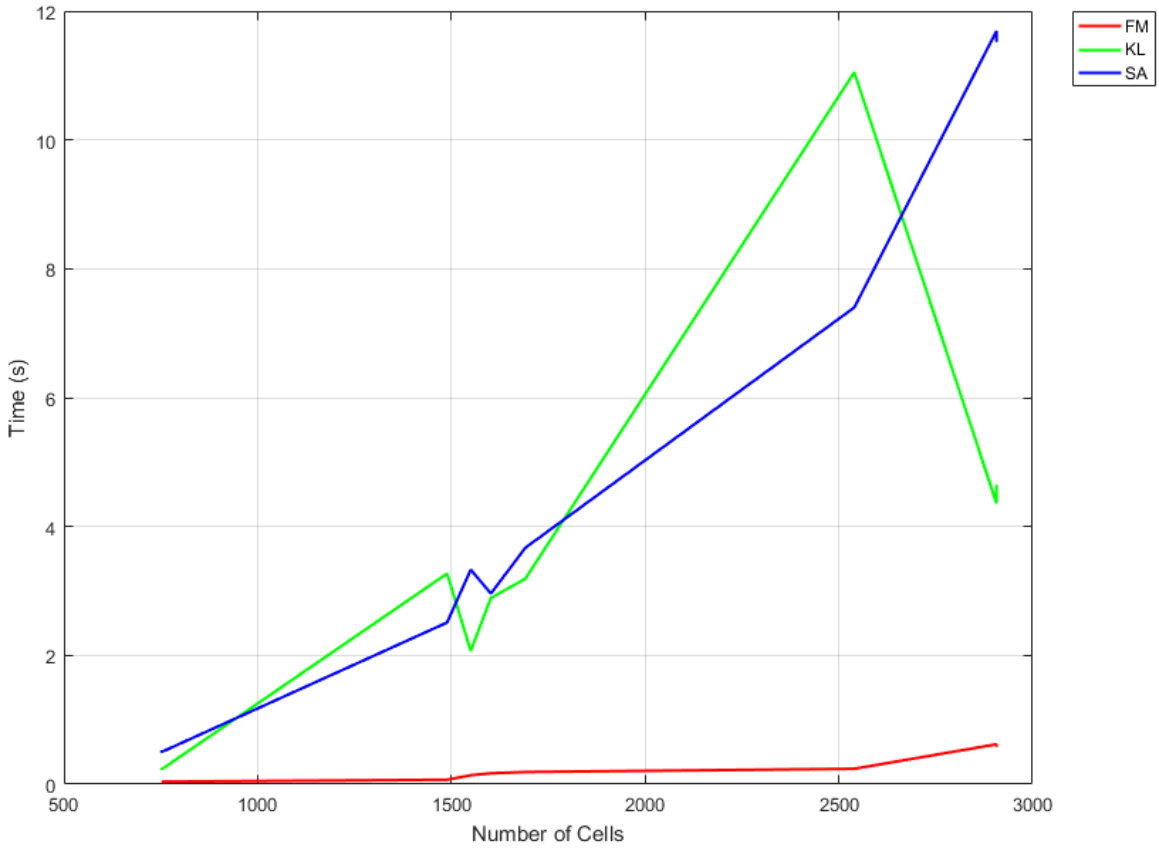

Fig. 1. Relationship between execution time and circuit complexity