Timing Minimization by Statistical Timing hMetis-based Partitioning

Cristinel Ababei **Kia Bazargan**

Department of Electrical and Computer Engineering
University of Minnesota, 200 Union St. SE
Minneapolis, MN 55455
{ababei,kia}@ece.umn.edu

Abstract

In this paper we present statistical timing driven hMetis-based partitioning. We approach timing driven partitioning from a different perspective: we use the statistical timing criticality concept to change the partitioning process itself. We exploit the hyperedge coarsening scheme of the hMetis partitioner for our timing minimization purpose. This allows us to perform partitioning such that the most critical nets in the circuit are not cut and therefore timing minimization can be achieved. The use of the hMetis partitioning algorithm makes our partitioning methodology fast. Simulations results show that 22% average delay improvement can be obtained. Furthermore, as a result of using the statistical timing model, the partitioning results can tolerate changes in temperature and process variation, hence causing less delay change compared to partitioning using static timing models.

1. Introduction

The increase of circuit complexity and the high demand for short time-to-market products force designers to reuse old designs (IP) while the increased chip densities allow them to put more and more functionality on the same chip (SoC). Therefore, it is envisioned that platform-based design will be the optimal design approach [17]. Within this new framework, where Field Programmable Logic Arrays (FPGAs) and embedded systems define the hierarchical design philosophy, we can no longer rely on flat design methodologies. Such methods are too time-consuming for today's large designs. Hierarchical design, where systems are built from pre-characterized library blocks/functions (such as multipliers, filters, or even processors) appears to be the proper approach. That is why our proposed solution in this paper targets circuits of moderate- and large-sized circuits, basic components of libraries that we mentioned above. For these circuits we have to perform gate-level physical design. Therefore, partitioning, as an early step during physical design, remains very important.

On the other hand, since interconnections/wiring contribute more than 70% to the circuit/block delay, partitioning has a great impact on the interconnect distribution and thus on the circuit performance. Therefore, it is imperative to account for timing as early as possible during the design process, particularly during partitioning, leading to an early wire planning.

In this paper we present statistical timing driven hMetis-based partitioning. For our timing minimization purpose, we exploit the hyperedge coarsening scheme of hMetis [12] partitioner. This allows us to perform partitioning such that the most critical nets in the circuit are not cut and therefore timing minimization can be achieved. Our approach is a different way for doing timing-driven optimization: we drive the partitioning as for the best timing to be obtained without performing any netlist alteration (e.g., buffer insertion and gate duplication), though our method can be easily modified to incorporate these techniques as well. The main contribution of our work is the use of a better timing criticality and of a new delay model within a net-based partitioning approach (using a fast partitioner), which proves to provide circuits that are more tolerant to delay variations. By improving on timing by minimizing critical wire delays at partitioning level, we provide a way of doing wire planning very early in the physical design process.

The remainder of the paper is organized as follows. Section 2 presents previous work on timing driven partitioning. Section 3 presents the criticality concept that we use as edge weight for the hMetis partitioner. In Section 4 we describe our proposed statistical timing driven partitioning methodology. The delay model that we use is presented in Section 5. Simulation results are presented in Section 6. We demonstrate the robustness of our partitioning algorithm in Section 7. We conclude, suggesting further research directions, in Section 8.

2. Previous Work

Timing driven partitioning approaches can be classified into two categories: (1) *top-down* partitioning approaches and (2) *bottom-up* clustering-based approaches. Approaches in the first category are usually based on the Fiduccia-Mattheyses (FM) [7] recursive min-cut partitioning method or on quadratic programming formulations [16], [20]. Timing optimization is obtained by minimizing the delay of the most critical path. The second category includes bottom-up clustering-based approaches. They are used mostly as a pre-processing step for min-cut algorithms [6], [14]. All previous approaches achieve delay minimization by netlist alteration such as

Proceedings of the 16th International Conference on VLSI Design (VLSI'03)
1063-9667/03 $17.00 © 2003 **IEEE**

logic replication, retiming, and buffer insertion in order to meet delay constraints while the cutsize is minimized. The focus is on delay improvement, and the cutsize is ignored. Gate replication in these methods can be massive.

We can identify a few problems for all previous timing driven partitioning approaches: (1) Unrealistic delay models are used. It is common to use the *general-delay* model, which considers delay 1 for all gates, delay 0 for interconnects inside a partition, and a constant delay for interconnects between partitions [6], [14], [16]. (2) Unrealistic simplifications are made. For instance, circuits are mapped to two-input gates only [6]. (3) Static timing analysis is used as a framework for timing analysis. However, it is known that there are uncertainties in both gate and wire delays, such as fabrication variations, changes in supply voltage and temperature, that are not captured by the delay modeling within the framework of the classical static timing analysis. (4) The run time for moderate-sized circuits is too long and makes these approaches impracticable for large-sized circuits. One reason for that may be that previous approaches usually separates the timing-driven partitioning into two steps: (i) clustering or partitioning and (ii) timing refinement based on netlist alteration [6], [16].

In this paper, we try to eliminate the above deficiencies. We approach timing driven partitioning from a different perspective: we use the statistical timing criticality concept to change the partitioning process itself such that delay minimization is achieved while delay uncertainties are captured. We use a more realistic delay model, which incorporates a statistical net-length estimation and we use the hMetis partitioning algorithm which is very fast.

3. Statistical Timing Analysis

In this section we present the concept of criticality within the framework of statistical timing analysis versus static timing analysis. The idea of static timing analysis is to compute the slack for every gate based on the latest arrival time and the required arrival time values. Each gate has a constant delay value. However, in reality there are several uncertainties in both gate and wire delays, such as fabrication variations, changes in supply voltage and temperature [8], [15], [19]. These uncertainties are modeled in *statistical timing analysis* by considering gate and wire delays as stochastic variables (i.e. as probability distribution functions). That means that the delay variation is captured by the standard deviation. Even though different methods of statistical timing analysis have been proposed [11], [13], we adopt the approach proposed by Berkelaar [3] and later improved by Hashimoto and Onodera [8] who introduced the concept of *criticality* which fits well into the partitioning framework.

Generally, for an *n*-input gate (see Fig.1.a), under the assumption of stochastic independence of the inputs, the *maximum* latest arrival time at all inputs can be modeled with a normal distribution whose probability density function is [8]:

$$f_{maxPIs}(x) = \sum_{i}^{n} \left[f_i(x) \cdot \prod_{j \neq i}^{n} F_j(x) \right] \quad (1)$$

where f_i and F_j are the probability density function (pdf) and the cumulative density function (cdf) of input i respectively.

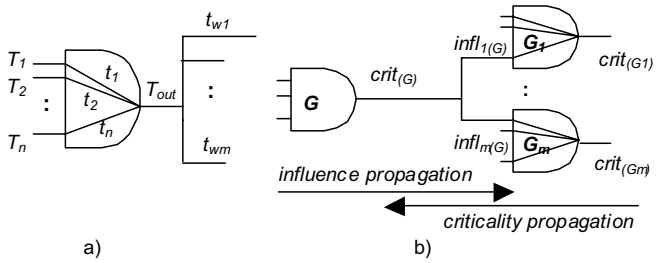

Fig.1 a) Example of general gate b) Influence and criticality computation

Since the internal gate delay[1] is also considered normally distributed, the gate output delay is calculated as the sum of two normal distributions: the maximum of all inputs and the internal gate delay. Wire delays are also considered stochastic variables. Hence, we can compute the probability density function of the overall circuit delay by computing the pdf of each primary output (PO). The equivalent of *slack* in static timing analysis are the notions of *influence* and *criticality* [8]. These notions address the problem of characterizing parts of the circuit from the point of view of timing similarly to the critical path concept. In what follows we briefly present the concepts of influence and criticality. The term between brackets in equation (1) represents the following probability:

$$f_i(x) \cdot \prod_{j \neq i}^{n} F_j(x) = P(T_i + t_i = x) \cdot \prod_{j \neq i}^{n} P(T_j + t_j \leq x) \quad (2)$$

The probability $P(T_i+t_i=x)$ expresses the magnitude of the *influence* that the i-th input gives to f_{maxPIs} at x. The influence $infl_i$ is defined as the influence proportion of the i-th input in the range $x>x_1$ as follows:

$$infl_i = C_1 \cdot \int_{x_1}^{\infty} f_i(x) \cdot \prod_{j \neq i}^{n} F_j(x) \cdot \exp(C_2 x) dx \quad (3)$$

where C_1 is a normalization coefficient to satisfy $\sum_i infl_i = 1$ and C_2 is a constant to emphasize the region of large arrival time. Criticality is meant to represent the timing criticality at each gate, i.e. the contribution to the circuit delay of all the paths that pass through that gate. It is computed using the following relation (see Fig.1.b):

[1] For the sake of simplicity we consider the internal delay from each input to output to be the same for a given gate. However, the gate delay is different for different gates.

$$crit(G) = \sum_{j}^{m} infl_{i(G)}(G_j) \cdot crit(G_j) \quad (4)$$

Equation (4) defines influence $infl_{i(G)}(G_j)$ as how much the $i(G)$-th input affects the timing at gate G_j for $x \geq x_j$. In other words, $infl_{i(G)}(G_j)$ represents how easily the timing criticality back-propagates from gate G_j to gate G. All influences are computed by propagation from primary inputs (PI's) towards PO's. Criticalities are computed by back-propagation from PO's towards PI's. *The gate with the largest criticality in a circuit is the most critical in terms of timing since its contribution to the circuit output delays is the most significant among all gates in the circuit.* Details can be found in [3], [8].

For example, the hypergraph shown in Fig.2.a as a Directed Acyclic Graph (DAG) depicts criticality values for all hyperedges. The corresponding circuit netlist is shown in Fig.2.b.

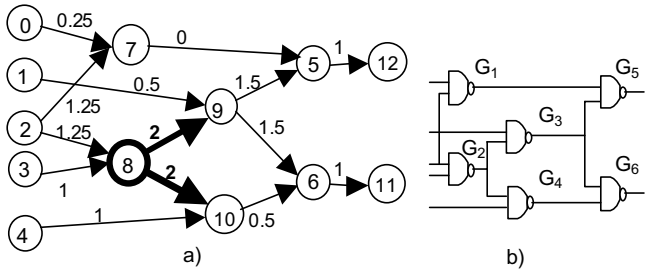

Fig.2 Illustration of the meaning of criticality: a) DAG with shown criticalities; hyperedge {8,9,10} is the most critical one because its associated criticality is the largest b) corresponding gate schematic

Gate G_2 in the circuit schematic (i.e. vertex 8 in the corresponding DAG) and its fanout net (i.e. hyperedge {8,9,10} in DAG) is the most critical one because its criticality, which equals 2, is the largest. In our partitioning methodology we want this hyperedge not to be cut because otherwise the circuit delay will increase.

In our partitioning methodology we use the criticality values as hyperedge weights. Thus, the hyperedge coarsening scheme of the hMetis partitioning algorithm clusters the most critical hyperedges early, which means that they would not be cut by the partitioning process. This has a great impact on the circuit timing, because the most critical nets in the circuit will not be cut during partitioning and subsequently, these critical nets will not become long/global interconnects.

The complexity of this statistical timing analysis and the calculation of all criticalities are linear with respect to the circuit size. [8]

One can argue that the slack for each node is also an indication of the gate criticality and thus the static timing analysis can be used in the same way. However, from our experiments that included both *static* and *statistical* timing analyses we found no one-to-one mapping between the gate criticality found by the static timing analysis and the gate criticality found by the statistical timing analysis. That means that a gate that is declared the most critical by the statistical timing analysis is not necessarily declared the most critical gate by the static timing analysis. We will show in Section 7 that the statistical timing based partitioning is more robust than the static slack-based partitioning.

4. Statistical Timing Driven Partitioning

In this section we present our statistical timing driven partitioning methodology. The partitioning is done by recursive bipartitioning. At each level we associate timing criticality as weight to all corresponding hyperedges in the hypergraph. Then, the hMetis partitioning algorithm is run using the hyperedge coarsening scheme. This scheme gives preference, during hypergraph coarsening, to the hyperdges that have large weights. By using timing criticality as hyperedge weight we practically discourage the partitioning algorithm from cutting edges with high delay criticalities.

Criticalities (i.e. hyperedge weights) are updated at each partitioning level. Initially we compute all criticalities in the circuit assuming zero delay for all wires. These criticalities are then used as weights associated to hyperedges. We call this process *forward annotation of criticalities*. After the first bipartitioning, we know which nets are cut and thus we are able to compute the delay for these wires by using the Elmore delay model. The wire delay calculation uses a statistical model for wire length proposed in [22]. These wire delays are then used to re-compute all criticalities in the circuit. We call this process *back annotation of the wire delays*. During the recursive bipartitioning we back annotate more and more wire delays. Hence, criticalities will reflect better the timing criticalities all over the circuit. The recursive bipartitioning process stops when each block contains a number of vertices smaller than a threshold specified by the user.

The pseudo-code of our statistical timing driven hMetis-based partitioning algorithm is as follows:

StatisticalTimingDrivenPartitioning( ) {

1. Compute initial criticalities; assign them as hyperedge weights
2. **Queue = G(V,E)** *// queue initialized with initial graph*
3. **while** *(**Queue** not empty)* {
    4. pop graph **g** from **Queue**
    5. partition **g** into **gA** and **gB** using hMetis
    6. push **gA** and/or **gB** in **Queue** if cardinality of their vertex set greater than **T** *// T = max # of gates allowed in each partition*
    7. Backannotate estimated Elmore delays to nets corresponding to cut hyperedges
    8. Update criticalities and edge weights
9. } }

5. Delay Model

Our delay model has two components. The first component is the gate delay. For all gates we consider a typical intrinsic delay that is given for a typical input

transition and a typical output net capacitance. This delay is actually the mean value of the pdf associated with the gate delay. For each pdf associated with all gates we consider a typical standard deviation of 15% [19]. The second component is the wire delay. We use the Elmore delay to model the wire delay. The Elmore delay for an edge *e* (an edge corresponds to the wire connecting the net source to one of its fanout sinks) is given by:

$$Delay(e) = R_e(\frac{C_e}{2} + C_t) \quad (5)$$

where R_e is the wire lumped resistance, C_e is the wire lumped capacitance, and C_t is the total lumped capacitance of the source node of each net. To compute R_e and C_e we need the length of each edge. For this, we use the statistical net-length estimation proposed in [22]. The average length of a net, connecting *m* cells enclosed in a rectangular area whose width is *a* and whose height is *b*, is given by:

$$L_{av} \approx (\alpha \cdot m^{\gamma} - \beta)\frac{a \cdot b}{a+b} + (a+b) \quad (6)$$

where α, β, and γ are fitting parameters computed in [22] as $\alpha \approx 1.1$, $\beta \approx 2.0$, and $\gamma \approx 0.5$. During recursive partitioning, when a net is cut, it is assigned a certain wire delay that will be used to re-compute all delays on the paths that include that net. The earlier a net is cut during recursive partitioning, the greater the back-annotated wire delay has to be. In our case, any net that is cut during the first bipartitioning step (see Fig.3) is assumed to be bounded by a rectangular area which is the same as the chip area and for simplicity we consider an aspect ratio equal to 1.

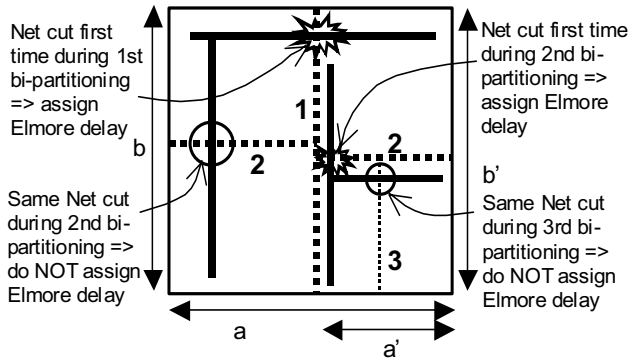

Fig.3 Illustration of the wire delay assignment to cut nets at different bipartitioning levels

At the second partitioning level *a* and *b* have different values that will ensure a smaller delay than that assigned during a previous partitioning level. The delay of each net is set only the first time when it is cut. In other words, if a net is cut again at a lower partitioning level, it does not have its delay increased or re-assigned (based on the net length estimation corresponding to the bounding box at this partitioning level) because otherwise its delay would be over increased. In our experiments we consider a 0.18μ copper process technology (unit length resistance r = 0.115, unit length capacitance c = 0.00015).

6. Simulation Results

In this section, we present simulation results. It is difficult for us to make a meaningful comparison of our *statistical* results with previous *static* timing analysis based works (except for the experiments presented in Section 7) because: (i) Our approach is based on statistical timing analysis, which is different from all previous approaches that are based on static timing analysis. Hence, we cannot compare statistical delay to static delay. (ii) We do not use netlist alteration in order to meet a timing requirement but we minimize timing by changing the partitioning process itself. Our goal is to show the potential timing improvement that can be obtained using our methodology, which can be further enhanced by using different netlist alteration techniques (iii) We use a different delay model vs. all previous approaches that are based either on the unit-delay model or on the global delay model. However, we compare our method against the case when the weights in graphs are constant corresponding to the case when simply hMetis would be used for circuit partitioning (we call it the *pure* partitioning method). *In this way, we show the potential timing improvement that can be obtained using our algorithm.* The experimental setup is shown in Fig.4.

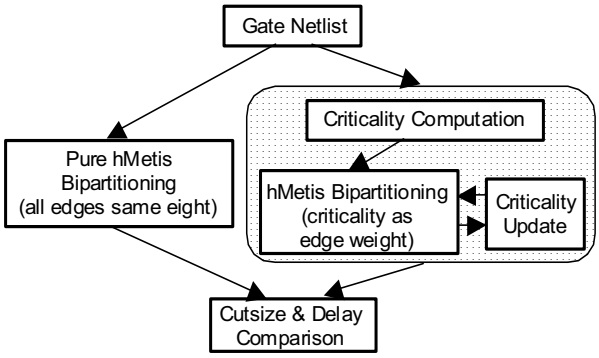

Fig.4 The simulation setup for comparison of our proposed partitioning algorithm to pure hMetis algorithm

We report simulation results for a set of circuits from the ISCAS89 benchmarks [10] and the largest four ITC99 benchmarks [9] (last four in Table 1). All circuits were first optimized using the *script.rugged* in SIS [18]. The results are presented in Table 1. The second column in Table 1 indicates the number of PI's and PO's, followed by the number of gates in the third column. For each circuit, *Cutsize* represents the number of all edges cut after the recursive bipartitioning. The *Delay* indicates the maximum mean delay (using the statistical delay model) among all PO's. The run time is rounded to the closest integer and is given in seconds.

We run the partitioning algorithm 60 times and report the average in Table 1. The maximum number of gates allowed for each partition was set to 10% of the total

number of gates (i.e. we did 10-way partitioning). As it can be seen, the *proposed partitioning methodology offers in average a 22% better del*ay. However, this is *at the expense of an increase of 33% in the cutsize*. On one hand, we obtain a better delay with our partitioning algorithm because we use a better statistical timing criticality as hyperedge weight. On the other hand, compared to the pure hMetis partitioning, the cutsize increases because we practically reduce the search space for the hMetis partitioner when criticality is used as hyperedge weight. The partitioner does not have the same freedom in exploring the search space as when all hyperedges have the same weight.

Table 1:Recursive statistical timing hMetis-based partitioning vs. hMetis partitioning for moderate-sized circuits (CPU(s) on UltraSPARC-II 450MHz, 2GB memory)

Circuit	PI/PO	No. of gates	Statistical Timing Driven hMetis-based Partitioning			Pure hMetis Partitioning		
			Cutsize	Delay	CPU(s)	Cutsize	Delay	CPU(s)
cordic	23/2	856	307	**24.8**	43	325	29.9	12
s9234	39/16	1257	220	**15.7**	32	174	16.38	11
misex3	14/14	1321	578	**62.5**	60	496	75.8	16
s13207	90/56	1570	237	**21.2**	40	202	27.9	18
frisc	19/16	3479	1318	**99.6**	131	797	163.4	39
too_large	38/3	6920	2656	**135.1**	297	2392	144.5	71
s35932	35/32	11304	792	**161.1**	239	415	249	58
s38584	115/74	12701	1304	**192.7**	287	499	279	73
mult32	64/64	12813	684	**9112**	238	581	9251	55
b21s	32/22	14606	0.94	**0.26**	232	1	1	55
b22s	32/22	22497	1.08	**0.74**	177	1	1	38
b17s	37/30	36547	1.17	**0.93**	324	1	1	86
b18s	36/22	101573	1.14	**0.92**	1080	1	1	292
Avg.			**1.33**	**0.78**	**2.74**	**1**	**1**	**1**

We observed that if we stretched the original criticality range into a smaller range we could trade delay vs. cutsize. In other words, if for example the original criticality range is *[0,max_old_crit]* we can stretch it into *[1,max_new_crit]*, with *max_new_crit* taking values 2, 5, 10, 50,…, and still obtain delay improvement with smaller increase in cutsize. For example, the values for delay and cutsize for *s38584* are as follows:

max_new_crit	2	5	10	50	100	200	500
delay	294.3	292	299	245.4	245.5	245.4	188.6
cutsize	496	526	507	602	637	572	818

Similar cutsize/delay tradeoff was observed for all circuits. This allows the user to "tune" the partitioning method to smoothly tradeoff between cutsize and delay.

The run time for our methodology is greater due to the criticality update operation. The run time for the pure hMetis algorithm includes the recording of the cut wires at each level of the partitioning as well as the delay computation.

7. Validation Scenarios

In this section, we describe two simple scenarios to further demonstrate the robustness of the statistical timing driven partitioning. It is known that due to the increase in chip clock frequency the amount of power consumption also increases, resulting in an increase in the chip temperature. However, the heat dissipation is usually unevenly distributed among the circuit gates, which leads to various temperatures across the whole area of the chip [21]. On one hand, higher temperatures slow down the transistors [5]. On the other hand, the interconnect resistance increases linearly with the temperature [1]. Thus, the delay of all gates and wires in areas with higher temperature will be larger than their estimated values during the design process. This motivated us to come up with two simple scenarios for testing the robustness of our proposed statistical timing driven partitioning. These scenarios are depicted in Fig.5.

In both cases we first perform recursive bipartitioning using our statistical timing driven partitioning algorithm or pure hMetis partitioning algorithm or a slack-based partitioning algorithm[2]. Then, we perform a *static* timing analysis to compute the maximum delay among all the PO's; we denote this delay as *delay1*. Third, in the first scenario (see Fig.5.a), we consider a 15% delay increase for all gates and their fanout wires that are placed in one of the partitions after the first level of bipartitioning. We choose a typical 15% for the delay increase [19] though for large temperature variations this increase can be larger [1]. In this way we try to mimic the case where half of the chip has a higher temperature, which in turn will determine a delay increase. Obviously, in reality, the temperature pattern will be more complex [4], but we restrict ourselves to this simplified version, which is similar to the example presented in [21].

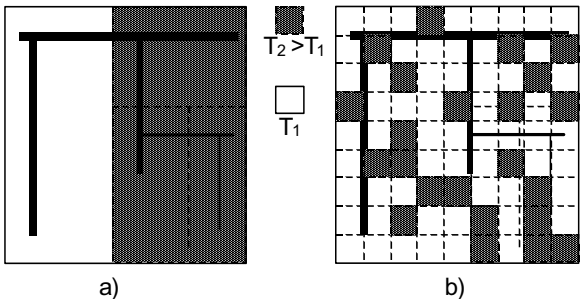

Fig.5 a) Half of the chip hotter b) Random hot spots over the whole chip area

In the second scenario (see Fig.5.b) we randomly choose 15% delay increase for all gates and their fanout wires. This case tries to mimic the situation when we can find hot spots everywhere on the chip [5].

Then, we perform a second *static* timing analysis to compute the maximum delay among all the PO's of the circuit using the new gate and wire delays; we denote this delay as *delay2*. Finally, we compute the overall circuit delay perturbation as 100*(delay2-delay1)/delay1. The simulation results, presented in Table 2, confirm that by

[2] We developed a slack-based partitioning algorithm similar to the proposed one except that instead of statistical criticality we used static criticality, which is computed inversely proportional with the slack. The smallest slack determines the largest weight associated to the corresponding hyperedge, and so on so forth.

using our partitioning algorithm the perturbation due to temperature variation of the circuit delay is in most of the cases smaller when we use our partitioning algorithm (17% and 13% in average smaller for the half-hotter scenario and for the random-hot-spots scenario compared to the pure hMetis; 29% and 8% in average smaller for the half-hotter scenario and for the random-hot-spots scenario compared to the static slack-based partitioning). In this way circuits are more stable under the disturbing influence of factors such as temperature, power supply fluctuations, and process variations.

Table 2:Delay change as percentage for the two validation scenarios

Circuit	**100*(delay2-delay1)/delay1**					
	Proposed algorithm		Pure hMetis		Slack-based algorithm	
	"half-hotter"	"random-hot-spots"	"half-hotter"	"random-hot-spots"	"half-hotter"	"random-hot-spots"
cordic	11.66	8.46	8.46	8.36	11.9	6.72
s9234	7.26	8.04	10.44	8.64	9.36	9.63
misex3	8.45	6.25	10.1	12.31	12.73	12.13
s13207	7.05	6.3	5.72	4.65	14.3	6.45
frisc	6.6	5.86	11.51	3.7	7.22	7.9
too_large	5.56	6.76	5.73	8.56	10.73	8.73
s35932	8.86	4.04	10.6	8.93	8.4	7.5
s38584	3.86	13.5	9.46	10	7.5	6.64
mult32	6.96	6.76	7.53	10.74	12.18	5.98
Avg.	**7.36**	**7.33**	**8.83**	**8.43**	**10.48**	**7.96**

8. Conclusion

In this paper we propose a timing driven partitioning algorithm. Because we change the partitioning process itself and we use the hMetis algorithm our algorithm is fast, thus applicable to large-sized circuits. Because we use a new delay model, which better reflects the timing criticality inside circuits, our algorithm is robust and circuits are more reliable than the circuits partitioned using pure hMetis or the slack-based partitioning algorithms. The proposed algorithm does not determine area increase because we do not use netlist alteration and it offers a smooth cutsize/delay tradeoff. The slight cutsize increase is the only disadvantage of our partitioning algorithm. We are currently working on multi-objective, multi-constraint hMetis-based partitioning methodologies.

References

[1] A.H. Ajami, K. Banerjee, M. Pedram, L. van Ginneken, 'Analysis of Non-Uniform Temperature-Dependent Interconnect Performance in High Performance ICs', *Proc. ACM/IEEE ASPDAC*, 2001.

[2] C.J. Alpert, 'The ISPD98 Circuit Benchmark Suite', *Proc. ISPD*, 1998.

[3] M. Berkelaar, 'Statistical Delay Calculation, a Linear Time Method', *Proc. TAU*, 1997.

[4] Y.K. Cheng, S.M. Kang, 'A Temperature-Aware Simulation Environment for Reliable ULSI Chip Design', *IEEE Trans. CAD*, Oct. 2000.

[5] C.C.N. Chu, D.F. Wong, 'A Matrix Approach to Thermal Placement', *IEEE Trans. CAD*, Nov. 1998.

[6] J. Cong, C. Wu, 'Global Clustering-Based Performance-Driven Circuit Partitioning', *Proc. ISPD*, 2002.

[7] C.M. Fiduccia, R.M. Mattheyses, 'A Linear-Time Heuristic for Improving Network Partitions', *Proc. ACM/IEEE DAC*, 1982.

[8] M. Hashimoto, H. Onodera, 'A Performance Optimization Method by Gate Resizing Based on Statistical Static Timing Analysis', *IEICE Trans. Fundamentals*, Dec. 2000.

[9] http://www.cad.polito.it/tools/9.html

[10] http://www.cbl.ncsu.edu

[11] H.-F. Jyu, S. Malik, 'Statistical Delay Modeling in Logic Design and Synthesis', *Proc. ACM/IEEE DAC*, 1994.

[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, 'Multilevel Hypergraph Partitioning: Application in VLSI domain', *Proc. ACM/IEEE DAC*, June 1997.

[13] J.-J Liou, K.-T Cheng, S. Kundu, A. Krstic, 'Fast Statistical Timing Analysis By Probabilistic Event Propagation', *Proc. ACM/IEEE DAC*, 2001.

[14] J. Minami, T. Koide, S. Wakabayashi, 'An Iterative Improvement Circuit Partitioning Algorithm under Path Delay Constraints', *IEICE Trans. Fundamentals*, Dec. 2000.

[15] S.R. Nassif, 'Modeling and Forecasting of Manufacturing Variations', *Proc. ACM/IEEE ASPDAC*, 2001.

[16] S.-L Ou, M. Pedram, 'Timing-driven Partitioning Using Iterative Quadratic Programming', at http://atrak.usc.edu/~massoud/, see "Coming Attractions!", 2001.

[17] Alberto Sangiovanni-Vincentelli, 'Defining platform-based design', http://www.eedesign.com/features/exclusive/OEG20020204S0062, 2002.

[18] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A. Sangiovanni-Vincentelli, 'SIS: A System for Sequential Circuit Synthesis', *Technical Report UCB/ERL M92/41*, University of California, Berkeley, May 1992.

[19] D. Sylvester, 'Measurement Techniques and Interconnect Estimation', *SLIP00*, 2000.

[20] M. Shih, E.S. Kuh, 'Quadratic Boolean Programming for Performance-driven System Partitioning', *Proc. ACM/IEEE DAC*, 1993.

[21] C-H.Tsai, S.M. Kang, 'Cell-Level Placement for Improving Substrate Thermal Distribution', *IEEE Trans. CAD*, Feb. 2000.

[22] P. Zarkesh-Ha, J.A. Davis, J.D. Meindl, 'Prediction of Net-Length Distribution for Global Interconnects in a Heterogeneous System-on-a-Chip', *IEEE Trans. VLSI Systems*, Dec. 2000.