EDA HW2 Report

許育棠 113062556

How to compile and execute

How to Compile

In "HW2/src/", enter the following command:

$ make

How to Run

Usage:

$ ./hw2 <txt file> <out file>

An executable file "hw2" will be generated in "HW2/bin/".

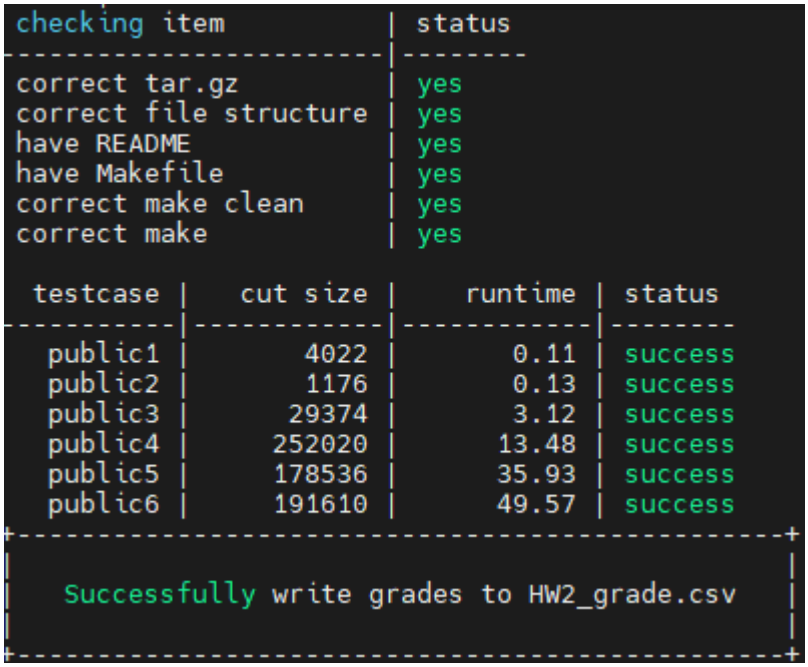
If you want to remove it, please enter the following command:

$ make clean

E.g., in "HW2/bin/", enter the following command:

$ ./hw2 ../testcase/public1.txt ../output/public1.out

Result of HW2_grading.sh



Algorithm Details

1. Initialization & Input Reading

The program starts in the main() function where it first parses command-line arguments (input and output file names). Then, it calls the processInput() function to read the input file. During this step, the program creates objects for Tech, LibCell, Cell, and Net, and sets up their relationships. It also initializes the Die objects (DieA and DieB) based on the input parameters and calculates each cell's area to decide which die it should initially belong to (to the one with lower area).

2. Clustering

---------- Pseudo code----------

Function Cluster:

clear dieA and dieB assignments

sort nets by descending weight

for each net in sorted order:

if all cells connected by net are unvisited:

form a cluster with those cell indices

compute total area for cluster in DieA and DieB

for each unvisited cell:

form a single-cell cluster

sort all clusters by the absolute difference |areaA - areaB|

for each cluster:

assign the cluster to the die where its area is lower, ensuring the die's max area is not exceeded

lock one cell in the cluster

update totalAreaA or totalAreaB accordingly

while a die exceeds its utilization constraint:

adjust by moving cells from the overutilized die to the other

for each net:

update net cell counts based on final cell assignments

The clustering function first sorts the nets by descending weight and groups cells connected by each net into clusters, calculating the total area of each cluster if assigned to DieA versus DieB to determine their area differences; cells that aren't grouped by any net become individual clusters. The clusters are then sorted in descending order of area difference so that clusters with the highest imbalance are assigned first, and each cluster is allocated to the die where it adds the least area—while ensuring that the total area does not exceed the utilization limits—by checking both dies and adjusting as needed to satisfy the maximum utilization constraints.

3. FM Algorithm Optimization

----------Pseudo code----------

function FM:

Initialize currentSum, maxSum, and an empty moveStack.

while a cell with positive gain exists:

Move the cell, update gains, and add it to moveStack.

Increase currentSum by the cell's gain.

Update maxSum and record the moveStack size if currentSum is highest.

Break if currentSum becomes negative.

Roll back the moves made after the point where maxSum was reached.

return maxSum

With the initial partition in place, the program computes each cell's gain—which reflects the impact of moving that cell on the overall cut size—using the computeGain() function. Then, the FM algorithm, implemented via functions such as getMaxGainCell(), moveCell(), and updateGain(), iteratively moves cells between DieA and DieB to reduce the cut size; it terminates early if the running partial sum becomes negative or if the maximum gain is less than zero, indicating that further moves would worsen the partition. The FM process is capped at two iterations and will stop if the maximum partial sum returned by an FM run is negative.

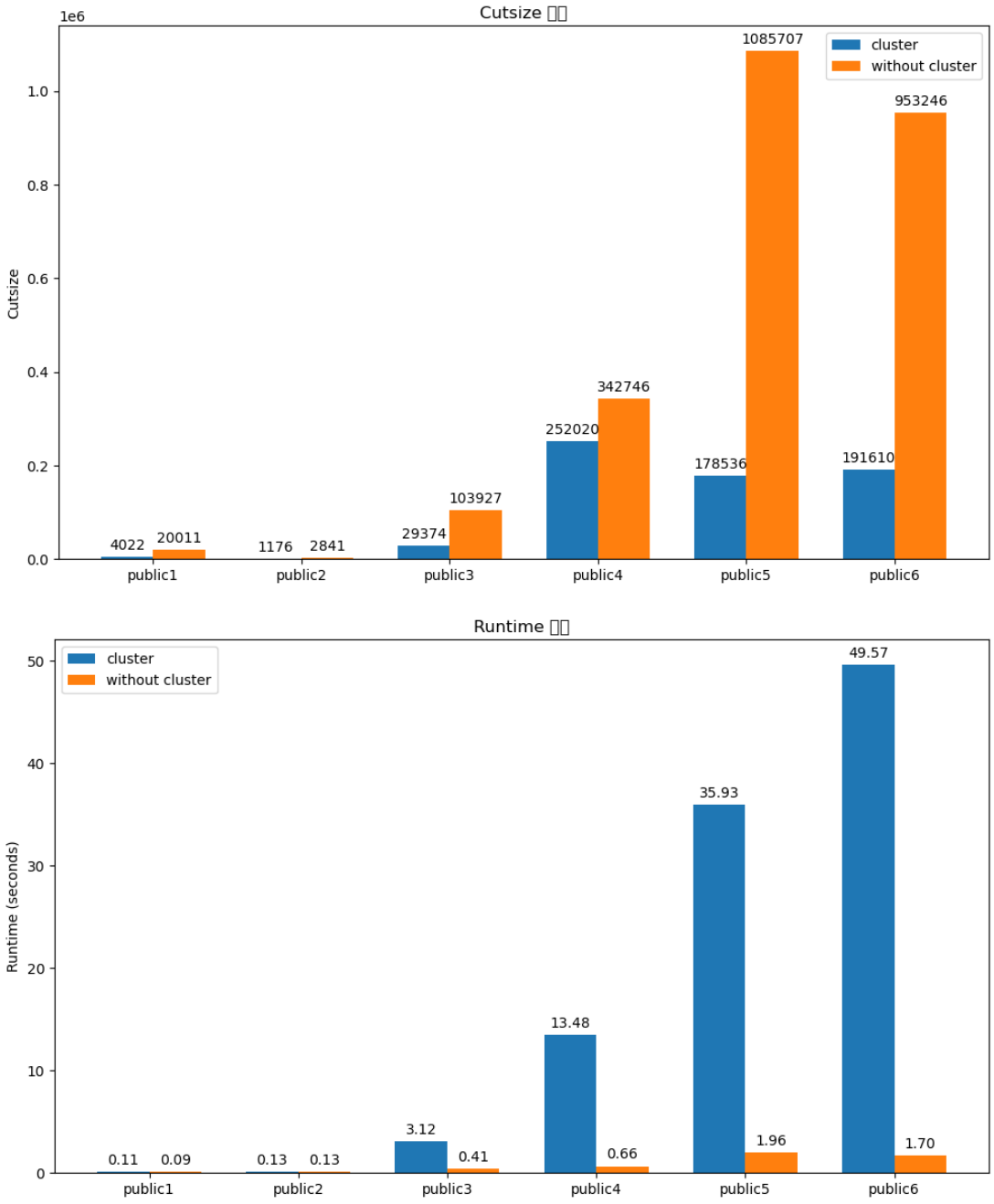
4. Output & Cleanup

Finally, the program computes the final cut size (using getCutSize) and writes the results (the list of cells assigned to each die and the cut size) to the output file via processOutput. The program then cleans up by releasing dynamically allocated memory and outputs the total execution time before exiting.normal cells. If we don't insert well tap cells and use tapless standard cells, it will cause latch-up. Latch-up occurs when parasitic PNPN thyristors in CMOS circuits create a low-impedance path between VDD and VSS, leading to high current flow and possible chip failure.

Techniques Used

1. Clustering

By using this method, the net with higher weight would not be cut. The cut size would be relatively small after this partition method. Then, the time spent on FM would be smaller since there is little refinement needed.

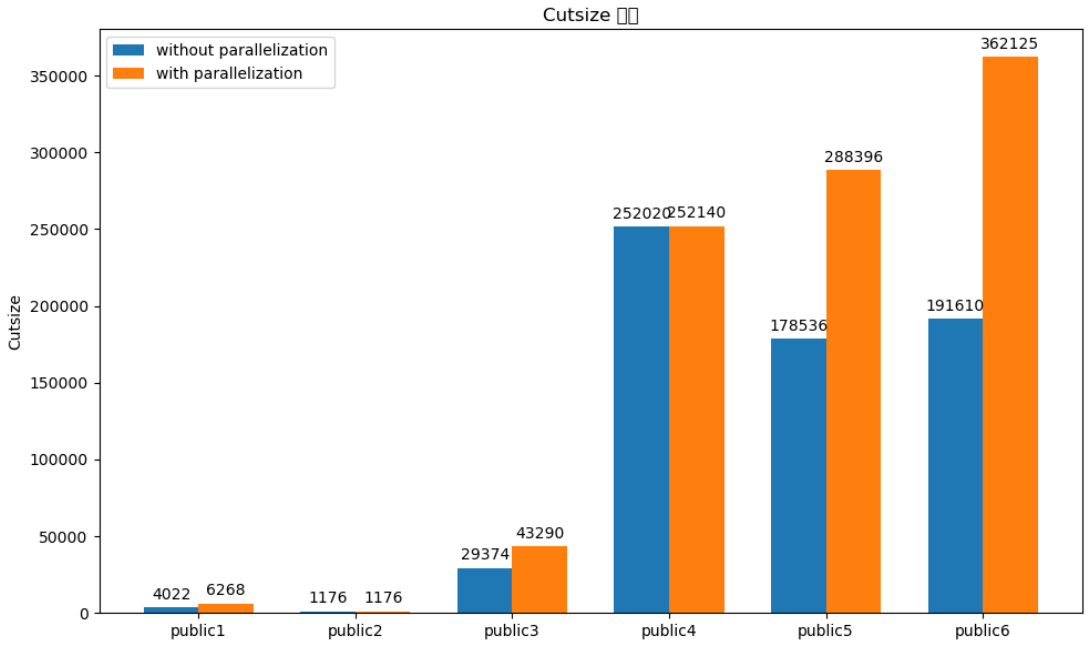


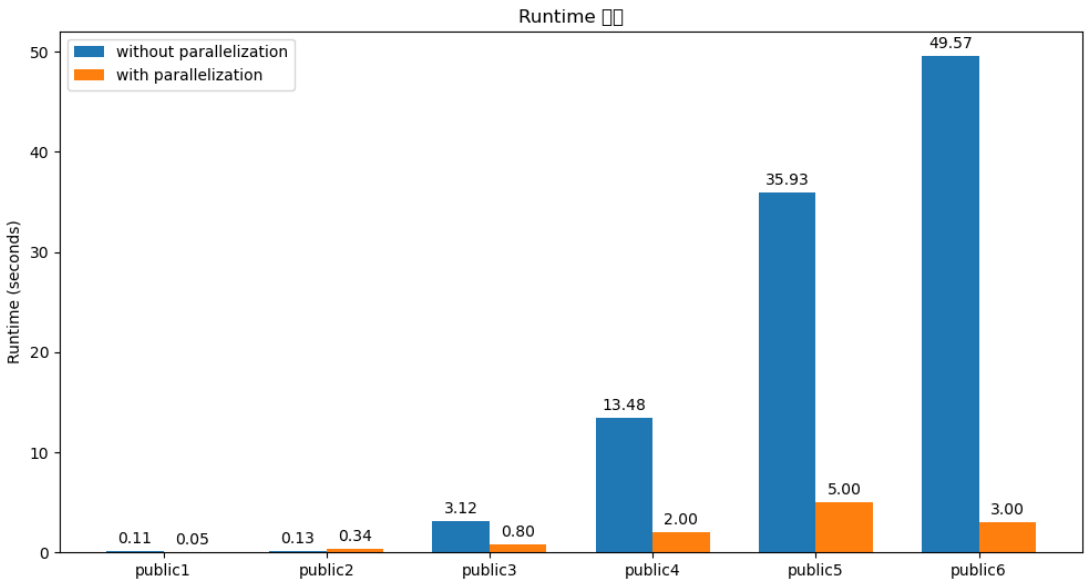
According to the result, using clustering can generate better solution quality but requires more time. The reason is that our FM has some constraints that will let it stop earlier. For example, each cell will only be selected once even though the move is illegal. If our FM is greedy, the solution quality without clustering would not be so bad but it

would cost much more time than using clustering.

Parallel Implementation

My OpenMP implementation focuses on parallelizing updating_gain() of the FM algorithm, since the bottleneck of the program is bounded by FM. We use directives like #pragma omp parallel for to distribute iterations among threads, while carefully protecting shared data (such as gain buckets and net counts) with critical sections or atomic operations. This allows threads to compute local updates in parallel and later merge changes safely, thus reducing synchronization overhead, even though the outer loop remains sequential due to dependency on the current state.





The experimental results indicates that while the parallelized approach tends to yield higher cutsize values, it dramatically reduces runtime across most test cases. For instance, public1, public3, public5, and public6 all show increases in cutsize when parallelization is applied, yet their runtimes drop significantly—from 49.57 seconds to 3 seconds for public6, for example. In contrast, public2 maintains the same cutsize with a slight increase in runtime, and public4 shows minimal difference in cutsize with a substantial runtime reduction. This illustrates a clear trade-off of my parallel implementation: parallelization accelerates processing time considerably at the cost of achieving slightly higher cutsize results.

What Have I Learned

This homework was challenging. I encountered two major issues: the FM algorithm's running time became too long due to the large number of cells in some cases, and ensuring all cells fit within the die while satisfying utilization constraints was difficult. I learned that a well-chosen initial partition can significantly reduce the cut size quickly, and that relaxing the termination condition of FM can effectively control its running time. Overall, by completing this homework, I gained valuable insights into solving large-scale two-

way min-cut partition problems efficiently.