Parallel Physical Design Automation System

Partitioning and Floorplanning

楊辰彬
312551030
資科工碩一
ycpin.cs12@nycu.edu.tw

劉承熙
311552067
網工碩一
brianliu.cs11@nycu.edu.tw

蔡政邦
312551129
資科工碩一
tsai.cs12@nycu.edu.tw

ABSTRACT

Physical Design Automation (PDA) is a crucial step in the Electronic Design Automation (EDA) process. As manufacturing processes advance, the number of transistors on a single chip increases accordingly. Utilizing traditional physical design automation workflows can be extremely time-consuming. Therefore, we have developed the Parallel Physical Design Automation System, which encompasses PDA's Partitioning and Floorplanning stages. We implemented this system using OpenMP and Pthread. The results showed that under 16 threads, our system is 12 times faster compared to the serial approach. This report will provide a detailed explanation and analysis of this system.

KEYWORDS

Parallel System, Physical Design Automation, Partitioning, Floorplanning, OpenMP, Pthread

1 Introduction

Modern electronic design constantly strives for higher performance, lower power consumption, and faster time-to-market, presenting significant challenges in handling Very Large-Scale Integration (VLSI) designs. Physical design, as the backend of VLSI Design, aims to progress the designed RTL into the following stages of circuit design, as illustrated in Figure 1.

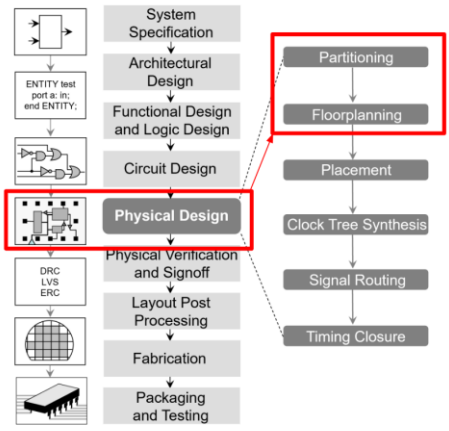

Figure 1: Very Large-Scale Integration (VLSI) design flow

In traditional physical design automation workflows, validating and optimizing VLSI circuits often require substantial time and computational resources. To address this challenge, we have developed the Parallel Physical Design Automation System, which aims to accelerate design validation, improve design quality, and shorten the time-to-market.

Our Parallel Physical Design Automation System primarily includes two stages: Partitioning and Floorplanning.

1.1 Partitioning

The purpose of partitioning is the decomposition of a complex system into smaller subsystems. With the advancement in chip development, the number of logic gates and modules on a single chip has increased, making it challenging to load entirely a single chip into an FPGA (Field Programmable Gate Array) for validation. Therefore, we need partitioning techniques to divide a complex chip into multiple sub-chips, which can then be individually loaded into FPGAs for parallel simulation. As shown in Figure 2, an FPGA Cluster is a device used for simulating complex chips.

Figure 2: FPGA Cluster

Then, we must consider dividing the circuit using an efficient partitioning algorithm. The primary goal is to minimize the connections between different circuit partitions. This approach reduces the dependency between partitions and reduces communication and data transfer costs. We have chosen the Fiduccia–Mattheyses Algorithm [1], a linear-time heuristic for improving network partitions, as our main algorithm for the

partitioning stage. Using a bisection method allows us to partition a complex system in parallel, as depicted in Figure 3.

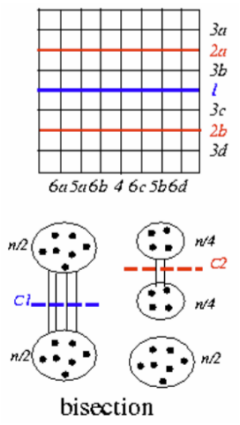

Figure 3: bisection method

1.2 Floorplanning

Floorplanning aims to provide chip designers with early feedback that evaluates architectural decisions, such as estimating chip areas. Given the widespread use of Hierarchical design and Intellectual Property (IP) modules in modern chip design, floorplanning can be used to estimate the chip area preliminarily. We have chosen to combine the B* Tree [2] with Simulated Annealing [3]. The B* Tree is an efficient and effective data structure for floorplan design, and Simulated Annealing is the most popular method for floorplan optimization. An illustrative diagram of floorplanning is shown in Figure 4.

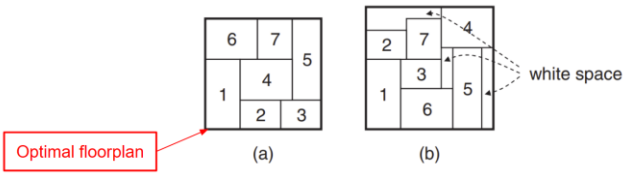

Figure 4: an illustrative diagram of floorplanning

2 Motivation

We observe that the multiple partitions generated in the partitioning stage can be processed in parallel during the subsequent partitioning stage [4]. Furthermore, the final partitions can be parallelly processed in the floorplanning stage to generate the optimal floorplan. Leveraging this concept, we designed the Parallel Physical Design Automation System, thereby enhancing the speed of both partitioning and floorplanning. As illustrated in Figure 5, we divide a complex circuit and then generate the floorplans for these circuit partitions.

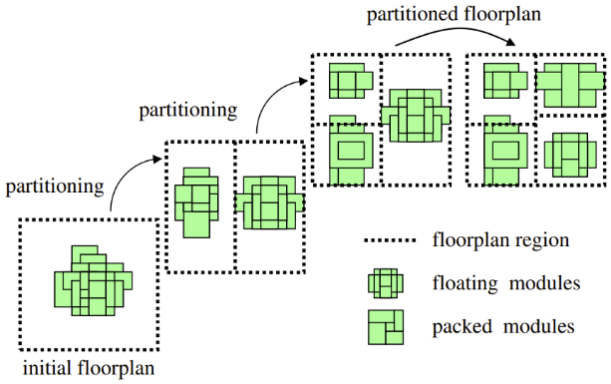

Figure 5: the flow chart of partitioning and floorplanning

3 Contribution

Our 'Parallel Physical Design Automation System' research project offers the following contributions:

1. We develop a parallelized physical design automation workflow that utilizes multicore processors and distributed computing environments to accelerate the physical design process.
2. The parallelized system achieves a significant speedup compared to the serial version, substantially reducing the time required for the design process.
3. We analyze and rationally explain the performance with different threads and benchmarks.
4. We also analyze the speedup in both process stages, along with suggestions for future improvements.

In summary, the primary contribution of this research project is the development of a highly parallelized physical design automation workflow. This workflow is designed to tackle the challenges of Very Large-Scale Integration (VLSI) circuit design, thereby enhancing design efficiency.

4 Proposed Solution

Combining Partitioning and Floorplanning involves a novel approach. Traditional Partitioning processes slice the integrated circuit step by step in sequence, whereas parallelized partitioning implies considering multiple sub-circuit partitions simultaneously instead of handling them one after the other. Subsequently, these partitions are processed in parallel during the floorplanning phase. Figure 6 illustrates that the input data consists of a complex circuit with multiple modules during the partitioning stage. The Parallel Physical Design Automation System allocates this data to Thread 0 for partitioning, resulting in two partitions. These two partitions are then allocated to Thread 0 and Thread 1 for further partitioning, and so on.

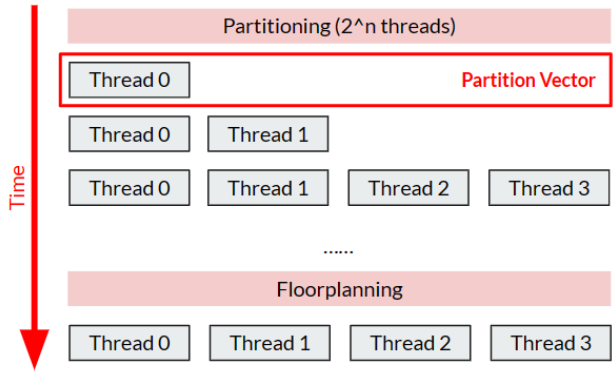

Figure 6: the workflow diagram of our Parallel Physical Design Automation System

This process continues until the sum of the areas of the modules in each partition is not less than a set Bound, marking the end of the partitioning stage.

Bound = total area of all modules in the circuit / p_bound (1)

Then, the Floorplanning stage is carried out. In this stage, all the partitions are independently assigned to each thread, and floorplan layouts are generated in parallel. This stage, compared to the partitioning stage, should achieve better parallelization. The results are shown in Figure 7, where each partition produces a floorplan. The first line represents the area of that floorplan, and the second line represents the aspect ratio of this floorplan. Subsequent outputs represent the numbers of each module within the partition and their coordinates at the bottom-left corner. A schematic illustration of the floorplan is shown in the figure.

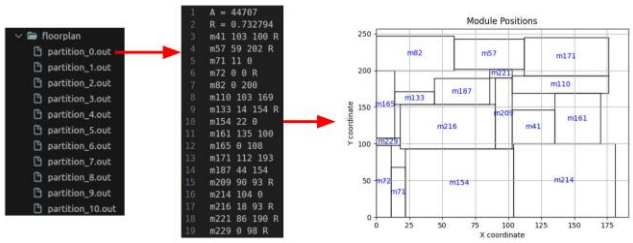

Figure 7: the floorplanning results of our Parallel Physical Design Automation System

5 Experimental Methodology

The Algorithm and its Parameters we adopted are as follows:

- Partitioning method - Fiduccia–Mattheyses Algorithm (epoch = 100).
- Floorplanning method - B* Tree + Simulated Annealing (alpha = 0.9, T_min = 0.0001, T_epoch = 10000).

The input data and parameters used for the experiment are as follows:

- Utilized 4, 8, 16, and 32 threads and conducted experiments on an AMD Ryzen Threadripper 3970X 32-core, 64-thread platform.
- The input netlist is shown in Figure 8, where the first line represents the balance factor r for partitioning, limiting the difference in the number of modules between two partitions during partitioning to not exceed 1/r of the larger partition. The rest represents each netlist's connected modules, with the number of modules in each netlist ranging from 2 to 20.

```
0.5
NET n0 m293 m136 m138 m298 m12 m237 m211 m148 m189 m247 m59 m285 ;
NET n1 m256 m129 m99 m228 m165 m37 m36 m72 m4 m110 m50 m82 m54 m23 m88 m153 m60 m24 ;
NET n2 m0 m256 m131 m298 m235 m279 m81 m274 m243 m30 m124 m222 ;
```

Figure 8: the input data of netlist

- The input module list is shown in Figure 9, where the first line represents the aspect ratio range limits for the floorplan, and the rest represents each module's number and its length and width, with both dimensions of the module being less than 100.

```
0.6 1.75
m0 75 94
m1 24 52
m2 97 97
```

Figure 9: the input data of module list

Our tested benchmark comprises three parts: <Module, Netlist, p_bound>, representing the number of modules, the number of netlists, and the p_bound used to control the number of partitions, respectively. The benchmark for the experiment is - <300, 500, 10>, <500, 1000, 20>, <1000, 1500, 100>.

5 Experimental Results

5.1 Experiment 1

<Module, Netlist, p_bound> = <300, 500, 10>. The experiment's results, as shown in Figure 10, indicate that approximately 35 partitions were generated in this experiment. At this point, the speedups of Pthread and OpenMP are similar.

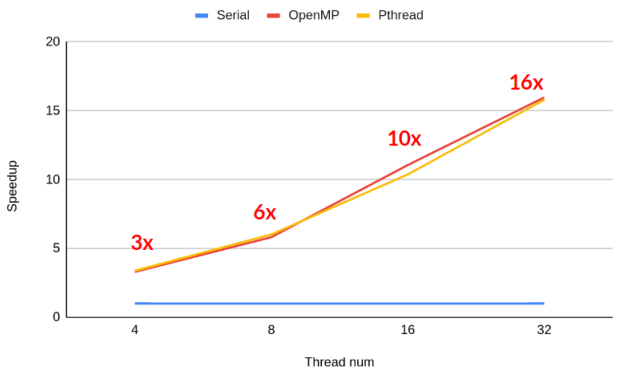

Figure 10: the result of experiment 1

5.2 Experiment 2

<Module, Netlist, p_bound> = <500, 1000, 20>. The experiment's results, as shown in Figure 11, indicate that approximately 50 partitions were generated in this experiment. At this point, the speedup of Pthread is slightly higher than that of OpenMP.

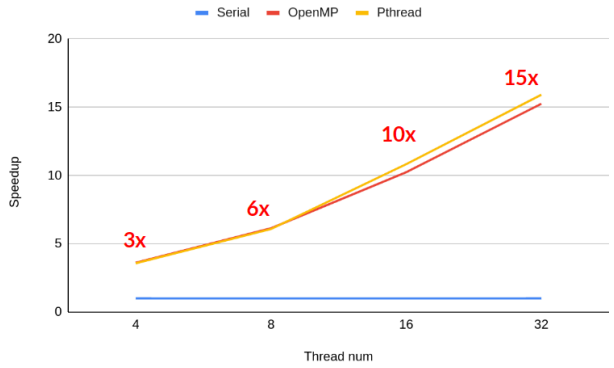

Figure 11: the result of experiment 2

5.3 Experiment 3

<Module, Netlist, p_bound> = <1000, 1500, 100>. The experiment's results, as shown in Figure 12, indicate that approximately 150 partitions were generated in this experiment. At this point, the speedup of Pthread is significantly higher than that of OpenMP.

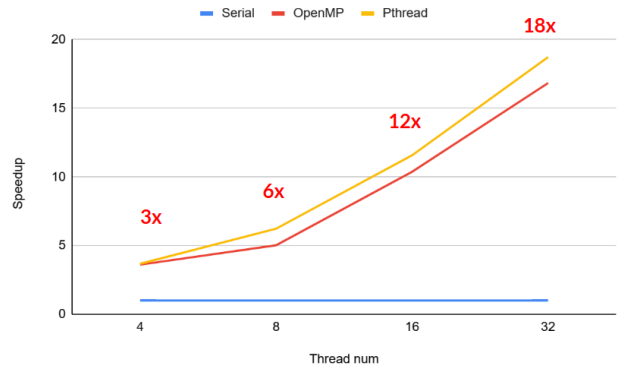

Figure 12: the result of experiment 3

Summarizing the above three experiments, it objectively shows that the more partitions there are in the system, the higher the speedup of Pthread compared to OpenMP. This can be inferred due to the more even workload distribution of Pthread and the use of critical sections in OpenMP, which leads to lower performance.

Next, we conducted experiments separately on the Partitioning and Floorplanning stages to consider how to achieve better performance.

5.4 Experiment - Partitioning Stage

<Module, Netlist, p_bound> = <1000, 1500, 100>. The results of the experiment, as shown in Figure 13, show that the speedup of the Partitioning Stage did not increase with the addition of threads. This is because the Partitioning stage did not use all the threads at the beginning and the end.

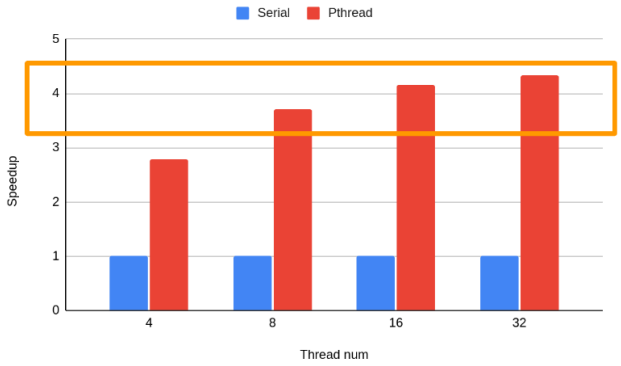

Figure 13: the result of partitioning stage experiment

5.5 Experiment - Floorplanning Stage

<Module, Netlist, p_bound> = <1000, 1500, 100>. The experiment's results, as shown in Figure 14, show that the speedup of the Floorplanning Stage significantly increased with the addition of threads.

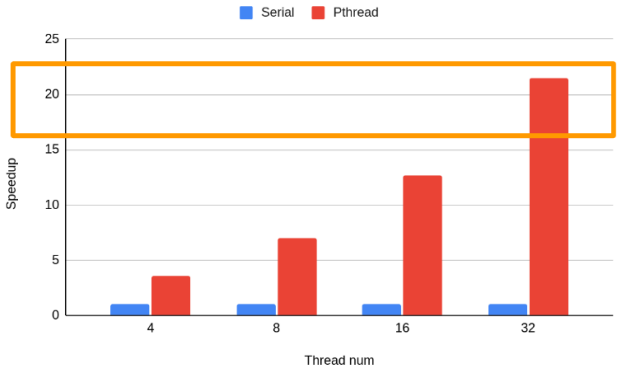

Figure 14: the result of floorplanning stage experiment

Although the performance of the Partitioning Stage could be better, it does not significantly drag down the overall system speedup. The reason is that partitioning only does 10~15% of the work; hence, the acceleration benefit of the Floorplanning Stage is still very prominent.

6 Conclusion

Our Parallel Physical Design Automation System accelerates the Partitioning and Floorplanning stages in PDA (Physical Design Automation). Through experimentation, it was found that as the total number of partitions in the system increases, the speedup of Pthread is better compared to OpenMP. Moreover, the speedup magnitude of the Partitioning stage is smaller compared to the Floorplanning stage because the Floorplanning stage has a more significant workload in the entire system, and the acceleration effect is more pronounced post-parallelization. Our research project is focused, demonstrates excellent acceleration effects, and

includes comprehensive experimentation and reasonable explanations. In the future, we plan to optimize this research project further and truly address issues related to the physical design phase of very large-scale integrated circuit chips.

7 Related Work

Hardware Acceleration of EDA Algorithms, published by Kanupriya Gulati and Sunil P. Khatri in 2010.

(https://link.springer.com/book/10.1007/978-1-4419-0944-2)

Provides guidelines on whether to use GPUs or FPGAs when accelerating a given EDA algorithm, with validation by a concrete example implemented on both platforms.

Demonstrates the acceleration of several popular EDA algorithms on GPUs, with speedups from 30X to 800X presents techniques in a way that the reader can use example algorithms presented to determine how best to accelerate their specific EDA algorithm.

REFERENCES

[1] C.M. Fiduccia and R.M. Mattheyses (1982). *A Linear-Time Heuristic for Improving Network Partitions.* 19th Design Automation Conference. https://ieeexplore.ieee.org/document/1585498

[2] Chang, Y.-C., Chang, Y.-W., Wu, G.-M., and Wu, S.-W. (2000). *B*-trees.* Proceedings of the 37th Conference on Design Automation; - DAC '00. https://dl.acm.org/doi/pdf/10.1145/337292.337541

[3] Chen, T.-C., and Chang, Y.-W. (2005). *Modern floorplanning based on fast simulated annealing.* Proceedings of the 2005 International Symposium on Physical Design. https://dl.acm.org/doi/abs/10.1145/1055137.1055161

[4] Tung-Chieh Chen, Yao-Wen Chang, and Shyh-Chang Lin. (2005). *IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs.* ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005. http://cc.ee.ntu.edu.tw/~ywchang/Papers/iccad05-imf.pdf