A Graph Partitioning Problem for Multiple-Chip Design *

Yao-Ping Chen, Ting-Chi Wang and D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

In this paper, we introduce a new graph partitioning problem that stems from a multiple-chip design style in which there is a chip library of chips containing predesigned circuit components (e.g. adders, multipliers etc) which are frequently used. Given an arbitrary circuit data flow graph, we have to realize the circuit by appropriately choosing a set of chips from the chip library. In selecting chips from the chip library to realize a given circuit, both the number of chips used and the interconnection cost are to be minimized. Our new graph partitioning problem models this chip selection problem. We present an efficient solution to this problem.

1 Introduction

The graph partitioning problem that we consider in this paper stems from a multiple-chip design style at GE as described in [6]. In this design environment, there is a chip library of chips containing predesigned circuit components (e.g., adders, multipliers etc) which are frequently used. Given an arbitrary circuit data flow graph, we have to realize the circuit by appropriately choosing a set of chips from the chip library. The chips selected will then be placed on a substrate and interconnected together. In selecting chips from the chip library to realize a given circuit, two goals are considered to reduce cost. First, the number of chips to be used is as small as possible. Second, the total length of interconnections across chip boundaries (i.e., the external interconnection cost) is minimized. This problem is similar to the multiple-way graph partitioning problem [1, 2, 3, 4, 5] except that some constraints are added.

We now describe the new graph partitioning problem. Given an undirected weighted graph $G = (V, E)$, let W_{uv} be the weight of edge $(u, v) \in E$, and C be a finite set of colors. The vertices of G are colored as given by a function $\alpha : V \rightarrow C$ where $\alpha(v)$ is the color of v. Let $\Omega = \{M_j | 1 \leq j \leq m\}$ where each M_j is a multiset with elements in C. Let $\Pi = \{P_1, \ldots, P_K\}$ be a partitioning of V, i.e., P_i's are disjoint subsets of V and $\bigcup_{i=1}^{K} P_i = V$. Let the multiset C_i be $\{\alpha(v) | v \in P_i\}$. Π is said to be a *legal* partitioning if for each i , there exists j such that $C_i \subseteq M_j$. In this case, we say P_i is of type M_j. We define the interconnection cost $\chi(\Pi)$ of a partitioning Π as the sum of W_{uv} over all the edge $(u, v) \in E$ such that u and v are in different P_i's in Π. The objective of our graph partitioning problem is to find a legal partitioning Π of V such that both $|\Pi|$ and $\chi(\Pi)$ are minimized.

The new graph partitioning problem models the chip selection problem as follows. The graph G corresponds to the circuit. The set of colors C corresponds to circuit components. Each multiset $M_j = \{c_1, \ldots, c_l\}$ corresponds to one type of chip in the chip library, and the c_i's are the components on chip M_j. Thus Ω is the chip library. The color of v (i.e. $\alpha(v)$) is the component type (e.g., adder, multiplier). $C_i \subseteq M_j$ means that the subcircuit P_i can be implemented by a chip of type M_j.

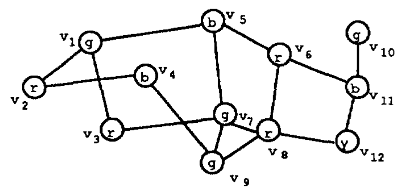

Figure 1: A colored graph.

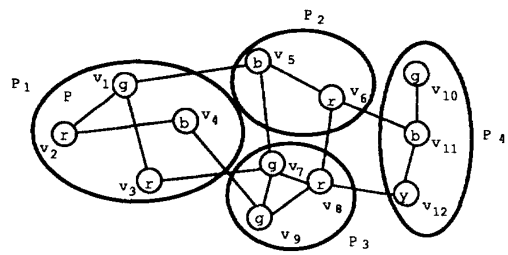

Figure 2: An illegal partitioning.

Figure 1 shows a colored graph G in which $V = \{v_1, v_2, \ldots, v_{12}\}$, $C = \{r, g, b, y\}$, $M_1 = \{r, r, g, g\}$, $M_2 = \{g, g, b, b\}$, $M_3 = \{g, b, y, y\}$, and $M_4 = \{r, r, r\}$. The partitioning shown in Figure 2 is illegal (since, for example, $C_1 = \{r, r, g, b\} \not\subseteq M_j, \forall j$), while the one shown in Figure 3 is legal (since P_i is of type $M_i, \forall i$). In Figure 3, $\Pi = \{P_1, P_2, P_3, P_4\}$ and $\chi(\Pi) = W_{v_1v_5} + W_{v_2v_4} + W_{v_3v_7} + W_{v_5v_6} + W_{v_7v_8} + W_{v_8v_9} + W_{v_6v_{11}} + W_{v_8v_{12}}$.

We present in this paper an algorithm to solve the new graph partitioning problem. The algorithm consists of three phases. In phase 1, the linear programming technique is used to minimize $|\Pi|$ (see section 2). In phase 2, we use a greedy method to obtain a good initial partitioning based upon the result from phase 1, such that the iterative improvement task in phase 3 can be alleviated as much as possible (see section 3). In phase 3, two techniques are iteratively used to improve the in-

*This work was partially supported by the National Science Foundation under grant MIP-8909586 by an IBM Faculty Development Award and by an ACM SIGDA Design Automation Fellowship.

0-7803-1254-6/93$03.00 © 1993 IEEE

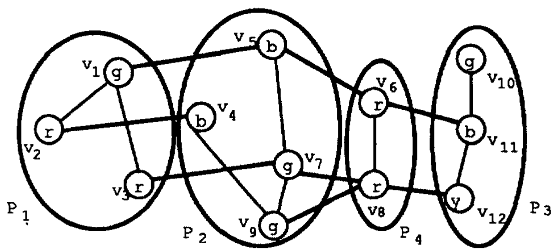

Figure 3: A legal partitioning.

terconnection cost $\chi(\Pi)$. One technique extends the 2-way partitioning approach in [1] (see section 4.1) , and the other technique determines a subset of Π to be repartitioned such that, without increasing $|\Pi|$, $\chi(\Pi)$ is decreased (see section 4.2).

2 Minimizing $|\Pi|$

This phase is based on the linear programming technique. Let x_j be the number of subset (P_i's) of type M_j in Π. Our goal is to minimize the following function:

$$K = x_1 + x_2 + \ldots + x_m \tag{1}$$

We now consider the constraints which x_i's are subjected to. First, we have

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_m \geq 0 \tag{2}$$

Let $n = |C|$ be the total number of colors. Let b_j be the number of vertices in V with color c_j. We represent each M_i by an n-tuple $(a_{i1}, a_{i2}, \ldots, a_{in})$, where a_{ij} denotes the number of times c_j appears in M_i. The following constraints must be also satisfied:

$$a_{1j}x_1 + a_{2j}x_2 + \ldots + a_{mj}x_m \geq b_j \qquad \forall j, 1 \leq j \leq n \tag{3}$$

Note that all a_{ij}'s, b_j's, and x_i's are integers. So it is actually an integer linear programming problem. Since the integer linear programming problem is NP hard, we consider getting an approximated solution by solving the linear relaxation of the integer program. We first obtain an optimal solution $(X_1, X_2, \ldots, X_m)$ (with each X_i being a positive real number) from the linear programming problem. After that, we let $x_i = \lceil X_i \rceil$. We note that $K = x_1 + x_2 + \ldots + x_m$ (i.e., $|\Pi|$) may not be optimal.

3 Initial Partitioning

In this phase we determine an initial legal partitioning $\Pi = \{P_1, \ldots, P_K\}$ of V with some consideration of interconnection cost minimization. Based upon the values of all x_i's obtained from phase 1, we let $Y_1, \ldots, Y_K$ be a collection of multisets defined as follows.

$$Y_i = \quad M_1, \quad \forall i, 1 \leq i \leq x_1 \tag{4}$$

$$Y_{x_1+i} = \quad M_2, \quad \forall i, 1 \leq i \leq x_2 \tag{5}$$

$$\vdots \tag{6}$$

$$Y_{(\sum_{j=1}^{m-1} x_j)+i} = \quad M_m, \quad \forall i, 1 \leq i \leq x_m \tag{7}$$

After this phase is finished, each P_i in Π will be of type Y_i. We use a greedy approach (as described in Algorithm 1) to get a good initial partitioning Π. The idea is that, if two vertices v_1 and v_2 are connected by the edge (v_1, v_2) with a very large weight $W_{v_1 v_2}$, then we try to assign both v_1 and v_2 to some subset P_i. To do this, we first sort the edges $\in E$ in descending order into a list and then sequentially consider each edge in the list (lines 2-31). When an edge $e = (v1, v2)$ is considered, there are 3 cases. (1) If both $v1$ and $v2$ have not been assigned to some P_i, we try to assign them to the same P_i (lines 9-20). (2) If one of the vertices has been assigned, we try to assign the other one to the same P_i(lines 21-30). (3) If both are assigned, no action is taken. If the two endpoints of the current edge can not both be assigned to any P_i, we just leave it alone and consider the next edge in the list. After considering all the edges, for those vertices which have not been assigned to any P_i, we just arbitrarily assign each of them to any available P_i (lines 33-42).

We now analyze the complexity of Algorithm 1. The sorting in line 2 needs time $O(|E| \log |E|)$. In the worst case, the loop from line 3 to 31 takes time $O(|E|K)$, and the loop from line 32 to 42 takes time $(|V|K)$. Since $E = O(|V|^2)$ and $K = |\Pi| \leq |V|$, the worst-cast complexity of Algorithm 1 is $O(|V|^3)$.

Algorithm 1 : Initial Assignment
1. $P_i \leftarrow \{\}, 1 \leq i \leq K$
2. Sort all edges in E into decreasing order and store them in Q_1.
3. L_1:
4. **if** $Q_1 = \{\}$ **then**
5.   **goto** L_2
6. **end if**
7. Get the biggest $e = (x, y)$ in Q_1.
8. $Q_1 \leftarrow Q_1 - \{e\}$
9. **if** neither x nor y is assigned **then**
10. $c_x \leftarrow \alpha(x)$
11. $c_y \leftarrow \alpha(y)$
12.   **for** $i \leftarrow 1$ to K **do**
13.   **if** $c_x \in Y_i$ and $c_y \in Y_i$ **then**
14. $P_i \leftarrow P_i \bigcup \{x, y\}$
15.   Mark x and y as "assigned"
16. $Y_i \leftarrow Y_i - \{c_x, c_y\}$
17.   **goto** L_1
18.   **end if**
19.   **end for**
20. **end if**
21. **else if** one of x, y is unassigned, say x **then**
22.   Determine $y \in P_i$.
23. $c_x \leftarrow \alpha(x)$
24.   **if** $c_x \in Y_i$ **then**
25. $P_i \leftarrow P_i \bigcup \{x\}$
26.   Mark x as "assigned"
27. $Y_i \leftarrow Y_i - \{c_x\}$
28.   **goto** L_1
29.   **end if**
30. **end if**
31. **goto** L_1
32. L_2:
33. Get an unassigned vertex v in V
34. $c_v \leftarrow \alpha(v)$
35. **for** $i \leftarrow 1$ to K **do**
36.   **if** $c_v \in Y_i$ **then**
37. $P_i \leftarrow P_i \bigcup \{v\}$
38. $Y_i \leftarrow Y_i - \{c_v\}$
39.   Mark v as "assigned"
40.   **goto** L_2
41.   **end if**
42. **end for**

4 Interconnection Cost Reduction

After phase 2, we use two techniques to iteratively reduce the interconnection cost $\chi(\Pi)$ of the partitioning Π. One technique

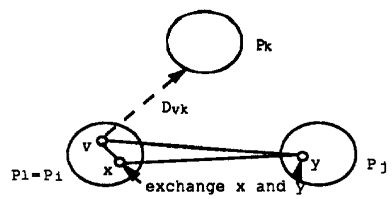

Figure 4: $P_l = P_i$ and $P_k \neq P_j$.

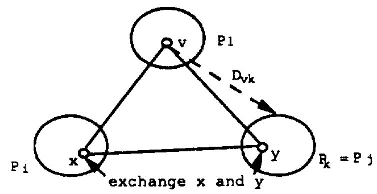

Figure 5: $P_k = P_j$ and $P_l \neq P_i$.

which extends the idea of [1] is presented in section 4.1, and the other is presented in section 4.2.

4.1 Constrained Multiple-Way Partitioning

In [1], an efficient heuristic method for partitioning was presented. This algorithm will be referred to as the K-L algorithm. We develop a constrained K-way partitioning based on this. In our application not all pairs of vertices in V are interchangeable, but only vertices with the same colors are. Similar to [1], we compute the internal cost I_v(scalar value), external cost E_v (vector) , and the difference D_v (vector) for each vertex $v \in V$. We let E_{vi} and D_{vi} denote the external cost and difference of vertex v with respect to each subset P_i where $v \notin P_i$. Algorithm 2 is our constrained K-way partitioning algorithm. Lines 1-11 compute the initial D_v values for each pass. A set Q of interchangeable pairs is constructed in Lines 15-16. Line 31-40 update the D_v values. Basically, there are 4 cases need to be considered when updating D_{vk} after x and y are picked to be swapped. x is in subset P_i and y is in subset P_j. A vertex v is in P_l, and we want to recalculate D_{vk} with respect to P_k. The first 2 cases (lines 32 and 34) are the same as in the 2-way partitioning. The third case is explained by Figure 4 and Figure 5. Figure 4 shows the case when $P_l = P_i$ but $P_k \neq P_j$, while Figure 5 shows the case when $P_k = P_j$ but $P_l \neq P_i$. Line 36 consider both cases. In Figure 4, after exchanging x and y, since $D_{vk} = E_{vk} - I_v$, E_{vk} remains unchanged and I_v should become $I_v - W_{vx} + W_{vy}$, so D_{vk} is recalculated by $D_{vk} + W_{vx} - W_{vy}$ as shown in line 37. Similarly in Figure 5, I_v remains unchanged, but E_{vk} should become $E_{vk} + W_{vx} - W_{vy}$. Thus D_{vk} is also recalculated using the formula in line 37. The fourth case expressed in line 38 is similar. It is obvious that the time complexity of each outermost pass of our algorithm is $O(|V|^3)$, since the ordinary K-L algorithm is a $O(|V|^3)$ procedure, and the constraint needed by our application does not affect the complexity.

Algorithm 2 : Constrained K-way Partitioning

```
0.  loop forever
1.  Clear the "locked" flag on all vertices
2.          for each vertex v ∈ V do
3.              Find the cluster P_l containing v
4.              I_v ← Σ_{v'∈P_l, v'≠v} W_vv'
5.              for i ← 1 to K do
6.                  if i ≠ l then
7.                      E_vi ← Σ_{v'∈P_i} W_vv'
8.                      D_vi ← E_vi − I_v
9.                  end if
10.             end for
11.         end for
12.         t ← 1
13.         loop forever
14.             S_2 ← {}
15.             Q ← {(x,y)|α(x) = α(y) and x ∈ P_u and y ∈ P_u'
16.              and u ≠ u' and x and y are not "locked" }
17.             repeat
18.                 Get (v_1, v_2) from Q. Q ← Q − {(v_1, v_2)}
19.                 Find P_i and P_j containing v_1 and v_2 resp.
20.                 g_v1v2 ← D_v1j + D_v2i − 2W_v1v2
21.                 S_2 ← S_2 ⋃{g_v1v2}
22.                 Q ← Q − {(v_1, v_2)}
23.             until Q = {}
24.             Find the biggest element g_xy ∈ S_2.
25.             if S_2 = {} or g_xy ≤ 0 then goto L_1 end if
26.             G_t ← (g_xy, x, y). t ← t + 1
27.             Mark x and y as "locked
28.             Find P_i and P_j containing x and y resp.
29.             for each unlocked vertex v ∈ V do
30.                 Find P_l containing v
31.                 for k ← 1 to K (k ≠ l) do
32.                     if l = i and k = j then
33.                         D_vk ← D_vk + 2W_vx − 2W_vy
34.                     else if l = j and k = i then
35.                         D_vk ← D_vk + 2W_vy − 2W_vx
36.                     else if l = i or k = j then
37.                         D_vk ← D_vk + W_vx − W_vy
38.                     else if l = j or k = i then
39.                         D_vk ← D_vk + W_vy − W_vx
40.                     end if
41.                 end for
42.             end for
43.         end loop
44. L_1:
45.         G ← max{Σ_{i=1}^{k} G_i(1)|1 ≤ k ≤ t}
46.         if G ≤ 0 then goto L_2 end if
47.         for i ← 1 to k do
48.             (g, v_1, v_2) ← G_i
49.             Interchange v_1 and v_2
50.         end for
51. end loop
52. L_2: exit.
```

4.2 Subset Replacement

In this section we consider a technique for replacing some P_i's in Π such that, without increasing $|\Pi|$, $\chi(\Pi)$ can be further reduced. For example, Figure 6 shows a portion of a colored graph in which, P_1 is of type $M_1 = \{r, g, b\}$, P_2 is of type $M_2 = \{g, b, b\}$, and P_3 is of type $M_3 = \{r, g, g\}$. Note that there are only 2 vertices in P_2. Therefore, one of the components b in M_2 is unused. Assume $M_4 = \{g, g, g, g\}$ and $M_5 = \{r, r, b, b\}$ are also available ($M_4 \in \Omega$ and $M_5 \in \Omega$) . It is obvious that the vertices in the subgraph can be partitioned into two new subsets P_1' of type M_4 and P_2' of type M_5. This is shown in Figure 7 in which $|\Pi|$ is reduced by 1. Also it is possible that the interconnection cost in the new partitioning Π can be reduced. Since it is observed that after the first pass of the method in the previous section, the gain of successive passes is small, we can perform subset replacement before repeating each pass.

To make this approach efficient, we restrict that the size of the subset of Ω (as $\{M_4, M_5\}$ in the example) which replaces the original set (as $\{M_1, M_2, M_3\}$) is at most 2. In order to find the candidate P_i's to be replaced, we chose a sub-

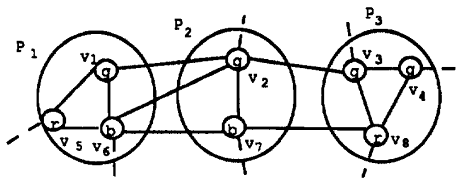

Figure 6: Old Π.

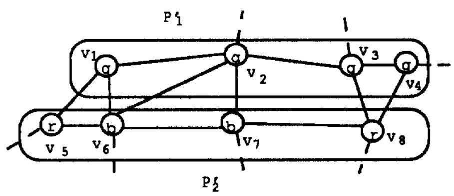

Figure 7: New Π.

set Π' of Π having large interconnection. Assume $|\Pi'| = s$. We then sum up each color in Π' and represent it as a component of vector $\vec{V}$. In our example, let 1st (2nd, 3rd respectively) vector component represents the sum of number of color r (g, b respectively). The vector $\vec{V}$ obtained by (1, 1, 1) + (0, 1, 1) + (1, 2, 0) is (2, 4, 2). We also represent each M_i as vector $\vec{V}_{M_i}$ in the same way. If $|C| = n$ and $|\Omega| = m$, then let $\vec{V} = (b_1, b2, \ldots, b_n)$, and $\vec{V}_{M_i} = (a_{i1}, a_{i2}, \ldots, a_{in}), 1 \leq i \leq m$. We iteratively consider a pair - $\vec{V}_{M_k}$ and $\vec{V}_{M_l}$ (there are $(m-1)m/2$ possibilities) to see if it is possible to do replacement. It is equivalent to solving the following system of linear inequalities.

$$X + Y \leq s \tag{8}$$

$$X \begin{pmatrix} a_{k1} \\ a_{k2} \\ \vdots \\ a_{kn} \end{pmatrix} + Y \begin{pmatrix} a_{l1} \\ a_{l2} \\ \vdots \\ a_{ln} \end{pmatrix} \geq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{9}$$

X and Y are nonnegative integers denoting the number of M_k's and M_l's respectively. Equation 8 implies that the new size can't exceed the original size, and equation 9 implies that the components provided by X M_k's and Y M_l's are enough in the sense that all the vertices in Π' can be assigned. If there is a solution, we locally reassign those components using Algorithm 1 and then again iteratively apply Algorithm 2 to improve the result. Otherwise we consider another pair of multisets in Ω.

5 Experimental Results

We have implemented our algorithms in C programming language. The linear programming codes were obtained from [7]. We ran our program on SUN SPARC station 1. The data we used are as follows. All graphs had 100 vertices. There were 5 colors. The weight of the edges were integers ranged from 1 to 30. We assumed the number of different multisets ($|\Omega|$) was 10. Each M_i had 2 colors and each color appeared 4 times, i.e. a total of 8 elements. We had one multiset for every pair of

Table 1: Comparison of our algorithm to S.A. algorithm

Edge density	$c1/c2$	$w1/w2$	$t1/t2$
0.05	0.936	1.041	0.000498
0.10	0.974	0.998	0.000974
0.15	0.974	1.003	0.001312
0.20	0.988	0.997	0.001339
0.25	1.016	0.995	0.001960
0.30	1.028	1.001	0.001569

colors. For the purpose of comparison, we also implemented a method based on simulated annealing to solve the same problem. Similar to [6], the cost function used by the simulated annealing method considered factors such as inter-chip wiring cost, number of chips, and how far the current partitioning is from the closest legal partitioning. Table 1 shows the results of running our program on graphs with 100 vertices and edge density (the ratio of the number of edges to the number of edges of a complete 100-vertex graph) ranged from 0.05 to 0.30. We experimented on 5 graphs for each edge density. In the table the term $t1/t2$ represents the ratio of the average cpu time consumed by our algorithm to that consumed by the simulated annealing (S.A.) algorithm. $c1/c2$ represents the ratio of the average number of chips of our algorithm to that of S.A. algorithm. $w1/w2$ is the ratio of the average wiring cost of our algorithm to that of S.A. algorithm. The second and third columns indicate that the final results obtained by running our method and simulated annealing method are of comparable quantities. However, our algorithm runs significantly faster as indicated by the fourth column. The average cpu time used by our algorithm is of the order of 10 seconds regardless of the edge density, since the number of passes of K-Way partitioning algorithm and the edge density of graphs are independent.

References

[1] B. W. Kernighan and S. Lin, " An efficient heuristic procedure for partitioning graphs", *Bell Syst. Tech. J.*, vol. 49. pp. 291-307, Feb. 1970.

[2] L. A. Sanchis, "Multiple-way network partitioning", *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 62-81, Jan. 1989.

[3] Ching-Wei Yeh and Chung-Kuan Cheng, "A general purpose multiple way partitioning algorithm", *Proc. 28th ACM/IEEE Design Automation Conf.*, pp.421-426, 1991.

[4] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks", *IEEE Trans. Comput.*, vol. c-33, no. 5, pp. 438-446, May 1984.

[5] T. Bui, C. Heigham, C. Jones and T. Leighton, "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms", *Proc. 26th ACM/IEEE Design Automation Conf.*, pp. 775-778, 1989.

[6] A. Chatterjee and R. Hartley, "A new simultaneous circuit partitioning and chip placement approach based on simulated annealing", *Proc. 27th ACM/IEEE Design Automation Conf.*, pp.36-39, 1990.

[7] W. H. Press, *Numerical recipes in C: the art of scientific computing*, Cambridge Camb#, pp. 329-343, 1988.