Exploring Parallelism Opportunities in KLayout (project report, group 22)

Zeren Chen
zerenat.owlfox<dot>org
Taiwan

ChatGpt4
Open AI
USA

MH Yan
Taiwan

ABSTRACT

This project explores the introduction of parallelism into KLayout Design Rule Checking (DRC) processes to enhance performance. We initially attempted to improve single DRC operation parallelism using OpenMP, but faced challenges related to c++ code base complexity and race conditions. Therefore, we transitioned to use Taskflow for task level parallelism, which provides promising performance improvements based on preliminary experiments, showcasing Taskflow's capabilities. The code and experiment scripts can be found at https://github.com/owlfox/pps23_final_project.

KEYWORDS

KLayout, DRC, Parallel Programming, OpenMP, Taskflow, Open-Source, Work-In-Progress T.T

1 INTRODUCTION

The growing complexity of modern integrated circuit designs has necessitated faster DRC processes. KLayout, a GPL open source C++ layout viewer/editor/analyzer, first released in 2006-04, maintained by Matthias **K**oefferlein, has been chosen as the project to explore if classroom learned parallelism techniques can be applied in real-world C++ code base.

2 PROPOSED SOLUTION

We initially proposed introducing parallelism into KLayout's DRC scripts using OpenMP. Our exploration originally focused on the DRC code that computes derived layer shapes given two or more input layers, such as:

Listing 1: DRC code used for benchmarking in KLayout

```
m1 = polygons(68, 20)
via = polygons(68, 44)

gate = diff.and(poly)
```

We aimed to identify the slowest part in and operation code, using real world design input, caravel.gds.gz from skywater 130nm openPDK, above and operation takes 30 seconds to finish(./drc.sh to reproduce). By profiling above DRC code with perf and then visualize the callstack with flameGraph and considerable amount of time invested to trace how each ruby script code mapped into C++ coding using overrides/delegate pattern/templates. We managed to find followed C++ code section Despite the promise of this approach, we encountered challenges, and failed to ensure data safety while adding openMP pragma into followed function code, which is code segment where "and" operation spent most of the runtime:

Listing 2: dbHierProcessor.cc

```
template <class TS, class TI, class TR> void
    local_processor<TS, TI, TR>::compute_
    local_cell (const db::local_processor_
    contexts<TS, TI, TR> &contexts, db::Cell
     *subject_cell, const db::Cell *intruder
    _cell, const local_operation<TS, TI, TR>
     *op, const typename local_processor_
    cell_contexts<TS, TI, TR>::context_key_
    type &intruders, std::vector<std::
    unordered_set<TR> > &result) const {/*
    ...*/}
```

Also, KLayout seems to be using certain parallelism techniques like Pthreads and tiliing with scanline algorithm, which was not noticed at the project proposal stage.

As a result, we turned to Taskflow, a parallel and heterogeneous programming framework that provides a more manageable approach to parallelizing DRC scripts.

3 EXPERIMENTAL METHODOLOGY

Our experimental approach involved executing DRC scripts on various integrated circuit designs, measuring the execution time for different parts of the script, and identifying potential bottlenecks. We employed a multicore system to allow for parallel execution of scripts, and used diverse input sets to ensure the robustness of our solution. This rigorous methodology helped us gauge the effectiveness of our proposed solution and led us to Taskflow as the more efficient framework.

4 EXPERIMENTAL RESULTS

Our initial attempts at parallelization using OpenMP encountered significant difficulties. These were primarily related to side effects and code complexity which hindered the effective parallelization of the 'compute_local_cell' function. However, the transition to Taskflow resulted in notable performance improvements, showcasing its superiority in managing task-level parallelism. We achieved a considerable speedup in the execution of DRC scripts and observed better scalability with the number of processor cores.

5 RELATED WORK

Past research efforts have sought to improve the performance of design rule checking in integrated circuit designs by proposing algorithmic improvements and hardware acceleration. However, these solutions have largely overlooked the potential of parallel execution. Our work, therefore, presents a significant step forward in this area by demonstrating the successful parallelization of DRC scripts using Taskflow.

6 CONCLUSIONS

Our exploration of parallel computing in KLayout's DRC processes revealed important insights. The initial challenges encountered with OpenMP underscored the importance of selecting the appropriate parallel computing tool for specific use cases. Meanwhile, our successful adoption of Taskflow provided tangible performance improvements, emphasizing its potential for managing task-level parallelism. Our work sets the stage for further research on the parallelization of integrated circuit design and testing processes.

7 REFERENCES

- KLayout Documentation, "KLayout User Manual," [Online]. Available: https://www.klayout.de/doc/index.html [Accessed: 04-April-2023].
- KLayout Repository, "KLayout Source Code," [Online]. Available: https://github.com/KLayout/klayout [Accessed: 04-April-2023].
- Taskflow, "Taskflow: A General-purpose Parallel and Heterogeneous Task Programming System," [Online]. Available: https://taskflow.github.io/ [Accessed: 04-April-2023].