Name: 方帷愷

Student ID: 110062636

How to compile and execute the program

To compile the program, we can use the following commands in the **`HW2/src/`** directory:

Commands	Description
`$ make`	It will generate the executable files **hw2** and **`hw2_parallel`** in **`HW2/bin/`**
`$ make main`	It will generate only the executable file **hw2** in **`HW2/bin/`**
`$ make parallel`	It will generate only the executable file **`hw2_parallel`** in **`HW2/bin/`**

To execute the program, we can use the following command in the **`HW2/src/`** directory:

```
Usage: ../bin/<exe> <net file> <cell file> <output file>
```

```
e.g.: $ ../bin/hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.out
```

Or in the **`HW2/bin/`** directory:

```
Usage: ./<exe> <net file> <cell file> <output file>
```

```
e.g.: $ ./hw2 ../testcases/p2-1.nets ../testcases/p2-1.cells ../output/p2-1.out
```

Final cut size and runtime

Cases	p2-1	p2-2	p2-3	p2-4	p2-5
Cut size	5	125	597	45173	123732
Runtime (s)	0.02	0.46	27.96	164.81	218.29

Analyze the runtime

Cases	p2-1	p2-2	p2-3	p2-4	p2-5
I/O time (s)	0.01	0.01	0.24	1.04	2.47
CPU time (s)	0.01	0.45	27.72	163.77	215.82

The program spent most of the time on the FM Algorithm, and about 1 second processing the I/O.

Details of the implementation

The implementation of my FM Algorithm is almost the same as the one described in class, except instead of finding the maximum partial sum and restore the result at the end of every pass, I store the best result directly every time I found a better result.

I implemented 2 bucket lists, one for set A and one for set B. Every time I move a cell, I remove it from the bucket list without inserting it into the other bucket list, so I reconstruct the bucket list before the next pass.

Instead of finding the maximum partial sum and restore the result, I store the best result directly every time I found a better result, and then use the best result as the input for the next pass.

To enhance the solution quality, I used random initial partition instead of rule-based partition, and use the best initial partition that will give the best solution quality.

To speed up my program, I only save best result when the cut size is starting to grow (i.e. Gain of move cell is less or equal to 0). Instead of saving the best set A and best set B, I only save best set A, this made my program to run faster than before.

I implemented parallelization using **`std::thread`** and **`std::mutex`**. However, if I try to parallelize the **`Update_Gain`** part of the program, it will result in a much longer runtime as shown in the following table:

Cases	**p2-1**	**p2-2**
Cut size	5	408 (Forced termination by timer)
Real (s)	0.94	295.35
User (s)	1.18	450.01
Sys (s)	0.21	4.37

I think the reason for this is there are a lot of data dependencies when updating the cell gains, causing the program to run longer than the serial version.

Compare results with top 5 students' results from last year

My program has better solution quality in the first 3 cases compared to the results from last year, and has about the same solution quality in the last 2 cases.

I think the key point of beating them is the initial partition, I tested a lot of initial partition using random seeds and found out that different initial partition can result in a very different solution quality, especially in the first 3 cases.

What I have learned from this homework

In this homework, I learned how to optimize the runtime of my program by choosing the most suitable data structure. I also improved my programming skills by debugging the program and knowing why the program didn't work as expected.