PARTITIONING IN THE PARROT FLOW FOR PHYSICAL SYNTHESIS

Samuel Nascimento Pagliarini, Marcelo de Oliveira Johann, Ricardo Reis

Universidade Federal do Rio Grande do Sul

ABSTRACT

This paper presents an ongoing work which comprises the insertion of a Partitioning step in the PARROT Flow for Physical Synthesis. A partitioning step was inserted to enclose a simulated annealing based-placement step of the flow. Runtime and wire length data were collected to compare the results within and without the partitioning step on a group of circuits. Also, using these same circuits, we have collected area and wire length to analyze the cost of the partitioning step versus the benefits in runtime and routability. Besides that, the different ways to create the partitions and manage the connections between them will be discussed.

1. INTRODUCTION

The reason we started to implement this partitioning tool was because of the existence of a complete circuit design flow, including placement, layout generation and routing, developed at our university. This flow is called PARROT Flow [1] [2]. However, currently this flow is extremely time demanding because of the simulated annealing (S.A.) approach in the placement tool, called MangoParrot [3]. Because of its characteristics, the placement tool requires some tuning in order to perform correctly. The most important parameter to set is the *num_reps_per_cell*, which controls the number of iterations in the inner loop of the S.A. technique. In a previous study it was determined that this parameter should be set as the number of cells to the second power, so the circuit could have an acceptable wire length/runtime trade off.

When the number of cells in the circuits starts to increase, the demanding time to place them by a S.A. approach becomes way too large. Although an analytical placement tool may be used to reduce the runtime, simulated annealing based techniques are probably the best ones when it comes to wire length reduction[4]. This is why a partitioning tool was implemented: by splitting the circuit into multiple parts, it is possible to keep the number of iterations under control so that the placement tool runtime is considerably decreased, therefore, diminishing the placement runtime problem.

Some problems arise when we must decide how to split the circuit and, after placing each partition, we must bind then together again in a certain way that the connectivity of the circuit is optimized. These problems will be presented in the next sections when the virtual pin representation will be introduced.

Section two presents the current state of this work. In section three some preliminary results are shown and discussed. And finally, the conclusions and future work are hold in section four.

2. CURRENT STATE OF IMPLEMENTATION

When measuring the run time improvement achieved through partitioning it really does not matter how the partitioning is done, only the number of cells in the partition being placed matter. However, that is not true for placement (measured in wire length) and routing (measured in unrouted nets) since it is possible to mess with all the connectivity of the circuit by making wrong decisions on which cell belong to which partition. The main idea is that a group of cells that are strongly connected should be put in the same partition, in a certain way that the connections between partitions are minimized and the partitioned circuit characteristics are kept close to the ones in the original circuit.

In graph theory, a cut is a partition of the vertices of a graph into two sets. The size of a cut is the total number of edges connecting two partitions, and a cut is minimal if the size of the cut is not larger than the size of any other cut. The idea of finding a minimal cut in a graph comprises with our idea of strongly connected cells.

Once the problem is identified we can make use of a third party tool to create the partitions. Finding the minimal cut in a graph is the approach of several partitioning tools available. The chosen tool is called hmetis [5] and it uses algorithms based on multilevel hypergraph partitioning schemes [6]. In order to minimize the placement run time, the partitioning run time must also be minimized, so the partitioner must split large graphs very fast. In the circuits that we performed experiments and collected data, this time is, at most, only 0,00002% of the total placement time without partitioning.

The hmetis tool must be supplied with a file containing a weighted graph description that represents the input circuit for the hmetis partitioner. The developed tool, called **weezer**, is responsible for creating this file and the partitioning/placement management. For later comparison with other tools and simplicity at the moment, it is desired that each partition have the same circuit area or at least close to that. To reach this requirement we also provide hmetis with a parameter defining the maximum acceptable difference between partitions areas. This parameter, called the unbalance factor, is currently set as 1%, the lowest possible value. Since weezer itself can not estimate each cell area, it uses the results from another external tool called CellSE [7] to estimate the areas and later put these values in the hmetis graph description file as weights.

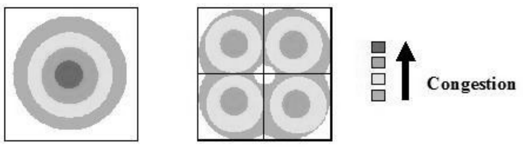

Fig. 1 – Congestion analysis

Using a last parameter which specifies the desired number of partitions we are ready to start the partitioning and weezer immediately makes a system call and runs hmetis. After the partitioning is done hmetis writes a file where each cell from the original circuit is given a partition number in which it belongs. Weezer reads that file and creates a new circuit description file (CDF) for each partition, corresponding to different placement inputs. These new files contain cells, nets, regular pins and virtual pins.

The nets that connect two or more cells from the same partition are written to the corresponding partition CDF. The regular pins that are used by at least one cell from the partition are also written. The virtual pins are created when a net that connects two or more cells from different partitions is found. The virtual pin representation is used to maintain the partitions connectivity.

It is important to notice that the number of virtual pins is equal to the minimal cut (or double if you count it once in each partition) only if we split the circuit in two partitions. When we split a circuit in more than two partitions the sum of the virtual pins is equal to the hyperedge cut(sum of minimal cuts from all internal bisections that hmetis made).

Also on virtual pins: they must be placed properly. To do this we applied a very simple rule in which a virtual pin position will be the closest possible to any other partition that contains the same net. This is clearly not optimal, but still considerably reasonable and fast to implement and run. It is reasonable because once close to one of the other possible partitions, in the worst case it will be just shifted inside the bounding box formed by this chosen partition and the possible other ones.

Before we do the placements we must estimate the circuit area and since all placements are row based, this means we must estimate how many rows each partition will occupy. The number of rows is the way we control the circuit aspect ratio. Currently we are aiming to keep the circuits with the width the same as the height, in a squared shape. If necessary or desired this shape can be easily changed. This feature will also be useful if a future floorplanning strategy provides a desired area for the weezer tool. To estimate the number of rows we roughly add up the areas from all the cells and divide it by the strip height, which will give us an average slightly bigger width, in comparison with the height.

At this stage weezer is ready to perform the placement of all individual partitions. This is done by calling the MangoParrot placer for each one.

Since MangoParrot is a wire length driven placer all the circuits placed with it looks like the left side of fig. 1,

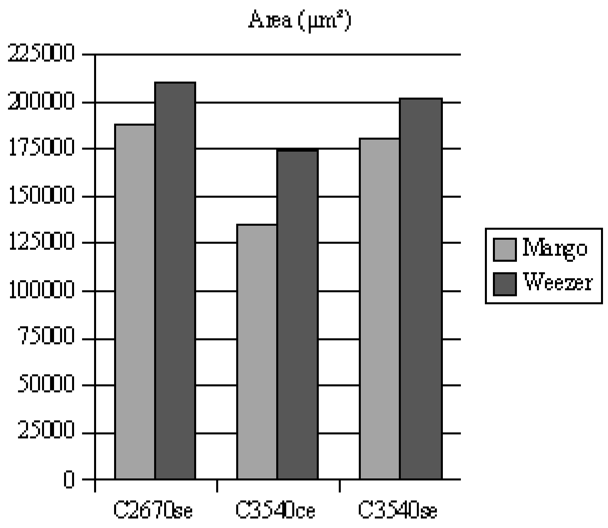

Fig. 2 – Area analysis.

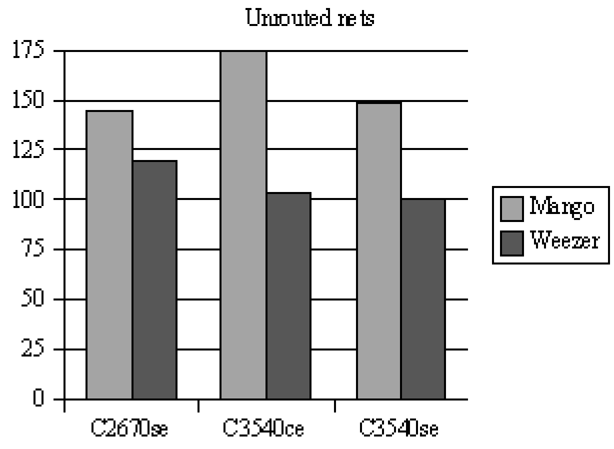

Fig. 3 – Unrouted nets analysis.

where the center is heavy populated. When weezer is used the circuit will look like the right side of fig. 1, where the congestion is more spread among the hole circuit area. The image shows that the rounded shapes may create some areas without cells, which could mess up the entire placement/partitioning process since our flow deals with relative positioning. So, when weezer is binding the partitions it detects these blank spaces and inserts floating inverters to maintain the circuit shape. These floating inverters act as filler cells and help the circuit to be more routable. This result and others are shown in the next section.

3. PRELIMINARY RESULTS

In order to observe the impact of the partitioning step in the placement runtime reduction we performed several placements using a variety of circuits, mostly obtained from ISCAS98 benchmarks. The results are shown in fig. 4. The weezer results were obtained using a 2x2 partitioning grid. In the *X* axis we have the number of cells. In the Y axis we have the time to place.

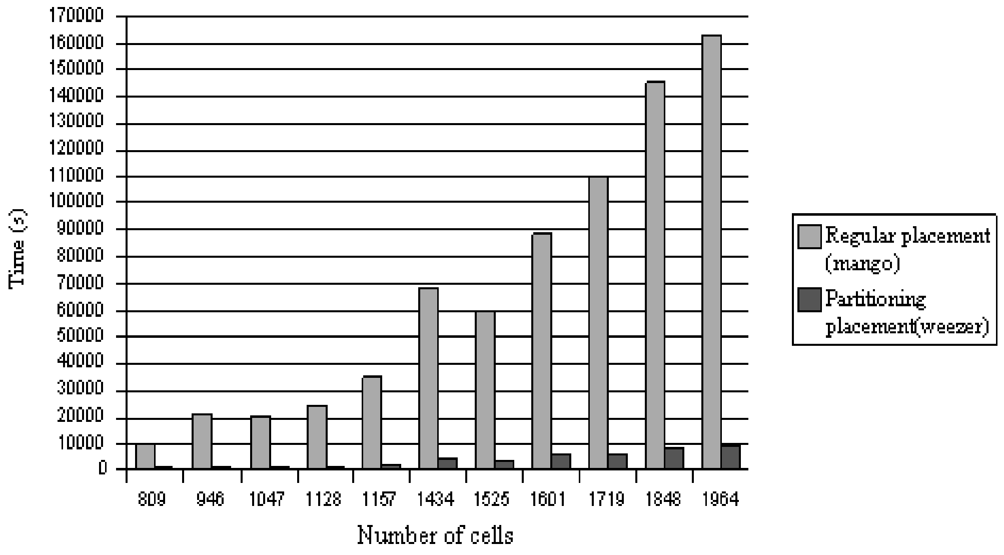

Fig. 4 – Mango versus weezer.

The values in X axis are not in any kind of scale since that number is obtained from real existing circuits. Because of that the behavior of the regular placement curve is a bit twisted but it is close to quadratic. The same way is the weezer placement curve, in which the actual behavior is close to linear. If we aim to change the curve behavior into really linear we can define a maximum number of cells allowed per partition. Let us say that the chosen number is 200 cells and it takes 250 seconds to place this number of cells. So, all placements maximum runtime can be calculated by ((*N* div 200)+1)*250 seconds, where *N* is the total number of cells for a particular circuit.

One important result that is not shown in fig. 4 is the whole partitioning management runtime, that is the sum of CellSE, hmetis and weezer itself runtimes. For the circuits placed in fig. 4 this time ranges from 0,99s to 1,92s, representing less than 0,001% of the sum of the placement time for all four partitions. These measurements and all the placements were made using an single core ATHLON XP 2000+.

Since a partitioning step is inserted in the flow we must understand how that will affect the final circuit layout and characteristics. In order to do that several circuits were submitted to the weezer tool. Some of the results obtained are showed in fig. 3 and fig. 4.

In fig. 3 we have the total circuit area for three different circuits. As expected the partitioning caused a little area overhead. On the other hand, fig. 4 shows that the number of unrouted nets is smaller. Altough the number of unrouted nets is still unacceptable, it is clear that the congestion distribution leads to an easier routing.

4. CONCLUSIONS AND FUTURE WORK

The main conclusion at the moment is that partitioning allows simulated annealing based placements to run in reasonable time. Since simulated annealing strategies may achieve a wire length very close to optimal, we must now find out if partitioning does not affect that condition. This may be achieved by comparing the results against an analytical based placer. However, even a wire length increase, due to congestion distribution, can improve the flow convergence by improving routability. Also, without the partitioning the current flow was not able to benefit from the use of parallelization. Now several partitions placements can be done in parallel since each partition placement is independent from each other.

It is interesting to observe that the placement itself does not require the creation of virtual pins but its final result may be improved if they are properly placed in each block boundary, especially for wire length reduction. In another words, using virtual pins is an affordable way to decrease the additional wire length without adding more complexity to the simulated annealing approach.

As mentioned before the virtual pin positioning is not optimal: that algorithm can be refined to match the optimal direction of the virtual pins, even in the cases when nets have terminals in more than two partitions. Also, the floating inverters could become real filler cells and a more detailed aspect ratio/floorplanning strategy could be developed.

These ideas are either under development or testing, so the results at the moment are not fully conclusive. This paper is mostly based on ongoing research although we strongly believe that all ideas deserve a closer look since all of then look very promising.

4. REFERENCES

[1] C. Lazzari, C. Domingues, C.J. Güntzel, and R. Reis, “A New Macrocell Generation Strategy for Three Metal Layer CMOS Technologies”, *VLSI-SoC*, Germany, 2003.

[2] C. Lazzari, “Parrot Punch”, http://www.inf.ufrgs.br/~clazz/parrotpunch [viewed 24/03/2008].

[3] R.F. Hentschke, “Algoritmos para o Posicionamento de Células em Circuitos VLSI”, *Master dissertation*, UFRGS, 2002.

[4] S.H. Gerez, “Algorithms for VLSI Design Automation”, Chichester: John Wiley, 1999.

[5] G. Karypsis, and V. Kumar, “Multilevel k-way Hypergraph Partitioning”, *VLSI Design*, Vol. 11, No. 3, pp. 285 - 300, 2000.

[6] G. Karypsis, and V. Kumar, “METIS - Family of Multilevel Partitioning Algorithms”, http://glaros.dtc.umn.edu/gkhome/views/metis [viewed 24/03/2008].

[7] A.M. Ziesemer, G.B.V Santos, R.F. Hentschke, and R.A.L. Reis. “Cell Size Estimation in an Automatic Layout Generation Flow”. *21th South Simposium on Microelectronics*, Porto Alegre, 2006.