Multi-Way Partitioning of VLSI Circuits

Prathima Agrawal and B. Narendran
{pa,naren}@research.att.com
Computing Systems Research Lab
AT&T Bell Laboratories
Murray Hill, NJ 07974

Narayanan Shivakumar
shiva@cs.stanford.edu
Department of Computer Science
Stanford University, Stanford, CA 94305

Abstract

Partitioning is one of the critical phases of hierarchical design processes like VLSI design. Good partitioning techniques can positively influence the performance and cost of a VLSI product. This paper proposes a partitioning algorithm with a new cost metric. Viewed from a VLSI layout point of view our cost metric minimizes the average delay per net. It can also be interpreted as achieving the minimum number of vias per net. This paper highlights how the seemingly slight difference between our metric and others could cause partitions to be evaluated considerably differently. Experimental results show that in addition to the expected improvements we get on our metric, the proposed algorithm does well on the traditional nets cut metric as well.

1 Introduction

The multi-way partitioning problem is one of partitioning the modules in a network into K subsets (partitions) of "approximately" the same size while minimizing the amount of interaction between the K partitions. This problem has several applications in VLSI circuit design ranging from circuit layout to logic simulation and emulation. K-way partitioning is typically used in a divide-and-conquer methodology so that complicated problems in CAD are solved by partitioning the problem into K smaller problems and solving the simpler sub-problems. For instance, partitioning is used as a pre-processing step to routing. In addition to VLSI applications, partitioning problems also arise in other areas like layout of printed circuit boards, hierarchical system design, and task/data assignment in parallel processing.

While minimizing the "interaction" between partitions remains the general goal of all partitioning, the exact metric to be optimized depends on the specific application context. For example, a common metric that is used in the bi-partitioning case ($K = 2$) is the number of nets that are cut by the partition. While this metric exactly captures the interconnection required for the bi-partition case, it does not generalize well to the multi-partitioning case. In fact, it is observed in [YeCL91] that the choice of cost metrics for different applications becomes more critical as the number of partitions increase.

The application context and the cost metric to be minimized also influence the algorithm to be used to do the partitioning. The general network partitioning problem is known to be NP-hard. Hence, a number of heuristics are used to solve the problem efficiently. The existing partitioning heuristics in the literature can be divided into bi-partitioning and multi-partitioning heuristics. The bi-partitioning heuristics include the iterative improvement methods, the graph spectral methods, min-cut algorithms, and net-based partitioning methods. The best known iterative improvement algorithms include the Kernighan-Lin (KL) [KeLi70], Fiduccia-Matheysses (FM) [FiMa82], and simulated annealing [KiGV83] algorithms. The graph spectral method computes a bi-partitioning solution based on the second smallest eigenvector of the Laplacian matrix of the given network [DoHo73, HaKa91]. The min-cut methods use maximum flow algorithms to compute a series of minimum cuts in the given circuit [YaWo94]. The net-based partitioning approach first computes a bipartitioning of nets, and then transforms the net partitioning to a module partitioning solution [HaKa92b, CoHK92].

The multi-way partitioning algorithms include the recursive bi-partitioning by Kernighan and Lin [KeLi70], a generalization of the FM-algorithm with lookahead by Sanchis [Sa89], a generalization of the graph spectral-based partitioning by Chan, Schlag, and Zien [ChSZ93], the primal-dual algorithms of [YeCL91], and a multi-way generalization of the net-based partitioning approach by Cong, Labio and Shivakumar [CoLS94].

The contributions of this paper are threefold. First, we propose a cost metric for partitioning that is somewhat different from what has been proposed in the literature. In Section 2, we define our cost metric, *Via-Count*, and justify its use. We show how our metric is more suitable to capture some of the physical attributes of multi-way partitioning in the VLSI context. A particular motivating factor behind the new metric was to capture the average delay a net experiences due to inter-partition communication of signals. We also show how the seemingly slight difference between our metric and other metrics could cause partitions to be evaluated considerably differently. Next, in Section 3, we present K-VIA-FM, a simple extension of a well known linear time heuristic for optimizing the number of nets cut in bi-partitioning [FiMa82] to optimize multi-way partitioning using our metric. Previous work along similar lines [Sa89] had to deal with a more complicated extension because of the difference in the metric being used. Finally, we present experimental results in Section 4. We show the results of partitioning the circuits in the MCNC suite, and compare the performance of K-FM [Sa89], K-DualPART/SF and K-DualPART/DF [CoLS94], and K-VIA-FM. We beat the K-FM and the K-DualPART algorithms on average from 31 to 70 % in terms of *Via-Count*. We also evaluate the partitioning results by considering the average number of vias per net. To validate the signal delay latency motivation of our metric, we also apply the partitioning algorithm to in a standard cell design context, and obtain very promising results. K-VIA-FM has an average of 28 to 58 % less estimated *signal latency* than do the solutions from K-FM and K-DualPART. We show that in addition to the expected improvements we get on the metric of our interest, the proposed algorithm does well on the traditional nets cut metric as well.

2 Cost Metrics for Partitioning

A network is defined to consist of a set $\mathcal{M}$ of *modules*, that are connected by a set $\mathcal{N}$ of *nets*. Each net $N \in \mathcal{N}$ is represented by the set of modules of $\mathcal{M}$ that it connects. Each module $M \in \mathcal{M}$ also has an associated *size* parameter, which we will denote by $\text{area}(M)$. The partitioning problem is to find a partition π of the set of modules into K sets $(\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots, \mathcal{M}_k)$, so as to optimize some objective function, and such that each partition satisfies some constraints on the total size of all its modules.

The size constraint on the partitions is usually some kind of an approximate "equipartitioning" constraint. Let $A = \sum_{M \in \mathcal{M}} area(M)$ be the total area of all modules, and $A_i = \sum_{M \in \mathcal{M}_i} area(M)$ denote the area of the i^{th} partition. Then, the size constraint requires that

$$(1-\alpha)\frac{A}{K} \;\le\; A_i \;\le\; (1+\alpha)\frac{A}{K}, \qquad i = 1, \ldots, K$$

In general, the user could specify any general constraint on the sizes of the partitions.

Various objective functions have been used in the literature, depending on the specific application context in which the partitioning is performed. One that is commonly used the number of nets that are cut by the partition, which we denote by $NC_\pi(\mathcal{M}, \mathcal{N})$:

$$NC_\pi(\mathcal{M}, \mathcal{N}) \;=\; |\{N \in \mathcal{N} \mid \exists i, j, i \ne j, N \cap \mathcal{M}_i \ne \phi, N \cap \mathcal{M}_j \ne \phi\}|$$

While this metric, and associated algorithms to optimize it had its origins in the bipartitioning case [KeLi70, FiMa82], it has been extended to multipartitioning as well [Sa89].

The ratio-cut metric [WeCh89, ChSZ93, YeCL92] is a variant of the NC metric defined above where the size constraint has been moved into the objective function. The ratio-cut objective function is given by

$$\text{Ratio-cut}_\pi(\mathcal{M},\mathcal{N}) = \frac{NC_\pi(\mathcal{M},\mathcal{N})}{\prod_{i,j,\ i\neq j}\text{area}(\mathcal{M}_i)\text{area}(\mathcal{M}_j)}$$

The *span* metric was used in [YeCL91], where the span of a net was defined to be 0 if the net is contained entirely in a single partition, and the number of partitions it spans, otherwise:

$$\text{span}_\pi(N) = \begin{cases} 0, & \text{if N is contained entirely in a single partition,} \\ |\{i|N \cap \mathcal{M}_i \neq \phi\}|, & \textit{otherwise.} \end{cases}$$

The objective function was to minimize the span of the network which was defined to be $\sum_{N\in\mathcal{N}} \text{span}_\pi(N)$.

In this paper, we use a cost metric that is similar to *span*. Our motivation was to capture the physical quantity that corresponds to the actual number of interconnects that are needed between partitions. To this end, we define *Via-Count* [1] for a particular net N as

$$\text{Via-Count}_\pi(N) = |\{i|N \cap \mathcal{M}_i \neq \phi\}| - 1$$

In the VLSI design process, the metric *Via-Count* could signify different physical quantities in different contexts. At a higher level, for instance, the number of interconnects between partitions represents the number of connections that have to be realized using (slower) external wires instead of faster intra-partition connections [HaKa92]. If we assume that delays are dominated by the slower external links, then the metric captures the average delay experienced by a net in the network.

Although our metric is very similar to the *span* metric (for a given net, the two are off by at most 1), we illustrate by an example how they may rate a given partition quite differently. In Figure 1 we show two different partitions, both of which are ranked identically by the *span* metric. We note that in one case, there are a large number of nets cut, but each net is cut only once. In

[1]The name *Via-Count* for the metric is a reference to the particular instance of partitioning of a VLSI netlist into layers, where each interconnect accounts for a *via*[WeEs85]. In other applications, the metric could signify other physical quantities of interest.

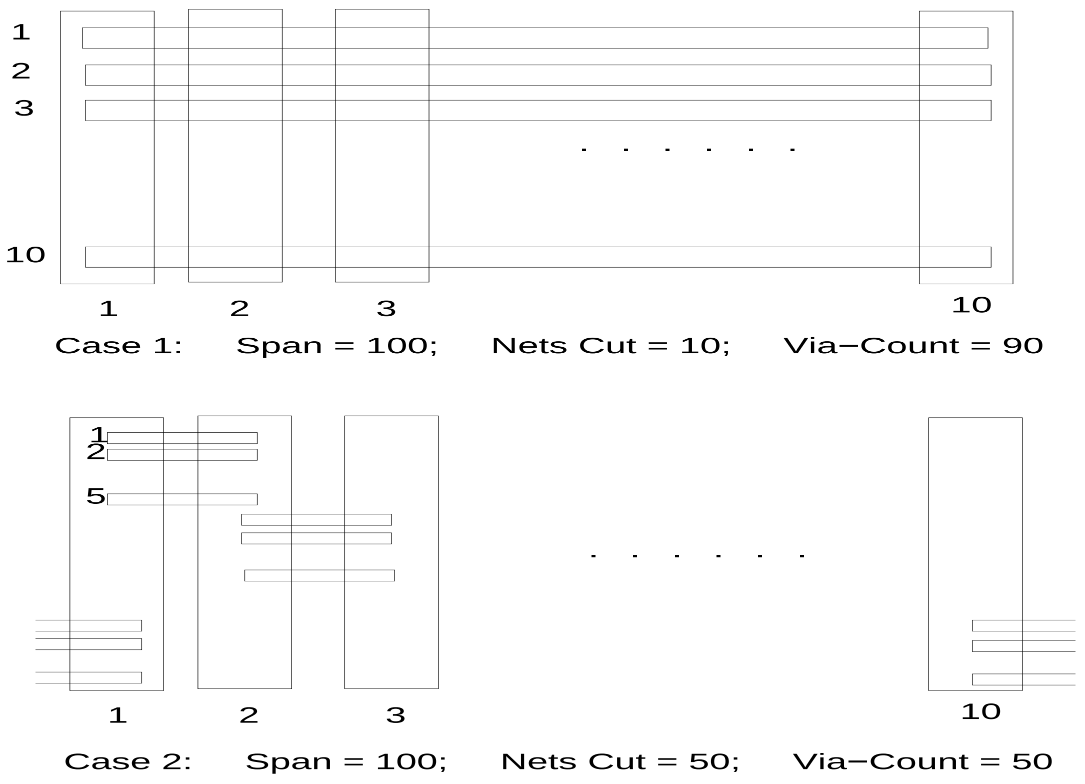
1
2
3
10
1
2
3
10
Case 1: Span = 100; Nets Cut = 10; Via−Count = 90
1
2
5
1
2
3
10
Case 2: Span = 100; Nets Cut = 50; Via−Count = 50

Figure 1: Differences between the cost metrics: Two partitioning cases are illustrated. The vertical rectangles are partitions, and each horizontal rectangle is a net.

the other case, fewer nets are cut, but more often. The *Via-count* distinguishes these two cases, where *span* does not.

We note that for the entire net, the following relationship holds between the two metrics:

$$\text{span}_\pi(\mathcal{M},\mathcal{N}) = \text{Via-count}_\pi(\mathcal{M},\mathcal{N}) + \text{NC}_\pi(\mathcal{M},\mathcal{N}).$$

To conclude this section, we state formally the optimization problem we address in this paper. Given a network $(\mathcal{M},\mathcal{N})$, and a parameter K representing the number of partitions, we wish to find a partition $\pi = (\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots, \mathcal{M}_k)$ of $\mathcal{M}$ such that the number of modules in each partition is within the user-specified bounds, that is

$$(1-\alpha)\frac{A}{K} \le A_i \le (1+\alpha)\frac{A}{K}$$

for all partitions $\mathcal{M}_i$, where A_i is the total area of all the modules in $\mathcal{M}_i$, and α is a user-specified parameter controlling the allowable slack in equi-partitioning constraint. Furthermore, we wish to minimize $\text{Via-Count}_\pi(\mathcal{M},\mathcal{N})$ subject to the above constraint.

3 The Partitioning Algorithm

In this section, we propose K-VIA-FM, a simple multi-way extension of the FM algorithm of Fiduccia and Matheyses [FiMa82] to iteratively reduce the *Via-count* of the partitioning solution. We choose the FM algorithm as the basis of our algorithm due to its simplicity, its linear time performance [FiMa82], and also due to its popularity as a post-processor in improving the solution quality of other deterministic partitioning algorithms [CoLS94].

In the FM algorithm and its derivatives [Sa89, Kr84], a given initial partitioning solution is chosen and is iteratively improved by moving modules between partitions [2]. Modules are chosen to be moved from one partition to another based on the *gain* induced by the module in moving to the new partition, where the gain is computed to be the improvement in quality (cost metric) of the partitioning solution when the module moves to the new partition. The FM algorithm uses an efficient gain computation strategy in which each module incrementally maintains in its gain array the gain in moving to each of the K partitions.

For a given module M, let *M.partition* represent the partition the module is currently assigned to, and *M.area* refer to the area of the module. Let *M.gain[i]* represent the improvement in the cost metric when module M moves to the i^{th} partition. For a given net N, let *N.count[i]* represent the number of modules belonging to net N and assigned to the i^{th} partition, $i = 1, 2, \ldots, K$. Let *Area[i]* represent the area of the i^{th} partition. In Figure 2 we outline the basic steps in the *K-VIA-FM* algorithm in pseudocode fashion. In Figure 3 we present the key steps in initializing the gain arrays of the modules, and in Figure 4 we present the steps required to incrementally maintain the gain arrays when a module is chosen to be moved from one partition to another.

In Section 1, we noted that another multi-way generalization of FM (which we call K-FM) was proposed by Sanchis [Sa89], where the objective function was the number of nets cut. As we shall see, the use of our metric actually allows us to use a simpler extension of FM than the one presented there. The reason for this is that for the nets cut metric, the gain values have to measure the potential of *all* of a net's modules being assigned to a single partition. For the *Via-Count* metric, on the other hand, the gain values measure the potential of a "local" event - that of a single interconnect between partitions being eliminated. This difference implies that our algorithm has to keep track of less information than K-FM to compute the gains.

[2]In practice, several random initial partitions are chosen and the best final solution is chosen after iteratively improving each of the initial partitions.

```
K-VIA-FM()
begin
      while (more gain possible)
      begin
     Create initial partition and initialize N.Count for each net N
             InitializeGain()
             Unlock all modules
             while (no more unlocked modules that can be moved)
             begin
                   Choose unlocked module M and destination partition "to"
                   that maximizes the value of M.gain[to], and such that
                   the move preserves size constraints on partitions.

                   from = M.partition
                   UpdateGain(M, from, to)

                   Lock module M
                   Area[from]  -= M.area       // maintaining areas of
                   Area[to]    += M.area       // from and to partition
             end
      end
end
```

Figure 2: Outline of K-VIA-FM.

```
InitializeGain()
begin
      for every module M in netlist
      begin
             for all i = 1, 2, . . ., K
             begin
                   M.gain[i] = 0
             end
             from = M.partition            // the current partition of M
             Area[from] += M.area          // Area of partition from

             for every net N containing M
             begin
                   for all j = 1, 2, . . ., K, j != from
                   begin
                   //
                   // computing gain of moving module M from
                   // "from" partition to "j" partition
                   //
                           if (N.count[j] > 0) and (N.count[from] = 1)
                                  M.gain[j] += 1
                           if (N.count[j] = 0) and (N.count[from] > 1)
                                  M.gain[j] -= 1
                   end
             end
      end
end
```

Figure 3: Initializing Gains

```
Let M be the module that was chosen to be moved from partition "from"
to partition "to".

UpdateGain(module M, partition from, partition to)
begin
      for every net N containing M
      begin
             for every module m in N, m != M
             begin
                     if (N.count[from] = 1)
                               m.gain[from] -= 1
                     if (N.count[from] = 2) and (m.partition = from)
                          for all i = 1, 2, . . ., k, i != from
                          begin
                               if (N.count[i] > 0)
                                       m.gain[i] += 1
                          end
             end

             N.count[from] -= 1        // reflecting new count of
             N.count[to] += 1          // net N after M has moved

             for every module m in N, m != M
             begin
                     if (N.count[to] = 1)
                             m.gain[to] += 1
                     if (N.count[to] = 2) and (m.partition = to)
                          for all i = 1, 2, . . ., k, i != to
                          begin
                               if (N.count[i] > 0)
                                       m.gain[i] -= 1
                          end
             end
      end
end
```

Figure 4: Updating Gains

	Number of Modules	Number of Nets
Test02	1724	1721
Test03	1664	1618
Test04	1541	1658
Test05	2650	2751
Test06	1813	1674
PrimGA1	914	902
PrimGA2	3121	3029

Figure 5: Some Information on MCNC Benchmarks.

4 Experimental Results

We implemented the K-FM from [Sa89], and the modified K-VIA-FM on SUN SPARC workstations. We compared the two algorithms against K-DualPART/DF and K-DualPART/SF from [CoLS94] which used the equi-constrained min-nets cut metric. We used the MCNC benchmark circuits for our comparisons.

Figure 5 shows the characteristics of the benchmark circuits including the number of modules and number of nets in the circuits. Figures 6 through 9 show the comparison of K-VIA-FM against K-FM and K-DualPART/SF and K-DualPART/DF in terms of the number of nets cut and the *Via-Count* for K = 5 and K = 10. We ran the K-DualPART algorithms once with the greedy initial placement, and the K-FM and K-VIA-FM algorithms approximately 10 times on random initial partitions to ensure comparable run-times. The module slack parameter was set to 10% in all the simulations.

One can see from the simulation results that K-VIA-FM consistently outperforms the K-DualPART algorithms in terms of number of the *Via-Count* metric. It is very interesting to note that the performance of K-VIA-FM is enhanced as K increases thereby indicating the scalability of the algorithm for high K as expected. A surprising result is that K-VIA-FM, with a mere change in the gain computation model and a different metric, does significantly better than the other three algorithms not only in the number of vias, but also in terms of nets cut (19 - 52 %). This suggests that the new metric could actually be used in other hyper-graph multi-partitioning applications where the conventional metric of minimizing the number of hyper-edges cut is required.

	KFM	K-DP/SF	K-DP/DF	K-VIA-FM
Test02	1174	765	819	468
Test03	1093	577	554	372
Test04	1158	448	427	386
Test05	1857	781	757	562
Test06	1201	859	832	422
PrimGA1	616	383	340	225
PrimGA2	2835	1333	1279	964
% improvement	65.26	33.72	31.25	

Figure 6: **Via-Count** Comparison for K = 5.

	KFM	K-DP/SF	K-DP/DF	K-VIA-FM
Test02	1915	1173	1182	624
Test03	1907	862	912	521
Test04	1893	796	855	527
Test05	3155	1304	1265	825
Test06	2076	1235	1245	625
PrimGA1	1058	549	534	341
PrimGA2	4534	1947	1886	1278
% improvement	70.70	39.63	40.04	

Figure 7: **Via-Count** Comparison for K = 10.

	KFM	K-DP/SF	K-DP/DF	K-VIA-FM
Test02	672	501	524	386
Test03	601	413	434	311
Test04	678	322	299	321
Test05	1076	581	559	457
Test06	649	551	545	313
PrimGA1	352	269	241	196
PrimGA2	1475	1013	903	784
% improvement	49.13	23.17	20.01	

Figure 8: Nets Cut Comparison for K = 5.

	KFM	K-DP/SF	K-DP/DF	K-VIA-FM
Test02	926	649	604	475
Test03	893	499	499	427
Test04	910	545	533	411
Test05	1524	740	740	624
Test06	934	637	601	425
PrimGA1	508	329	313	278
PrimGA2	1916	1178	1178	974
% improvement	51.96	21.09	18.88	

Figure 9: Nets Cut Comparison for K = 10.

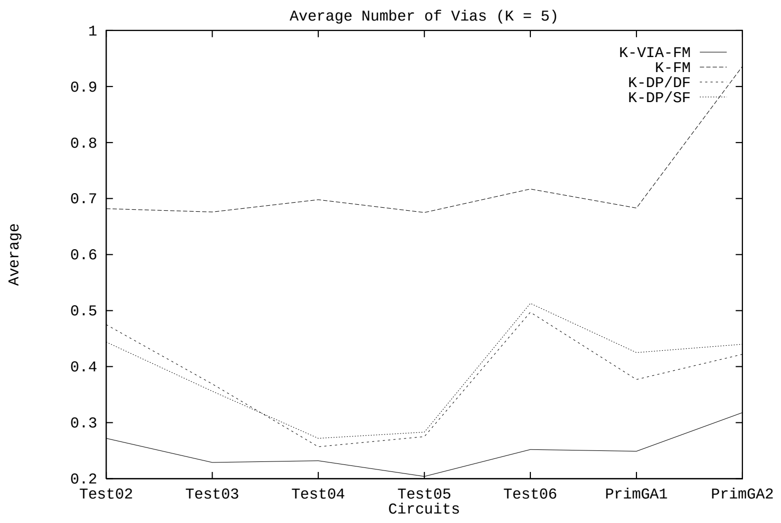

Figure 10: Average Number of Vias per Net (K = 5).

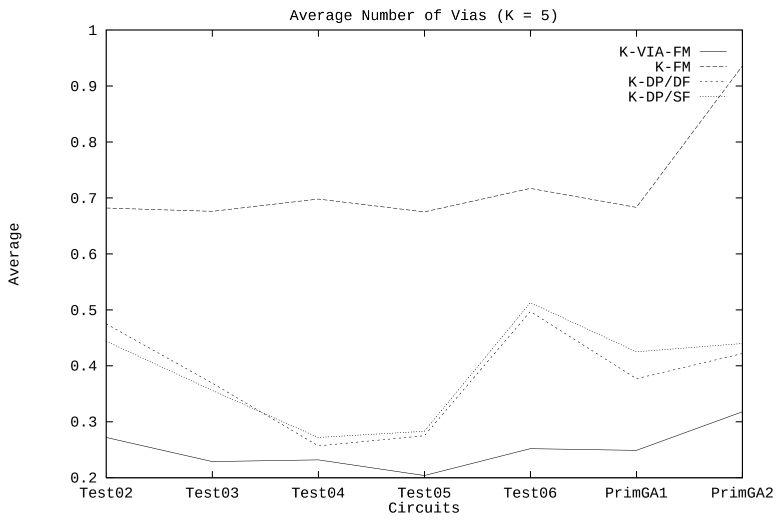

Figure 11: Average Number of Vias per Net (K = 10).

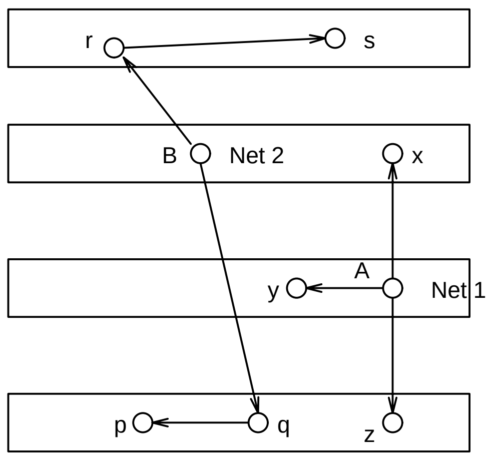

Figure 12: Standard Cell Embedding.

As described earlier, one of the attributes of the partition the *Via-Count* metric was designed to capture was the average delay experienced by a net. The assumption here is that the delay experienced in a net is proportional to the number of (high latency) inter-partition links it suffers. In Figures 10 and 11, we present the average number of vias per net as a measure of the expected signal latencies in a general partitioning example. A more realistic validation of the correlation between our metric and delay estimates, in the context of standard cell design, is presented in the following Section.

4.1 Delay Estimation in Standard Cell Design

In standard cell design, modules are partitioned into rows as in Figure 12. Hence to evaluate the suite of algorithms in terms of average signal latency in a real design problem, we embedded our modules into rows. After partitioning, we estimate the signal latency for each net to be the distance of the farthest module from the source of the net. For instance, in Figure 12, the estimated latency of net 1 is one since both modules x and y are one row away from the source of the net, A. Similarly, the net latency of net 2, is two. Since inter-partition vias constitute the most significant communication cost in a net, we believe our metric to be a very natural metric at the partitioning stage.

Figures 13 and 14 plot the results of these experiments. As can be observed, K-VIA-FM does indeed produce the lowest estimated delays amongst the algorithms.

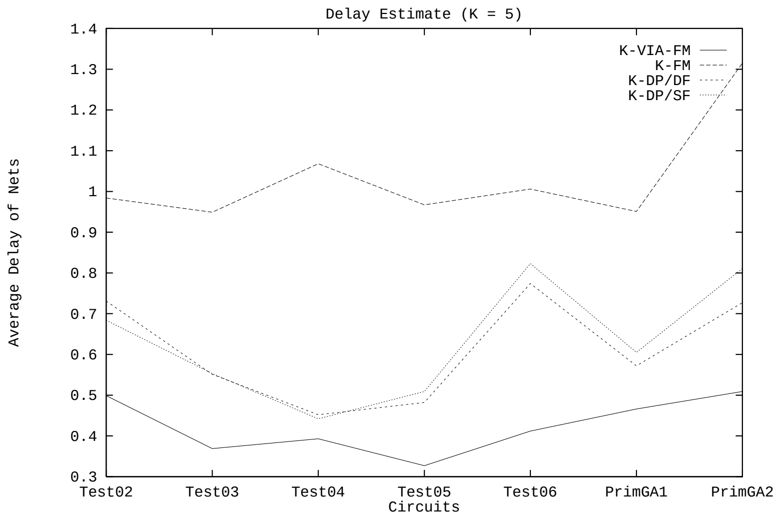

Figure 13: Comparison of Estimated Delays (K = 5).

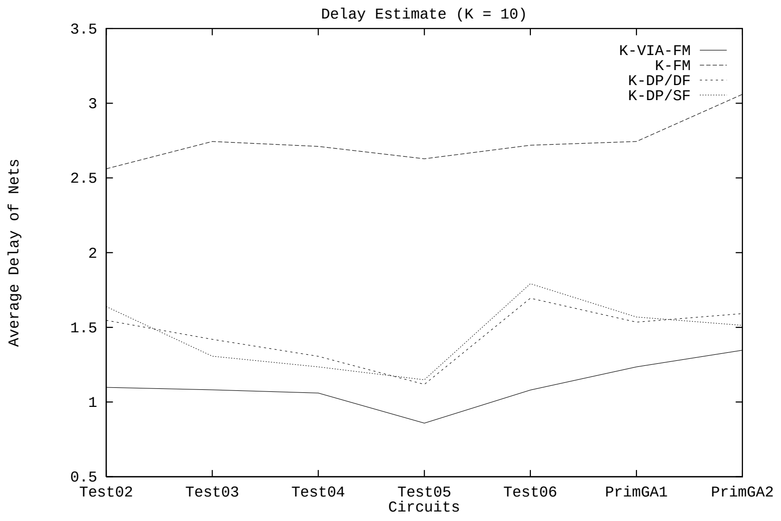

Figure 14: Comparison of Estimated Delays (K = 10).

5 Conclusion

We introduced a new natural metric, *Via-Count*, to better estimate inter-partition delays for evaluating partitioning solutions. We presented K-VIA-FM, an algorithm that iteratively improve solutions for the new metric. We compared the performance of K-VIA-FM with other algorithms, both with respect to the new metric as well as the number of nets cut. In fact, our results show that in multi-partitioning cases, minimizing the number of vias actually reduced the number of nets cut.

Future work will explore network flow based approaches to optimize the new metric. Also, we hope to use signal directionality information to incorporate a better delay estimating gain computation model into the partitioning phase. Another potential direction is to compare actual routing results when using existing partitioning algorithms against K-VIA-FM.

References

[ChSZ93] P. Chan, M. Schlag, and J. Zien, *Spectral K-Way ratio-cut partitioning and clustering*, Proc. 20th ACM/IEEE Design Automation Conference, June 1993.

[CoHK92] J. Cong, L. Hagen, and A. Kahng, *Net partitions yield better module partitions*, Proc. 29th ACM/IEEE Design Automation Conference, pp. 47-52, June 1992.

[CoLS94] J. Cong, W. Labio, and N. Shivakumar, *Multi-Way VLSI partitioning based on dual net representation*, Proc. IEEE International Conference on Computer-Aided Design, pp. 56-62, November 1994.

[DoHo73] W. Donath and A. Hoffman, *Lower bounds for partitioning of graphs*, IBM Journal of Research and Development, pp.420-425, 1963.

[FiMa82] C. Fiduccia and R. Matheyses, *A linear time heuristic for improving network partitions*, Proc. ACM/ IEEE Design Automation Conference, pp. 175-181, June 1982.

[HaKa91] L. Hagen and A. B. Kahng, *Fast spectral methods for ratio cut partitioning and clustering.* Proc. Proc. IEEE International Conference on Computer-Aided Design, pp. 10-13, June 1991.

[HaKa92] L. Hagen and A. B. Kahng, *A new approach to effective circuit clustering*, Proc. IEEE International Conference on Computer-Aided Design, pp. 422-427, November 1992.

[HaKa92b] L. Hagen and A. B.Kahng, *New spectral methods for ratio cut partitioning and clustering.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1074-1085, September 1992.

[KeLi70] B. Kernighan, and S. Lin, *An efficient heuristic for partitioning of electrical circuits*, Bell System Technical Journal, vol. 49, pp. 291-307, February 1970.

[KiGV83] S. Kirkpatrick, C. D. Gelat, and M. P. Vecchi, Jr., *Optmization by simulated annealing*, Science, vol. 220, pp. 671-680, May 1983.

[Kr84] B. Krishnamurthy, *An Improved min-cut algorithm for partitioning VLSI networks*, IEEE Transactions on Computers, vol. C-33, pp. 438-446, May 1984.

[Sa89] L. Sanchis, *Multi-Way network partitioning*, IEEE Transactions on Computers, vol. 38, no. 1, pp. 62-81, January1989.

[WeCh89] Y-C, Wei and C-K, Cheng, *Towards efficient hierarchical designs by ratio cut partitioning*, IEEE International Conference on Computer-Aided Design, pp. 298-301, November 1989.

[WeEs85] N. Weste and K. Eshraghian, *Principles of CMOS VLSI design*, Addison-Wesley, 1985.

[YaWo94] H. Yang and D. F. Wong,*Efficient Network Flow Based Min-Cut Balanced Partitioning*, Proc. IEEE International Conference on Computer-Aided Design, pp. 50-55, November 1994.

[YeCL91] C. W. Yeh, C. K. Cheng, and T. T. Lin, *A general purpose multi-way partitioning*, Proc. 28th ACM/IEEE Design Automation Conference, pp. 421-426, June 1991.

[YeCL92] C. W. Yeh, C. K. Cheng, and T. T. Lin, *A probabilistic multicommodity flow solution to circuit clustering*, Proc. IEEE International Conference on Computer-Aided Design, November 1992.