CS6135 VLSI Physical Design Automation - Report
Homework 2: Two-way Min-cut Partitioning

(1) 姓名:張芝岑 學號:113065702

(2) How to compile and execute your program, and give an execution example.

1. cd src/

2. make

3. cd ..

4. cd bin/

5. $ ./hw2 ../testcase/public1.txt ../output/public1.out(continue untill public6)

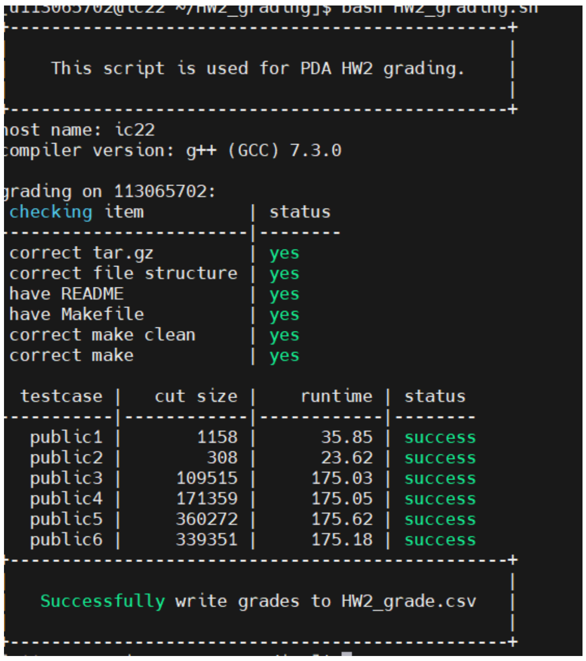
6. cd ..

7. pwd (with the bash file)

8. bash HW2_grading.sh

(3) The final cut size and the runtime of each testcase. Paste the screenshot of the result of running the HW2_grading.sh as the picture shown below.

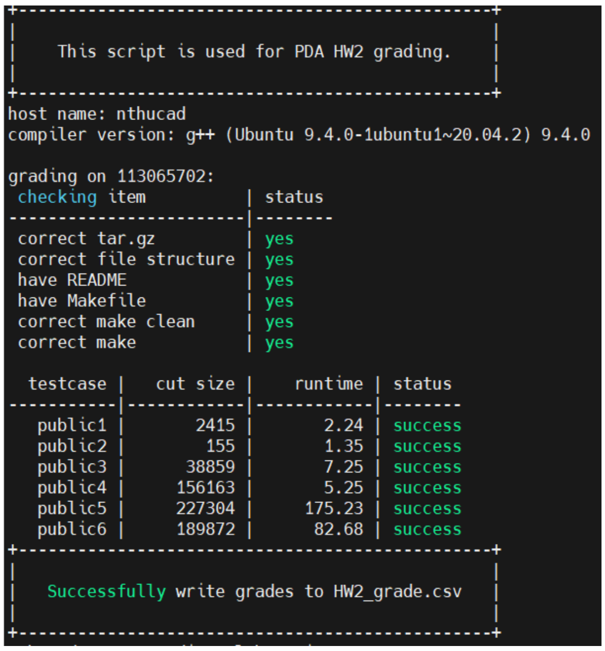
Figure 1. Baseline Version **Figure 2. Parellel Version Bonus**



```
u113065702@ic22 ~/HW2_grading]$ bash HW2_grading.sh
+------------------------------------------------------+
|                                                      |
|      This script is used for PDA HW2 grading.        |
|                                                      |
+------------------------------------------------------+
host name: ic22
compiler version: g++ (GCC) 7.3.0

grading on 113065702:
 checking item          | status
------------------------|--------
 correct tar.gz         | yes
 correct file structure | yes
 have README            | yes
 have Makefile          | yes
 correct make clean     | yes
 correct make           | yes

  testcase |   cut size |      runtime | status
-----------|------------|--------------|--------
   public1 |       1158 |        35.85 | success
   public2 |        308 |        23.62 | success
   public3 |     109515 |       175.03 | success
   public4 |     171359 |       175.05 | success
   public5 |     360272 |       175.62 | success
   public6 |     339351 |       175.18 | success
+------------------------------------------------------+
|                                                      |
|    Successfully write grades to HW2_grade.csv        |
|                                                      |
+------------------------------------------------------+
```



```
+------------------------------------------------------+
|                                                      |
|      This script is used for PDA HW2 grading.        |
|                                                      |
+------------------------------------------------------+
host name: nthucad
compiler version: g++ (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0

grading on 113065702:
 checking item          | status
------------------------|--------
 correct tar.gz         | yes
 correct file structure | yes
 have README            | yes
 have Makefile          | yes
 correct make clean     | yes
 correct make           | yes

  testcase |   cut size |      runtime | status
-----------|------------|--------------|--------
   public1 |       2415 |         2.24 | success
   public2 |        155 |         1.35 | success
   public3 |      38859 |         7.25 | success
   public4 |     156163 |         5.25 | success
   public5 |     227304 |       175.23 | success
   public6 |     189872 |        82.68 | success
+------------------------------------------------------+
|                                                      |
|    Successfully write grades to HW2_grade.csv        |
|                                                      |
+------------------------------------------------------+
```

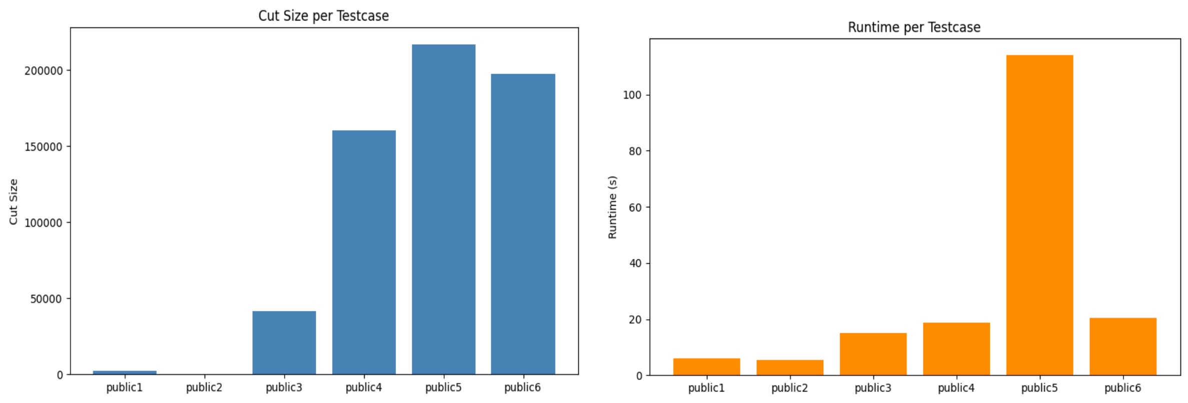


Figure 3-4. Baseline Version

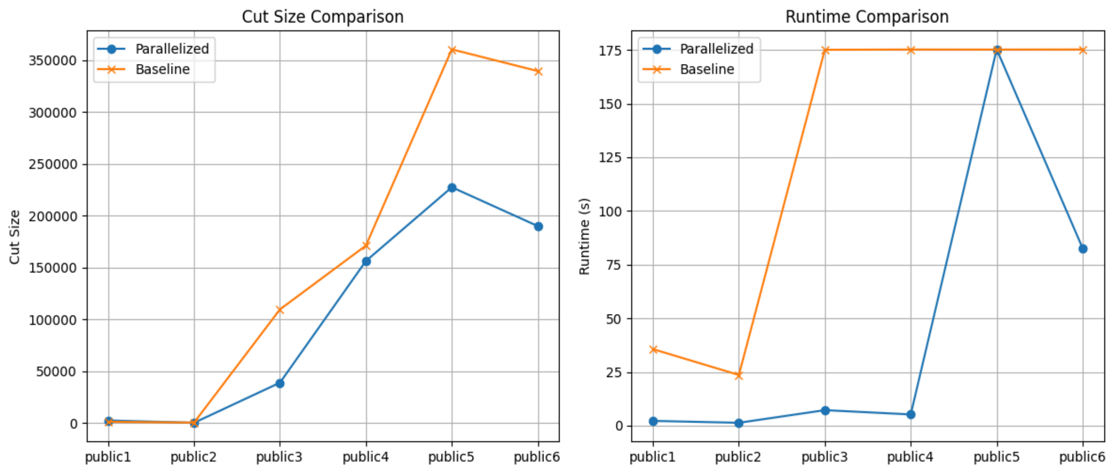


Figure 5. Baseline Version compares to Parallelized version of cut size and runtime

(4) The details of your algorithm. You could use flow chart(s) and/or pseudo code to help elaborate your algorithm. If your algorithm is similar to some previous work, please cite the corresponding paper(s) and reveal your difference(s).

To enhance the runtime performance of my implementation, I applied “**#pragma omp parallel for”** parallelizing computationally intensive loops using OpenMP. As a result, I chose to compile my solution with **g++ 9.4.0(nthucad)**, which offers more mature and robust support for OpenMP compared to earlier versions.

The reasons for this selection are “**efficient parallel loop handling**”, for the reason that g++ 9.4.0 provides optimized implementations for #pragma omp parallel for, allowing better thread management and reduced overhead.

While the smarter thread scheduling and cache-aware strategies means this version leverages improved load balancing and minimizes cache thrashing, enhancing execution efficiency in multi-threaded environments.

In contrast, older compiler versions **(**e.g., **g++ 7.3.0 or 9.3.0)** often fail to fully utilize OpenMP features or may lead to undesirable effects such as thread contention, false sharing, and load imbalance. Additionally, I also realize that the older compilers may struggle with nested parallel loops or critical section handling, resulting in degraded performance or incorrect behavior. Therefore,I believe using a newer compiler like g++ 9.4.0 was essential for realizing the full benefits of parallelization and ensuring consistent, high-performance execution across all test cases.

I have using the Fiduccia Mattheyses (FM) Algorithm In this homework, I use a modified Fiduccia Mattheyses algorithm to address the two-way min-cut partitioning problem under area and technology constraints. The design builds on the classical FM algorithm and introduces enhancements to handle weighted nets, heterogeneous technology libraries, and area legality. The complexity of this step is O(n log n). While I design the Gain Bucket (Partition::bucket) to be Sorted buckets indexed by gain. And using the Rollback List (gi_prefix_sum) continues to maintain cumulative gain sequence. The function of Cell Locking is to prevent repeated movement in a single FM pass.

Randomization of Equal-Gain Cells, will allow cells with the same gain to be shuffled before processing. Prevents deterministic local traps and improves diversity of explored solutions. Rollback Strategy will move sequences recorded in each pass. After the pass, the algorithm reverts to the point yielding the maximum gain. Gain values account for the weights of nets fully, unlike the classical FM that treats all nets equally.

(5) What techniques did you use to improve your solution's quality? Additionally, analyze how they contributed to the improvements. Plot the effects of those different settings like the ones shown below.

Initial Partition (like MIN_DIFF) can assign cells based on area difference under two technologies and starts with a balanced area. Gain Bucket with Rollback can record sequence of moves and undo if accumulated gain is negative and avoids greedy traps. While the randomized Tie-breaking can shuffle equal-gain candidates to diversify search path and Breaks symmetry, enhances local optima

For the part of using the parallelization with OpenMP, it accelerates computationally expensive sections like cut evaluation, gain initialization, and gain updates during FM refinement and also maintains correctness and cut size quality. With the use of it, it has reduced runtime dramatically, like testcase public6 needs 82s vs 175s in the original one.

I also use the gain bucket with lazy update, as it uses a priority queue to greedily select the best move without recomputing all gains every iteration. It allows more balanced quality and speed and fast convergence to local optima, while prevents quality degradation during refinement at the same time.

(6) If you implement parallelization (for the algorithm itself), please describe the implementation details and provide some experimental results.

I parallelized the initial gain computation phase using OpenMP. Since each cell's gain is computed independently before the first move, this step is highly parallelizable.

```
#pragma omp parallel for

for (int i = 0; i < C; i++) {

    compute_gain(i);

}
```

Stage	Parallelization Method
Initialization	Parallel cell assignment with critical section for area update
Gain Computation	Parallel loop over nets with thread-local aggregation
Gain Bucket Traversal	Parallel traversal with synchronization during move execution
Cutsize Evaluation	Parallel reduction across all nets

(7) What have you learned from this homework? What problem(s) have you encountered in this homework?

Through using the FM algorithm, it is highly adaptable when extended to hardware-aware constraints. Parallelization and careful gain management can result in significant performance improvements. While the rollback mechanism ensures that only the most effective moves are retained. Generating a legal and balanced initial partition proved difficult, especially in dense benchmarks. But shared state updates in OpenMP regions required careful management using atomic or critical constructs. Therefore, in some test cases, improvement stalls after 2–3 passes, possibly due to area limits.