(4), since v''_{out} is ignored:

$$\begin{cases} H_{11} = \frac{f_1(z)}{g(z)} \\ H_{12} = \frac{z f_2(z)}{g(z)} \end{cases} \tag{15}$$

where

$$\begin{cases} f_1(z) = -C_B C_D C_K + C_A C_E C_K + z^2(-C_A C_B C_G \\ \quad + C_B C_D C_I + 2C_B C_D C_K - C_A C_E C_K) \\ \quad - z^4(C_B C_D C_I + C_B C_D C_K) \\ f_2(z) = C_A C_B C_H - C_B C_D C_J - C_A C_E C_K + C_A C_B C_L \\ \quad + z^2(C_B C_D C_J + C_A C_E C_K - C_A C_B C_L) \\ g(z) = C_B^2 C_D - C_A C_B C_E + z^2(C_A C_B C_C \\ \quad - 2C_B^2 C_D + C_A C_B C_E - C_B C_D C_F) \\ \quad + z^4(C_B^2 C_D + C_B C_D C_F). \end{cases} \tag{16}$$

In the bilinear SC resistor simulation all the switches change position twice for each input sample [1]. The transfer function is given by

$$\begin{aligned} H(z) &= \frac{v_{in}(z) H_{11}(z^{1/2}) + z^{-1/2} v_{in}(z) H_{12}(z^{1/2})}{v_{in}(z)} \\ &= \frac{f_1(z^{1/2}) + f_2(z^{1/2})}{g(z^{1/2})} \end{aligned} \tag{17}$$

which gives

$$\begin{aligned} H(z) = (&C_A C_H - C_D C_J - C_D C_K + C_A C_L \\ &+ z(-C_A C_G + C_D C_I + C_D C_J + 2C_D C_K - C_A C_L) \\ &- z^2(C_D C_I + C_D C_K))/(C_B C_D - C_A C_E \\ &+ z(C_A C_C - 2C_B C_D + C_A C_E - C_D C_F) + z^2(C_B C_D \\ &+ C_D C_F)). \end{aligned} \tag{18}$$

The exact analytic results produced by symbolic methods are generally difficult to interpret due to the large number of terms involved. Fortunately, we are able to approximate these expressions based on the magnitude of the individual circuit parameters. There are several methods that can be used for the approximation procedure [3], [5]. For example, CASCA truncates all terms that are some factor smaller than the largest term.

VII. Conclusion

This paper presents a method that can be used to perform time-discrete analysis using any symbolic analysis tool intended for analysis of time-continuous networks. An equivalent analog circuit is used to describe the capacitors in the SC network and the inductive transsusceptance is used to model the interaction between clock phases. Simple variable substitutions allow us to model the time-discrete behavior using only time-continuous network elements. The nullor is used to model switches and ideal operational amplifiers where each nullor reduces the rank of the compacted nodal analysis matrix by one.

References

[1] P. E. Allen and E. Sánchez-Sinencio, *Switched Capacitor Circuits*. New York: Van Nostrand Reinhold, 1984.

[2] P. E. Fleischer and K. R. Laker, "A family of active switched capacitor biquad building blocks," *Bell Syst. Tech. J.*, vol. 58, no. 10, pp. 2235–2269, Dec. 1979.

[3] H. Floberg, "Computer aided symbolic circuit analysis," Thesis, Dept. Appl. Electron., Lund University, Lund, Sweden, Dec. 1992.

[4] H. Floberg, "CASCA" Tutorial, Dept. Appl. Electron., Lund University, Lund, Sweden, 1994.

[5] G. Gielen, H. Walscharts, and W. Sansen, "ISAAC: A symbolic simulator for analog integrated circuits," *IEEE J. Solid-State Circuits*, pp. 1587–1597, Dec. 1989.

[6] P.-M. Lin, *Symbolic Network Analysis*. Amsterdam, The Netherlands: Elsevier, 1991.

[7] C. F. Kurth and G. S. Moschytz, "Nodal analysis of switched-capacitor networks," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 93–105, Feb. 1979.

[8] ——, "Two-port analysis of switched-capacitor networks using four-port equivalent circuits in the z-domain," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 166–180, Mar. 1979.

[9] E. Hökenek and G. S. Moschytz, "Analysis of general switched-capacitor networks using indefinite admittance matrix," *Proc. Inst. Elect. Eng.*, vol. 127, pp. 21–33, Feb. 1980.

On Implementation Choices for Iterative Improvement Partitioning Algorithms

Lars W. Hagen, Dennis J.-H. Huang, and Andrew B. Kahng

***Abstract*—Iterative improvement partitioning algorithms such as the FM algorithm of Fiduccia and Mattheyses [8], the algorithm of Krishnamurthy [13], and Sanchis's extensions of these algorithms to multiway partitioning [16] all rely on efficient data structures to select the modules to be moved from one partition to the other. The implementation choices for one of these data structures, the *gain bucket*, is investigated. Surprisingly, selection from gain buckets maintained as last-in-first-out (LIFO) stacks leads to significantly better results than gain buckets maintained randomly (as in previous studies of the FM algorithm [13], [16]) or as first-in-first-out (FIFO) queues. In particular, LIFO buckets result in a 36% improvement over random buckets and a 43% improvement over FIFO buckets for minimum-cut bisection. Eliminating randomization from the bucket selection not only improves the solution quality, but has a greater impact on FM performance than adding the Krishnamurthy gain vector. The LIFO selection scheme also results in improvement over random schemes for multiway partitioning [16] and for more sophisticated partitioning strategies such as the two-phase FM methodology [2]. Finally, by combining insights from the LIFO gain buckets with the Krishnamurthy higher-level gain formulation, a new higher-level gain formulation is proposed. This alternative formulation results in a further 22% reduction in the average cut cost when compared directly to the Krishnamurthy formulation for higher-level gains, assuming LIFO organization for the gain buckets.**

***Index Terms*—Fiduccia–Mattheyses algorithm, gain bucket implementation, hypergraph partitioning, iterative movement, Kernighan–Lin algorithm, multiway partitioning, VLSI netlist partitioning.**

I. Introduction

In production software for circuit partitioning, iterative improvement is a nearly universal approach, either as a postprocessing refinement to other methods or as a method in itself. Iterative

Manuscript received January 26, 1995; revised September 22, 1995. This work was supported in part by NSF Grant MIP-9257982 and matching funds from High-Level Design Systems. This paper was recommended by Associate Editor C.-K. Cheng.

L. Hagen is with Cadence Design Systems, Inc., San Jose, CA 95134 USA (e-mail: lars@cadence.com).

D. Huang is with AvanWise, Inc., Fremont, CA 94538 USA (e-mail: dhuang@avanticorp.com).

A. Kahng is with the Department of Computer Science, University of California, Los Angeles, CA 90095-1596 USA.

Publisher Item Identifier S 0278-0070(97)09243-9.

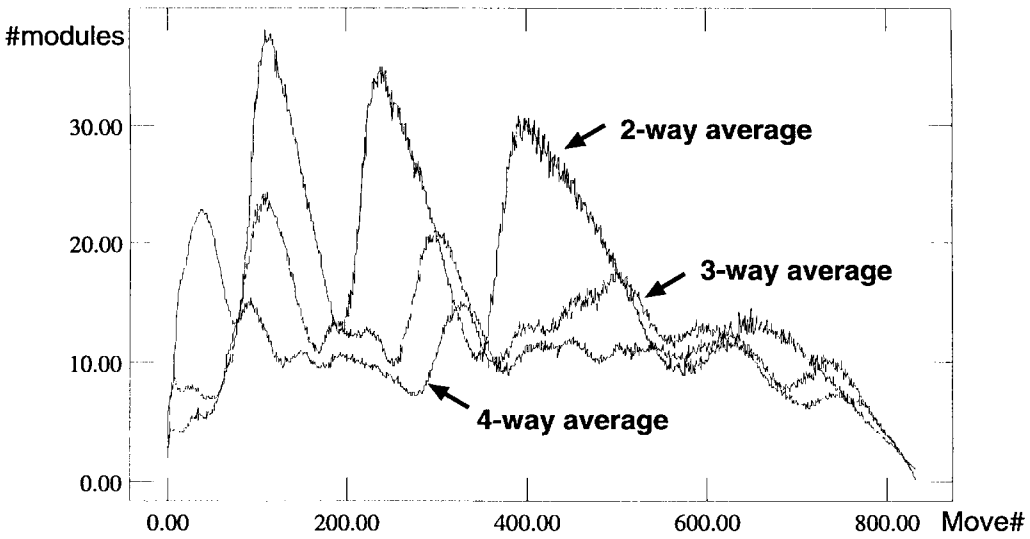

Fig. 1. Average number of modules in the highest-gain bucket at each move during the first pass of FM for two, three, and four-way balanced partitioning for test case Primary1 (833 modules). The average numbers were generated from 1000 separate FM runs.

improvement partitioning algorithms are typically variants of the Kernighan-Lin method (KL) [11], [18] or its algorithmic speedup by Fiduccia and Mattheyses (FM) [8]. Examples of more recent methods are due to Krishnamurthy [13], Sanchis [16], Dutt [7], Hoffman [10], Dasdan and Aykanat [6], and Saab [15]. Iterative improvement algorithms such as KL and FM start with a current feasible solution and iteratively perturb it into another feasible solution, adopting the perturbation as the next solution only if it improves the cost function. The type of perturbation (or "move") used determines a topology over the set of feasible solutions, known as a *neighborhood structure*. For the cost function to be "smooth" over the neighborhood structure, the perturbation (also known as a *neighborhood operator*) should be small and "local." In netlist partitioning, the cost function is typically the number of nets cut; a given move has *gain* corresponding to the decrease in cut nets that would result from the move.

Over the past decade, FM has become perhaps the single most widely used and cited partitioning algorithm in the very large scale integration (VLSI) CAD area. The primary difference between the KL and FM algorithms lies in their respective neighborhood operators. KL iteratively makes a highest-gain swap of a pair of modules between two partitions; FM iteratively makes a highest-gain shift of a single module from one partition to another.[1] This subtle difference allows FM to achieve significant improvement in runtime with little loss in solution quality. FM amortizes the cost of updating the module gains, such that the total cost of finding the highest-gain module is $O(p)$ per pass, where p is the total number of pins. The enabling data structure is an array of "gain buckets" which groups the modules of a given partition according to their gains.

Many works have investigated possible improvements and extensions to the FM algorithm. One often cited extension is that of Krishnamurthy [13], which introduces efficient "look-ahead" into the FM algorithm to improve tie-breaking when the highest-gain bucket contains more than one module. Specifically, Krishnamurthy extends the gain value of a module into a gain *vector* which stores a sequence of potential gain values corresponding to sets of future moves. Given a U/W partition, Krishnamurthy defines the *binding number* $\beta_U(s)$ of signal net s with respect to partition U to be the number of unlocked modules of s in partition U, unless there is a locked module of s in partition U, in which case $\beta_U(s) = \infty$. Intuitively, the binding number $\beta_U(s)$ is a measure of how difficult it is to move net s out of partition U. The binding number $\beta_W(s)$ is similarly defined. The kth-level gain $\gamma_k(v_i)$ of module $v_i \in U$ is then given by[2]

$$\gamma_k(v_i) = |\{s \in E \mid v_i \in s, \beta_U(s) = k, \beta_W(s) > 0\}| - |\{s \in E \mid v_i \in s, \beta_U(s) > 0, \beta_W(s) = k-1\}|.$$

Each element $\gamma_k(v_i)$ in the gain vector corresponds to the kth-level gain of module v_i. Note that the first-level gain $\gamma_1(v_i)$ corresponds to the gain used in the FM algorithm.

Intuitively, the positive term (i.e., first term) in the formula for γ_k counts nets with binding number $k-1$ that are "created" (for the partition that the module is moving "from") by the move, while the negative term (i.e., second term) counts nets with binding number $k-1$ that are "destroyed" (for the partition that the module is moving "to") by the move. Krishnamurthy's method uses lexicographic ordering of the vectors $(\gamma_1, \gamma_2, \gamma_3, \cdots)$ to break ties when an FM gain bucket contains more than one module. Krishnamurthy compared his FM plus higher-level gain (FM+HL) algorithm with the original FM algorithm and found that adding second- and third-level gains improved the average solution quality with only $O(kp)$ added computational expense, where k is the number of values maintained in (i.e., the size of) the gain vector. This was confirmed by Sanchis [16], who extended FM+HL (and thus implicitly FM as well) to multiway partitioning.

II. Tie-Breaking in the Fiduccia–Mattheyses Algorithm

During a typical pass of FM, there are usually many ties (i.e., the highest-gain bucket will contain more than one module). Fig. 1 shows the number of modules in the highest-gain bucket at each

[1] A *pass* of each algorithm generates such moves until every module has been moved exactly once, then adopts the prefix of this move sequence with highest total gain. When a pass results in zero gain, the algorithm terminates.

[2] The notation used for the Krishnamurthy formulas are adapted from [13]. Note that in order to handle 1-pin nets correctly, the term $\beta_U(s) > 0$ should be changed to $\beta_U(s) > 1$. However, 1-pin nets can also be eliminated while reading in the netlist, obviating the need for such a change.

TABLE I
AVERAGE CUTSIZE RESULTS FOR 100 RUNS OF FM (COLUMN 3) AND KRISHNAMURTHY HIGHER-LEVEL GAINS (COLUMNS 4-6) USING LIFO, RANDOM, AND FIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE NUMBERS IN PARENTHESES GIVE THE MINIMUM CUTSIZES OBSERVED OVER 100 RUNS

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	757.09(401)	743.57(453)	774.79(407)	760.47(439)
	random	1670.58(1142)	1299.16(834)	1108.28(665)	977.26(586)
	FIFO	1765.94(1432)	1342.30(1113)	1188.18(835)	1020.79(554)
industry3 (15433)	LIFO	755.27(315)	798.68(361)	796.81(325)	814.60(338)
	random	2381.63(1381)	1163.55(531)	1011.66(424)	952.48(344)
	FIFO	3285.25(1921)	1452.46(598)	1151.14(548)	1154.22(456)
avq.small (21918)	LIFO	834.92(542)	910.91(675)	950.31(577)	963.52(627)
	random	1507.03(1335)	1403.37(1277)	1383.65(1240)	1387.77(1223)
	FIFO	1841.80(1681)	1578.77(1441)	1498.67(1318)	1489.15(1279)
avq.large (25178)	LIFO	1140.22(816)	970.67(598)	1011.70(545)	1030.13(459)
	random	1836.89(1649)	1582.17(1416)	1527.13(1383)	1510.19(1328)
	FIFO	2192.99(2058)	1867.08(1734)	1767.62(1604)	1741.74(1609)
Average Improv. vs. random	LIFO	51.37%	36.97%	29.10%	24.75%
	random	0	0	0	0
	FIFO	-21.31%	-14.66%	-11.26%	-12.07%

move throughout the first pass of FM for the Primary 1 test case in two-way, three-way, and four-way balanced partitioning (we plot the average over 1000 runs). Note that on average there are more modules in the highest-gain bucket during two-way partitioning than during three-way or four-way partitioning.

From the figure accompanying the algorithm description in [8] one can infer that the Fiduccia and Mattheyses gain buckets function as last-in-first-out (LIFO) stacks (remove at head, insert at head). However, the gain buckets could just as easily function as first-in-first-out (FIFO) queues (remove at head, insert at tail) while supporting the same algorithmic complexity. Neither Krishnamurthy nor Sanchis points out any tie-breaking schemes for cells with identical gains. In fact, both used randomized selection in case of ties.

In the following section we compare LIFO selection with random selection (used by Sanchis [16]) and FIFO selection (an alternative organization which as far as we know has never been used before). Our testbed is the code distributed by Sanchis [17] with appropriate modifications made for handling LIFO and FIFO selection. In all of our experiments, we assume the modules have unit area and constrain the partition sizes to differ by at most one.

III. EXPERIMENTAL RESULTS FOR MIN CUT BISECTION

Our first experiment compares tie-breaking schemes in (two-way) minimum-cut bisection. Table I gives the average and minimum cutsizes for 100 FM runs using the three selection schemes (LIFO, random, and FIFO). The "Pure FM" column of Table I clearly shows the effects of the selection methodology. Surprisingly, the FIFO scheme is much worse than random selection. The LIFO scheme, on the other hand, gives considerable improvement over random selection. An explanation for this improvement may be that organizing the buckets such that the "most recently visited" modules are placed near the beginning of the buckets implicitly causes neighborhoods or clusters of modules to be moved together. Furthermore, since there are two bucket gain structures, one for each partition, it is possible for each partition to "pull" on different clusters while maintaining the balance. If these clusters are noninterfering, i.e., widely separated, more of the early moves will result in positive gain, enabling the current pass to reach a lower-cost point in the solution space. In other words, within each pass the solution cost curve will have a relatively sharper decline, and stay at lower costs as it returns back to the initial cost.[3]

Columns 4–6 of Table I show the effects of LIFO, random, and FIFO selection schemes on the Krishnamurthy higher-level gains [13]. Introducing second-level ($k = 2$) and in some cases third-level ($k = 3$) gain seems to improve the solution quality for random and FIFO selection schemes. With regard to LIFO selection, we note the following.

- For constant k, the LIFO results are consistently better than the random or FIFO results.
- For each of the test cases, the $k = 1$ (FM) results using LIFO selection are significantly better than the results for any k using random or FIFO selection. In other words, the gain bucket organization has a greater effect on solution quality than the number of gain elements considered.
- For test cases industry3 and avq.small, the $k = 1$ (FM) results are better than the $k > 1$ results under the LIFO scheme. Recall that the Krishnamurthy gain formula favors a module in a net that is locked to the side the module is moving to, and disfavors a module in an unlocked net having few modules on the side the module is moving to. In some sense, the LIFO organization has a similar function but with no penalty for moving a module that belongs to unlocked nets. That Krishnamurthy gains occasionally perform worse than LIFO FM suggests that following previously moved modules (i.e., moving to the side to which a net is locked) is more important than "staying away from the minority" (i.e., not moving to the side having very few modules of the incident nets).

IV. EXPERIMENTAL RESULTS FOR MULTIWAY PARTITIONING

We have also tested the LIFO, random, and FIFO selection schemes for three-way and four-way balanced partitioning using Sanchis's [16] extension to multiway partitioning of the FM algorithm and Krishnamurthy's higher-level gains. Tables II and III give the average and minimum cutsizes over 50 FM runs using the three selection schemes. We measure cutsize as the number of nets cut by the partitioning. The results show that although LIFO gives

[3] In bipartitioning, the cost at the end of the pass is exactly the same as the cost at the beginning of the pass.

TABLE II
AVERAGE CUTSIZE RESULTS FOR 50 RUNS OF THREE-WAY FM AND KRISHNAMURTHY HIGHER-LEVEL GAINS (COLUMNS 4-6) USING LIFO, RANDOM, AND FIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE NUMBERS IN PARENTHESES GIVE THE MINIMUM OBSERVED CUTSIZES

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	2316.12(2020)	2126.39(1873)	2103.59(1760)	2083.08(1517)
	random	2482.34(2308)	2182.02(1876)	2119.28(1761)	2124.90(1585)
	FIFO	2554.18(2346)	2159.87(1815)	2124.80(1739)	2136.78(1740)
industry3 (15433)	LIFO	3891.39(2718)	3592.73(2852)	3465.39(2624)	3449.68(2102)
	random	4393.32(4044)	3679.82(2896)	3584.32(2869)	3587.62(2789)
	FIFO	4574.50(3993)	3793.00(2859)	3567.18(2827)	3655.98(3103)
avq.small (21918)	LIFO	2634.36(2434)	2498.24(2374)	2499.12(2343)	2483.48(2331)
	random	2707.12(2562)	2572.88(2448)	2527.56(2390)	2516.36(2375)
	FIFO	2846.60(2659)	2595.74(2435)	2537.12(2416)	2536.82(2364)
avq.large (25178)	LIFO	3000.82(2876)	2721.34(2610)	2702.34(2599)	2709.92(2581)
	random	3034.54(2902)	2776.88(2651)	2733.76(2560)	2738.39(2616)
	FIFO	3132.34(2988)	2801.14(2703)	2742.04(2593)	2733.66(2618)
Average Improv. vs. random	LIFO	5.48%	2.45%	1.58%	2.04%
	random	0	0	0	0
	FIFO	-3.85%	-0.96%	-0.12%	-0.78%

TABLE III
AVERAGE CUTSIZE RESULTS FOR 50 RUNS OF FOUR-WAY FM AND KRISHNAMURTHY HIGHER-LEVEL GAINS (COLUMNS 4–6) USING LIFO, RANDOM, AND FIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE NUMBERS IN PARENTHESES GIVE THE MINIMUM OBSERVED CUTSIZES

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2 (12637)	LIFO	2624.50(2346)	2441.78(2193)	2415.48(2191)	2387.90(1905)
	random	2778.54(2605)	2489.92(2254)	2441.16(2185)	2402.74(2047)
	FIFO	2829.68(2643)	2448.56(2294)	2415.08(2149)	2399.70(2136)
industry3 (15433)	LIFO	4968.88(4506)	4469.78(3575)	4366.54(3437)	4312.96(3594)
	random	5260.00(4885)	4497.02(3780)	4445.40(3676)	4311.02(3559)
	FIFO	5529.56(5196)	4575.36(4197)	4358.54(3590)	4352.42(3520)
avq.small (21918)	LIFO	3247.94(3055)	2921.18(2755)	2900.58(2734)	2889.34(2707)
	random	3443.82(3241)	2986.47(2889)	2920.42(2782)	2940.38(2817)
	FIFO	3597.83(3386)	3014.98(2877)	2924.66(2817)	2917.80(2776)
avq.large (25178)	LIFO	3573.78(3402)	3223.38(3050)	3154.78(3024)	3166.92(3070)
	random	3771.10(3614)	3261.08(3114)	3175.18(3042)	3167.34(3002)
	FIFO	3909.34(3660)	3298.88(3094)	3169.58(3012)	3130.94(3018)
Average Improv. vs. random	LIFO	5.50%	1.47%	1.04%	0.58%
	random	0	0	0	0
	FIFO	-3.78%	-0.55%	0.76%	0.27%

TABLE IV
AVERAGE CUTSIZE RESULTS FOR 100 RUNS OF TWO-PHASE FM FOR MINIMUM-CUT BISECTION USING LIFO, RANDOM, AND FIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE NUMBERS IN PARENTHESES GIVE THE MINIMUM OBSERVED CUTSIZES

Benchmark	Ave. (min) cutsize			Improvement of
	LIFO	random	FIFO	LIFO vs. random
industry2	419.16(260)	453.69(301)	454.21(304)	8%
industry3	564.71(292)	630.72(327)	643.31(312)	10%
avq.small	408.79(206)	480.04(275)	499.11(297)	15%
avq.large	645.63(415)	745.35(495)	768.35(519)	13%

better results than both random and FIFO, the overall gain is much less than was observed for two-way partitioning. Furthermore, the Krishnamurthy higher-level gains appear to play a bigger part for multiway partitioning than for two-way partitioning.

Notice that there is much less variability in the multiway partitioning results, i.e., the minimum is relatively close in value to the average. One explanation for the lack of solution variance is that having on average fewer modules in the highest-gain bucket (recall Fig. 1) may reduce the importance of tie breaking. Another observation, which may help explain why the Krishnamurthy gain vector is more important for multiway partitioning, is that there are now several possible destination partitions for a given module; thus, the direction in which a module may be "pulled" by previously moved modules is no longer unique.

TABLE V
RESULTS COMPARING OUR NEW MULTILEVEL GAIN FORMULATION WITH KRISHNAMURTHY'S MULTILEVEL GAIN FORMULATION USING THE LIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE AVERAGES ARE BASED ON 100 RUNS; NUMBERS IN PARENTHESES GIVE THE MINIMUM OBSERVED CUTSIZES

Benchmark (#nodes)	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2	Ours	757.09(401)	618.82(334)	623.16(333)	622.62(275)
(12637)	Krishnamurthy	757.09(401)	743.57(453)	774.79(407)	760.47(439)
industry3	Ours	755.27(315)	702.92(300)	721.96(349)	710.97(312)
(15433)	Krishnamurthy	755.27(315)	798.68(361)	796.81(325)	814.60(338)
avq.small	Ours	834.92(542)	630.81(398)	652.72(416)	657.84(373)
(21918)	Krishnamurthy	834.92(542)	910.91(675)	950.31(577)	963.52(627)
avq.large	Ours	1140.22(816)	683.85(457)	752.06(475)	699.59(406)
(25178)	Krishnamurthy	1140.22(816)	970.67(598)	1011.70(545)	1030.13(459)
Average Improvement		0	22.27%	21.49%	23.67%

TABLE VI
AVERAGE CUTSIZE RESULTS FOR 100 RUNS OF OUR NEW MULTILEVEL GAIN FORMULATION (COLUMNS 4–6) USING LIFO, RANDOM, AND FIFO ORGANIZATION SCHEMES FOR THE GAIN BUCKETS. THE NUMBERS IN PARENTHESES GIVE THE MINIMUM OBSERVED CUTSIZES

Benchmark	Method	Pure FM Ave.(Min)	2nd-level Gain Ave.(Min)	3rd-level Gain Ave.(Min)	4th-level Gain Ave.(Min)
industry2	LIFO	757.09(401)	618.82(334)	623.16(333)	622.62(275)
(12637)	random	1670.58(1142)	808.59(354)	682.66(327)	691.73(292)
	FIFO	1765.94(1432)	786.33(337)	722.07(324)	667.65(348)
industry3	LIFO	755.27(315)	702.92(300)	721.96(349)	710.97(312)
(15433)	random	2381.63(1381)	739.81(336)	706.63(307)	679.79(348)
	FIFO	3285.25(1921)	749.90(303)	712.04(328)	742.52(328)
avq.small	LIFO	834.92(542)	630.81(398)	652.72(416)	657.84(373)
(21918)	random	1507.03(1335)	785.81(641)	799.01(584)	780.41(591)
	FIFO	1841.80(1681)	862.82(663)	823.65(573)	814.61(634)
avq.large	LIFO	1140.22(816)	683.85(457)	752.06(475)	699.59(406)
(25178)	random	1836.89(1649)	964.04(769)	1034.63(848)	951.25(802)
	FIFO	2192.99(2058)	1114.02(935)	1135.28(884)	1053.74(840)
Average Improv.	LIFO	51.37%	19.31%	13.04%	11.89%
vs. random	random	0	0	0	0
	FIFO	-21.31%	-5.99%	-4.84%	-5.23%

V. EXPERIMENTAL RESULTS FOR TWO-PHASE FM

Our third set of experiments tests the LIFO, random, and FIFO selection schemes within the so-called two-phase FM approach. Two-phase FM [3], [9], which has gained attention recently due to much better results than "single-phase" FM, is essentially the state of the art in iterative partitioning (see, e.g., [5]). The method generates a partition by running two FM phases on the netlist. In phase I, a clustering of the netlist is constructed and FM is run on this clustered instance, after which phase II uses the "flattened" solution from phase I as the starting solution for running FM on the original netlist.[4] Table IV gives the LIFO, random, and FIFO results for two-phase FM using the recent WINDOW clustering of [2].[5] As in Table I, there is a noticeable difference among the three selection schemes. LIFO selection is consistently better than either random or FIFO selection, with average improvement of LIFO over random being 12%. Furthermore, the LIFO two-phase FM results are 41% better than the LIFO results for single-phase FM. These results demonstrate that the bucket organization also plays a part within sophisticated partitioning approaches.

VI. A KRISHNAMURTHY VARIANT FOR MIN CUT BISECTION

The observation in Section III, that it may be more important to move modules which are incident to locked nets, suggests an alternative multilevel gain formulation for two-way partitioning. If a net is cut, and only one partition contains locked modules incident to this net, higher priority in moving should be given to the modules in the partition having no locked modules incident to this net. Such an objective can be achieved by increasing the gain elements of a module each time it is incident to a net which becomes locked to the opposite partition. For instance, assume module a is being evaluated for a move from partition U to partition W. If a net which contains module a has at least one module locked in partition W, and only free modules in partition U, we will increase all kth-level gains by 1, for $k \geq 2$. We avoid changing the first-level gain since it should

[4] The balance constraint of the FM algorithm on the clustered netlist is set to half the total area $\pm$ the size of the largest cluster. This "relaxation" of the balance constraint is necessary to allow FM to find a good partitioning solution of the clustered netlist. In phase II, which uses the clustered netlist solution as the starting point, an initial set of greedy moves is performed to get a solution satisfying the balance constraints for bisection (half the total area ± 1).

[5] The WINDOW clustering algorithm first generates a linear ordering of the modules according to a specific "attraction function" and then uses dynamic programming to split the linear ordering optimally based on the clustering objective function. The orderings used correspond to the "scaled cost" (generalized ratio-cut) metric proposed by Chan *et al.* [4].

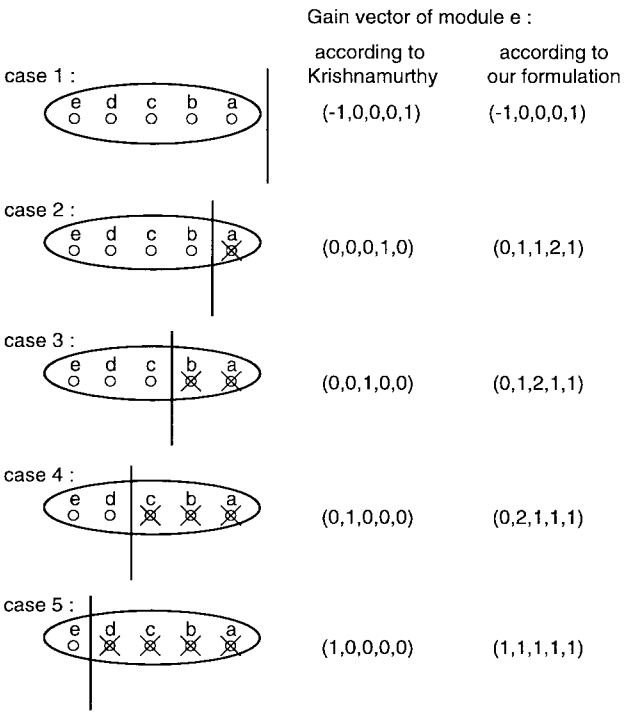

Fig. 2. Evolution of the gain vector for module e in a five-pin net (a, b, c, d, e) according to the Krishnamurthy level gain formulation and our new gain formulation.

always reflect the "actual" gain resulting from a move of this module. However, we add one to all the other gain levels so that the increased priority is guaranteed to affect the tie-breaking.

Our alternative gain formulation can be expressed as follows for $k \geq 2$:

$$\begin{aligned}\gamma_k(v_i) = &|\{s \in E \mid v_i \in s, \beta_U(s) = k, \beta_W(s) > 0\}| \\ &- |\{s \in E \mid v_i \in s, \beta_U(s) > 0, \beta_W(s) = k-1\}| \\ &+ |\{s \in E \mid v_i \in s, 0 < \beta_U(s) < \infty, \beta_W(s) = \infty\}|.\end{aligned}$$

The first two terms are identical to Krishnamurthy's formulation [13]. The third term is new and represents the "attraction" to locked modules. Fig. 2 contrasts the evolution of Krishnamurthy's gain vector against that of the gain vector resulting from our new formulation. Initially, an uncut net contains modules a, b, c, d, and e and both gain vectors for module e are (-1, 0, 0, 0, 1). After module a is moved to the other partition and becomes locked, the gain vector of module e is changed to (0, 0, 0, 1, 0) in Krishnamurthy's formulation, but is changed to (0, 1, 1, 2, 1) in our formulation. When module e is the only remaining module (case 5), the gain vectors are (1, 0, 0, 0, 0) and (1, 1, 1, 1, 1) for Krishnamurthy's and our formulation, respectively. Note that in this last case, the Krishnamurthy gain vector will not distinguish between module e and some other module x having gain vector (1, 0, 0, 0, 0), where none of the nets incident to x have locked modules. Again, this may be an important difference since one would seemingly prefer to "uncut" the locked net incident to module e before committing the unlocked net incident to module x. Our experimental results also seem to support this view.

We tested our new gain formulation using the same LIFO, random, and FIFO selection schemes described in Section II. Table V compares LIFO results using our new formulation against LIFO results using the original Krishnamurthy formulation. The third column (pure FM) results are identical since our new formulation does not affect the first-level gain. Overall, we achieve 22% improvement over Krishnamurthy's formulation; in some cases (e.g., industry2 and avq.small) our formulation leads to substantial reduction in the size of the minimum cuts found.

Table VI shows the LIFO, random, and FIFO results for our new gain formulation. Just as with the Krishnamurthy formulation, the results using a LIFO selection scheme with our new formulation are significantly better than the results using random or FIFO selection schemes. Notice, however, that the second-level gain results (column 4) using random and FIFO selection schemes show significant improvement over the pure FM results (column 3) with our new formulation. This is in sharp contrast to the results using the Krishnamurthy formulation, which did not show much improvement with higher-level gains using either random or FIFO selection. An explanation for this might be that with our new formulation, the higher-level gains are computed more carefully and tend to obviate the need for a "good" selection scheme (i.e., the results for random and FIFO will more closely mirror the results of LIFO as the length of the gain vectors increases). Also, our new formulation explicitly gives higher priority to the neighbors of moved modules, which is similar to the effect of the LIFO selection scheme.

VII. Conclusion

We have shown that implementation choices play an important part for both the FM algorithm of Fiduccia and Mattheyses [8] and the algorithm of Krishnamurthy [13]. In particular, selection from gain buckets based on the implicit ordering of a linked list representation is advantageous and will result in improved partitioning solutions. Eliminating randomization from the bucket selection not only improves the solution quality, but has a greater impact on FM performance than adding the Krishnamurthy gain vector. This reopens the question of interpreting such seminal works in the literature as [16] and [13], whose studies used random bucket selection. Organizing the gain buckets as LIFO stacks leads to a 36% improvement versus random bucket organization and a 43% improvement versus FIFO queues. We have also presented an alternative higher-level gain formulation, based on Krishnamurthy's approach, which incorporates some of the intuition behind the LIFO organization. This alternative formulation results in a further 22% reduction in the average cut cost when compared directly to the Krishnamurthy formulation for higher-level gains, assuming LIFO organization for the gain buckets.

We believe that a more detailed study is necessary to better understand the effect of choices in the FM implementation on the solution quality and runtime. Thus, our future work investigates not only further tie-breaking mechanisms, but also interesting effects that result from the order imposed by the netlist representation and the list of free modules.[6]

References

[1] C. J. Alpert and A. B. Kahng, "Geometric embeddings for faster and better multi-way netlist partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 743–748.

[2] ——, "A general framework for vertex orderings, with applications to netlist clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1994, pp. 63–67.

[3] T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms," in *Proc. ACM/IEEE Design Automation Conf.*, 1989, pp. 775–778.

[4] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "Spectral K-way ratio-cut partitioning and clustering," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1088–1096, Sept. 1994.

[6] The input format of a netlist is typically a function of how the other development tools represent and output the circuit, and may group related nets or modules together or far apart. This relatedness/unrelatedness will in turn be reflected within the data structures used by FM to store the netlist information.

[5] C.-K. Cheng and Y.-C. A. Wei, "An improved two-way partitioning algorithm with stable performance," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1502–1511, 1991.
[6] A. Dasdan and C. Aykanat, "Improved multiple-way circuit partitioning algorithms," in *Proc. ACM/SIGDA Int. Workshop Field-Programmable Gate Arrays*, 1994.
[7] S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1993, pp. 370–377.
[8] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[9] L. Hagen and A. B. Kahng, "A new approach to effective circuit clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1992, pp. 422–427.
[10] A. G. Hoffman, "The dynamic locking heuristic—A new graph partitioning algorithm," in *Proc. IEEE Int. Symp. Circuits Systems*, 1994, pp. 173–176.
[11] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[12] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
[13] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. 33, pp. 438–446, 1984.
[14] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York: Wiley-Teubner, 1990.
[15] Y. Saab, "A fast and robust network bisection algorithm," *IEEE Trans. Comput.*, vol. 44, 1995.
[16] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62–81, 1989.
[17] ——, private communication, Mar. 1994.
[18] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. ACM/IEEE Design Automation Conf.*, 1972, pp. 57–62.
[19] C. Sechen, "Placement and global routing of integrated circuits using simulated annealing," Ph.D. dissertation, Univ. California, Berkeley, 1986.
[20] Y.-C. Wei and C.-K. Cheng, "Toward efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 298–301.

Delay Abstraction in Combinational Logic Circuits

Noriya Kobayashi and Sharad Malik

***Abstract*—In this paper we propose a data structure for abstracting the delay information of a combinatorial circuit. The particular abstraction that we are interested in is one that preserves the delays between all pairs of inputs and outputs in the circuit. Such abstractions are useful when considering the delay of cascaded circuits in high-level synthesis and other such applications in synthesis. The proposed graphical data structure is called the *concise delay network*, and is of size proportional to $(m + n)$ in best case, where m and n refer to the number of inputs and outputs of the circuit. In comparison, a delay matrix that stores the maximum delay between each input–output pair has size proportional to $m \times n$. For circuits with hundreds of inputs and outputs, this storage and the associated computations become quite expensive, especially when they need to be done repeatedly during synthesis.**

We present heuristic algorithms for deriving these concise delay networks. Experimental results shows that, in practice, we can obtain concise delay network with the number of edges being a small multiple of $(m+n)$.

***Index Terms*—Combinational logic circuits, data structures, delay estimation, directed graphs.**

I. Introduction

In this paper we propose a data structure for abstracting the delay information of a combinatorial circuit. The particular abstraction that we are interested in is one that preserves the delays between all pairs of inputs and outputs in the circuit. There are several applications of such an abstraction.

- Consider the problem of determining the delay of a pair of cascaded operation units in high-level synthesis. (Such a cascade is also referred to as *operator chaining*.) The worst case delay of the cascade is not necessarily the sum of the worst case delays of the individual units. This is because the critical paths in the two units need not be concatenated in the cascade. However, if the delays between all pairs of inputs and outputs of the original units are known, then this information can be used to derive the correct worst case delay.
- Consider the case in logical and physical synthesis when only one module is modified and the change in the delay needs to be propagated through the entire design. A complete pass through the design may be avoided, if for each combinational block we can make available the delay between each input and output pair.

One such delay abstraction is a *delay matrix* that stores the delays for each input-output pair for each combinational block. This requires large memory space since it has $m \times n$ entries, where m and n are the number of input and output terminals of a circuit. All entries of a delay matrix have to be referred to during delay computation. Large matrices make the delay computation task slower. Another alternative is to use a network with the same topology as the original circuit. Such a network is typically quite large, and it makes the delay computation task much slower.

In [1], delay matrices are used as the timing model for high-level synthesis. In [2], bipartite graphs equivalent to delay matrices are

Manuscript received November 9, 1995; revised May 7, 1997. This paper was recommended by Associate Editor K. Sakallah.

N. Kobayashi is with C&C Media Research Laboratories, NEC Corporation, Kawasaki 216, Japan (e-mail: noriya@ccm.cl.nec.co.jp).

S. Malik is with Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: sharad@ee.princeton.edu).

Publisher Item Identifier S 0278-0070(97)09233-6.