Hierarchical Global Floorplacement Using Simulated Annealing and Network Flow Area Migration

Wonjoon Choi Kia Bazargan
ECE Department, University of Minnesota, MN 55455
{wjchoi,kia}@ece.umn.edu

Abstract – *Floorplanning large designs with many hard macros and IP blocks of various sizes is becoming an increasingly important and challenging problem. This paper presents a global floorplacement method that combines a hierarchical simulated annealing floorplanning method with a partitioning-based global placement technique. A novel area migration method formulated as a min-cost, max-flow network flow problem is used to improve area utilization, and provide a communication mechanism between the partitioning engine and the placement method for better design quality. The network flow area migration method can be used in managing incremental changes in the design as well. Our global placement wire length is 12% better than the detailed placement wire length of a previous work, while our global placement is almost 8 times faster than their global placement.*

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Placement and routing.

General Terms
Algorithms, Management, Design.

Keywords
Floorplanning, floorplacement, global placement, hierarchical, network flow, area migration, simulated annealing.

1. Introduction
The size and complexity of future designs have made it imperative to employ hierarchical physical design methods. IP reuse is a definite component of future CAD tools. There will be large numbers of pre-defined IP blocks of various sizes that have to be integrated into the system. Floorplanning of these blocks will be a challenge. Furthermore, incremental changes should be handled by the tools to allow for any possible area / delay budget violations of different teams, as well as any changes resulting from debugging or ECOs.

2. Previous Work
There have traditionally been two classes of floorplanning and placement methods: iterative (usually simulated annealing), and partitioning-based. Iterative methods are generally slow but they can efficiently handle the module shape management problem. On the other hand, partitioning-based placement shows good scalability but cannot handle the sizing problem well. Recently, a placement method was developed that handles large blocks by modeling them as tightly coupled clusters of smaller blocks, forcing the placement tool to keep them together [1].

3. Partitioning-based Floorplacement
Our proposed floorplanning and placement method (see Figure 1) combines the quality of annealing-based and scalability of partitioning-based floorplanning and placement methods. We call our floorplanning and placement a "floorplacement" method, as it starts with floorplanning large modules and gradually deals with smaller modules while removing possible area overlaps using a novel network flow formulation.

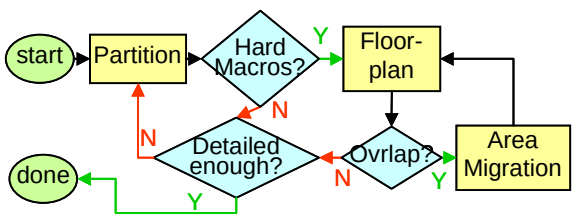

Figure 1 Our flow

In our method, partitioning keeps the number of modules small (hence a runtime controlled annealing), and floorplacement optimizes metrics such as area, timing and congestion. Employing a polynomial time legalization step (called area migration in Figure 1) at each level of the hierarchy allows us to keep the floorplacement legal without spending too much time in the annealing process.

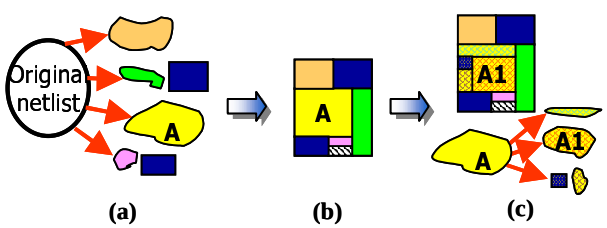

Figure 2 An example of our flow. (a) The original netlist is partitioned into four soft modules and two "large" hard macros. (b) Floorplan generated. (c) Soft module A is further partitioned into "large" macros and modules.

Figure 2 shows an example of how our method works. We start with a netlist and a fixed floorplan area. At each level of the partitioning, "large" hard macros[1] and IP blocks are extracted from the netlist and the rest of the gates and small hard macros and IP blocks are partitioned (clustered) into a number of "soft" modules using the

[1] A module whose side is larger than 1/K of the sub-floorplan's width/height (e.g., K=8).

hMetis partitioning tool [3]. The mixed hard/soft modules are floorplanned using the Wong-Liu slicing floorplanning method. The same process is recursively repeated on the "soft" modules.

4. Area Migration by Network Flow

To achieve better area utilization, we allow soft modules to be broken into smaller sub-modules and moved to empty regions on the chip, or even merged with other modules. Depending on user preferences and timing criticalities of the soft modules, the sub-modules might be placed at disconnected regions on the chip.

We model the area migration problem as an instance of weighted network-flow problem [2]. Figure 3 shows a very simple example of the network flow formulation. Area can migrate from "spatially critical" modules (e.g., "B", "C") to empty r egions or other modules. The dashed rectangle is the area allocated to the floorplan. The edge capacities are not shown, but they determine how much area can migrate along the edge. We also define edge costs as the amount of increase in timing or wirelength if area migrates using the edge.

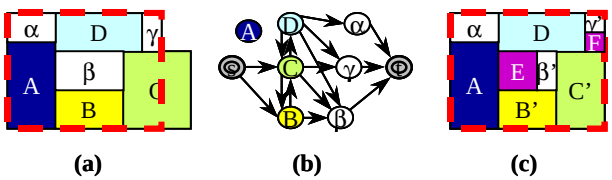

Figure 3 Area migration formulation. a) floorplan. b) network graph. c) floorplan after area migration.

When some area is taken from a soft module to migrate to another location, some internal nets become external, i.e., connect cells from different modules in the floorplan. We use Rent's rule to estimate the number of wires "exposed".

5. Experimental Results

We performed global floorplacement (recursively perform floorplacement until the modules contain less than 30 gates) on the ISPD98 IBM circuits and compared our results to those from [1]. Runtimes are measured on a 900 MHz PC/Intel system running Linux. Wirelength is measured using half-perimeter of the minimum bounding box containing all terminals of a net (HPWL). The amount of overlap in a final floorplan is measured by computing the sum of overlaps between each pair of modules, divided by the whole chip area. The floorplanning area is set to have a square shape with 15% of white space.

Table 1 shows that our global placement HPWL is 12% better than the detailed placement HPWL of [1] (they do not report their global placement HPWL), while our global placement is almost 8 times faster than their global placement ("ShredPlace by Capo" + "Floorplan time by Parquet" in [1]).

6. Conclusion

We showed our global floorplacement method can produce reasonably good results very quickly. The method is scalable and provides quality / runtime tradeoff by balancing the annealing runtime at each hierarchy level and the effort put into the area migration method. The area migration method can improve the floorplan and provide a means for incremental design.

7. References

[1] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-Blocks Using Floorplanning and Standard-Cell Placement", *International Symposium on Physical Design (ISPD)*, pp. 12 – 18, 2002.

[2] A. V. Goldberg, "An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm", *Journal of Algorithms*, 22, pp. 1--29, 1997.

[3] G. Karypis and V. Kumar. "Multilevel k-way Hyper-graph Partitioning". *Design Automation Conference (DAC)*, pp. 343-348, 1999.

			Capo + Parquet + Capo		Our Floorplanning Flow				Comparison	
circuit	#nodes	#nets	HPWL	runtime(min)	#Hmacros	HPWL	runtime (min)	Overlap	HPWL	speedup
ibm01	12752	14111	3.96E+06	17	3	2.78E+06	2	6.31%	0.70	8.5
ibm02	19601	19584	8.37E+06	28	12	6.64E+06	3	5.52%	0.79	8.0
ibm03	23136	27401	1.22E+07	38	10	1.04E+07	5	7.62%	0.86	7.0
ibm04	27507	31970	1.35E+07	42	8	1.17E+07	6	7.85%	0.87	7.0
ibm05	29347	28446	1.15E+07	8	0	1.31E+07	5	1.72%	1.13	1.6
ibm06	32498	34826	1.03E+07	50	12	9.89E+06	9	5.46%	0.96	5.6
ibm07	45926	48117	1.58E+07	49	11	1.62E+07	14	7.04%	1.03	3.5
ibm08	51309	50513	2.12E+07	84	15	1.80E+07	15	8.06%	0.85	5.6
ibm09	53395	60902	1.96E+07	53	13	2.02E+07	16	7.91%	1.03	3.3
ibm10	69429	75196	6.07E+07	215	8	4.23E+07	27	4.29%	0.70	8.0
ibm11	70558	81454	2.85E+07	90	14	2.76E+07	23	9.69%	0.97	3.9
ibm12	71076	77240	5.17E+07	659	15	4.80E+07	27	7.06%	0.93	24.4
ibm13	84199	99666	3.94E+07	131	12	3.54E+07	33	10.11%	0.90	4.0
Avg			**2.28E+07**	**112.62**		**2.02E+07**	**14.23**	**6.82%**	**0.88**	**7.91**

Table 1 Comparison between our method and [1]