Physical Design Automation

HW2 FM Algorithm Implementation

109062526 楊皓崴

1. How to compile and execute your program, with execution example

STEP1. Access the correct directory => cd HW2/src

STEP2. Compilation => make

STEP3 Execution => ../bin/<exe> <net file> <cell file> <output file>

e.g.: ../bin/hw2 ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output

STEP4. Verify= > ../verifier/verify <testcase.nets> <testcase.cells> < output file>

e.g.: ../verifier/verify ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output

```
--How to Compile
  In ~/HW2/src directory, enter the following command:
  $ make
  It will generate the executable file "hw2" in "~/HW2/bin/"

  $ make clean
  It will remove the executable file

--How to Run
  In ~/HW2/src directory, enter the following command:
  $ ../bin/<exe> <net file> <cell file> <output file>
  e.g.:
  $ ../bin/hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.output

  In ~/HW2/bin directory, enter the following command:
  $ ./<exe> <net file> <cell file> <output file>
  e.g.:
  $ ./hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.output
```

```
[g109062526@ic51 ~]$ cd HW2/src
[g109062526@ic51 src]$ make
g++ -std=c++11 main.cpp utils.cpp class.cpp algo.cpp -o ../bin/hw2
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output
INIT CUT = 148688
MIN CUT = 130513
16.862u 0.316s 0:17.23 99.6%    0+0k 0+2480io 0pf+0w
[g109062526@ic51 src]$ ../verifier/verify ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output

/*********************************************************************/

OK!! Your output file satisfy our basic requirements.

/*********************************************************************/
```

2. The final cut size and the runtime of each testcase

Testcase	Final Cut Size	Runtime (sec)
1	6	0.02
2	266	0.05
3	3037	2.82
4	47560	9.44
5	130513	15.75

```
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.output
INIT CUT = 84
MIN CUT = 6
0.007u 0.005s 0:00.01 0.0%      0+0k 0+8io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-2.nets ../testcases/p2-2.cells ../output/p2-2.output
INIT CUT = 917
MIN CUT = 266
0.045u 0.009s 0:00.05 80.0%     0+0k 0+72io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-3.nets ../testcases/p2-3.cells ../output/p2-3.output
INIT CUT = 27569
MIN CUT = 3037
2.682u 0.127s 0:02.82 99.2%     0+0k 0+1416io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-4.nets ../testcases/p2-4.cells ../output/p2-4.output
INIT CUT = 64371
MIN CUT = 47560
9.223u 0.185s 0:09.44 99.5%     0+0k 0+2480io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output
INIT CUT = 148688
MIN CUT = 130513
15.455u 0.252s 0:15.75 99.6%    0+0k 0+2480io 0pf+0w
```

3. I/O and Computation Time Analysis

Input cells: O(# of cells)

Input nets: O(# of pins)

Computation: O(# of pins)

In this part, the timing may be slightly different from the result shown above since I added several lines of codes to separately measure the run time of I/O and algorithm computation.

Testcase	I/O time	Computation time
1	0.000000	0.000000 (too short to measure)
2	0.03	0.01
3	0.61	2.17
4	1.22	8.07
5	4.49	11.88

```
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.output
I/O time: 0.000000
INIT CUT = 84
MIN CUT = 6
Computation time: 0.000000
0.006u 0.005s 0:00.01 0.0%      0+0k 0+8io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-2.nets ../testcases/p2-2.cells ../output/p2-2.output
I/O time: 0.030000
INIT CUT = 917
MIN CUT = 266
Computation time: 0.010000
0.047u 0.010s 0:00.06 83.3%     0+0k 0+72io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-3.nets ../testcases/p2-3.cells ../output/p2-3.output
I/O time: 0.610000
INIT CUT = 27569
MIN CUT = 3037
Computation time: 2.170000
2.868u 0.144s 0:03.02 99.3%     0+0k 0+1416io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-4.nets ../testcases/p2-4.cells ../output/p2-4.output
I/O time: 1.220000
INIT CUT = 64371
MIN CUT = 47560
Computation time: 8.070000
9.344u 0.254s 0:09.64 99.4%     0+0k 0+2480io 0pf+0w
[g109062526@ic51 src]$ time ../bin/hw2 ../testcases/p2-5.nets ../testcases/p2-5.cells ../output/p2-5.output
I/O time: 4.490000
INIT CUT = 148688
MIN CUT = 130513
Computation time: 11.880000
16.463u 0.349s 0:16.86 99.6%    0+0k 0+2480io 0pf+0w
```

4. The details of your implementation

There are 4 .cpp files in this homework.

main.cpp: the overall main file that perform this task

class.cpp: definition of the cell and net object data structure

utils.cpp: methods of reading input files (building cell/net objects) and output

algo.cpp: methods of FM algorithm

- What is the difference between your algorithm and FM algorithm described in class.

Most of the logic is exactly the same as described in class. I think the biggest difference is at the way to check balance between two partition. In the homework spec, the balance condition is described as $\text{abs}\big(\text{area(A)} - \text{area(B)}\big) < \frac{total\_area}{10}$. The second difference is that I implement a EARLY_STOP() to accelerate, that is, I stop the program if the cut did not improve for 10 iterations.

The second difference is that I repeated the FM_algorithm several times. Since that the initial partition is critical to the result, I use the previous FM_algorithm output partition as the next iteration's initial partition. In this manner, the cut is guaranteed to decrease in each iteration. An also, since my program run relatively fast in the original version, I have plenty of time before TLE to repeat the FM algorithm.

- Did you implement the bucket list data structure?

I did not handcraft a class object named as bucket_list. However, I directly make use of the "map" data structure in C++ which is more convenient . In map<int, vector<Cell

*> >, int is the gain of the corresponding vector of cells.

- How to find the max partial sum and restore the result?

I store a sequence of moved cells and an index pointing at the best result in current iteration. After the algorithm terminate, I move back the cell in the move_sequence indexed after the best_index.

- What did you do to enhance the solution quality or speed up your program?

In my observation, the solution quality is related to two parts. The first part is the way to initialize the partition. The second part is the order of cells in bucket_list's vector. For the first part, initially, I used random() to decide which partition a cell should be placed in and the result is unpromising. Therefore, I change the random method. I try to put the first half of cells in cell_list in part A and the second half of cells in part B and I got a large improvement. For the second part, I try traverse the bucket_list's vector forward or backforward direction. The result show that if I traverse the vector backward the run time will be shorter and the cut size is generally smaller. Lastly, I try to repeat FM algorithm as I mentioned above to further improve my MIN CUT.

- If you implement parallelization, please describe the implementation details and provide some experimental results.

I did not implement parallelization.

5. Result comparison with top 5 of last year, show your advantage either in runtime or in solution quality. Are your results better?

	Cut Size					Runtime(sec)				
ranks	p2-1	p2-2	p2-3	p2-4	p2-5	p2-1	p2-2	p2-3	p2-4	p2-5
1	6	191	4441	43326	122101	0.01	0.07	3.05	5.01	42.06
2	6	161	1065	43748	125059	0.01	0.1	3.11	9.84	18.77
3	6	358	2186	45430	122873	0.04	0.78	21.21	115.38	59.78
4	5	302	1988	46064	124862	0.03	0.17	7.04	6.93	8.22
5	6	411	779	46356	125151	0.01	0.16	5.49	12.31	13.57
mine	6	266	3037	47560	130513	0.02	0.05	2.82	9.44	15.75

If so, please express your advantages to beat them

Most of my run time is way better than theirs. In the beginning, my code does not

run this fast. There are mainly 3 part that I did to improve the runtime.

Firstly, and also the most significantly, there are a lot of pointer in my code and I wrote a lot of things to access a specific data such as “moved_cell->connectedNets[i]->adj[j]”. In code like this, accessing through”->” is very time consuming. Therefore, I modify most of these “->” by using a temporary variable to store it.

Secondly, iterating the bucket_list’s vector backward can improve the run time by several seconds.

Thirdly, I deleted unused data variables in I/O such as number of pins. This can improve the time in I/O.

For the solution quality, my min_cut in p2-1, p2-2 and p2-3 is comparable with the top-5 results. Especially in p2-2 and p2-3, my results surpass some of the top-5 results. However, in p2-4, 2-5, my results are relatively worse than top-5s.

I think the reason that some of the min_cut is not that good is because that I did not do any optimization in the vector of bucket_list. Each vector just stores a sequence of Cell pointers. If I optimize it by sorting that array with cell->pin or cell->size. Perhaps it could be possible that the result may improve. However, the running time might be sacrificed.

6. What have you learned from this homework? What problem have you encountered?

First of all, after implementing the FM algorithm, I have a further understanding of the details of the algorithm, such as the order of cells inside bucket list and how to iterate it can affect the performance of the program. Besides, I found out that accessing data member via “->” e.g., ptr->member is very time-consuming. Just modifying this part can save up to 50% of the runtime originally. Lastly, in this homework, utilization of pointer is very common, this helps me refresh the concept of memory in C++.

As for the problem I have encountered, firstly, is about checking the criticality after moving a particular cell. Due to the reason that I have move a cell previously in move_max_gain_cell_and_update_area(), I cannot directly break the loop if the cur_cell is locked. On the contrary, "continue” shall be use here. This is the primary bug that take me a while to debug during implementing the algorithm. The second problem I encounter is about the performance of the program. In the beginning, the running time and cut size is both inferior. After I decrease the use of “->”, try different way to initialize partition and try to traverse the bucket_list vector backward, the performance is better.