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 \( \{\alpha(v)|v \in P_i\} \). \( \Pi \) 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 \( \Pi \) 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 \( \Pi \). The objective of our graph partitioning problem is to find a legal partitioning \( \Pi \) 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 \( M_j \) corresponds to one type of chip in the chip library, and the \( C_i \)'s are the components on chip \( M_j \). Thus \( \Omega \) 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 shows a colored graph \( G \) in which \( V = \{v_1, v_2, \ldots, v_n\} \), \( C = \{r, g, b, y\} \), \( M_1 = \{r, r, g, g\} \), \( M_2 = \{g, g, b, b\} \), \( M_3 = \{g, b, y, y\} \), and \( M_4 = \{r, r, r\} \). The partitioning shown in Figure 2 is illegal (since, for example, \( C_1 = \{r, r, g, b\} \subseteq M_j, \forall j \) 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 \( \chi(\Pi) = W_{v_1v_2} + W_{v_2v_4} + W_{v_3v_7} + W_{v_4v_6} + W_{v_5v_6} + W_{v_6v_9} + W_{v_9v_12} + W_{v_{12}v_1} \).

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-
terconnection cost $\chi(\Pi)$. One technique extends the 2-way par-
tioning approach in [1] (see section 4.1), and the other tech-
nique determines a subset of $\Pi$ 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 $\Pi$. Our goal
is to minimize the following function:

$$K = x_1 + x_2 + \ldots + x_m$$  \hspace{1cm} (1)

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

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_m \geq 0$$  \hspace{1cm} (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 $M_j$ by
an $n$-tuple $(a_{1j}, a_{2j}, \ldots, a_{nj})$, where $a_{ij}$ denotes the number
of times $c_j$ appears in $M_i$. The following constraints must be
also satisfied:

$$a_{1j}x_1 + a_{2j}x_2 + \ldots + a_{nj}x_m \geq b_j, \forall j, 1 \leq j \leq n$$  \hspace{1cm} (3)

Note that all $a_{1j}$'s, $b_j$'s, and $x_j$'s are integers. So it is actu-
ally an integer linear programming problem. Since the inte-
ger linear programming problem is NP hard, we consider get-
ing 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 num-
ber) from the linear programming problem. After that, we let
$x_1 = |X_1|$. 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 intercon-
nect 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.

$$Y_i = M_j, \forall i, 1 \leq i \leq x_1$$  \hspace{1cm} (4)

$$Y_{x_1+i} = M_j, \forall i, 1 \leq i \leq x_2$$  \hspace{1cm} (5)

$$\vdots$$

$$Y_\text{m.} = M_j, \forall i, 1 \leq i \leq x_m$$  \hspace{1cm} (7)

After this phase is finished, each $P_i$ in $\Pi$ will be of type $Y_i$.
We use a greedy approach (as described in Algorithm 1) to get
a good initial partitioning $\Pi$. 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 $W_{v_1v_2}$, then we try to assign both $v_1$ and $v_2$ to some
subset $P_i$. To do this, we first sort the edges $e$ in descending
order into a list and then sequentially consider each edge in the
list (lines 2-31). When an edge $e = (v_1, v_2)$ is considered, there
are 3 cases: (1) If both $v_1$ and $v_2$ 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
different 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 31-42).

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. $P_i = \{\}$, 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. $Q_1 = Q_1 \setminus \{e\}$
9. if neither $x$ nor $y$ is assigned then
10. $c_x = \sigma(x)$
11. $c_y = \sigma(y)$
12. for $i = 1$ to $K$ do
13. if $c_x \notin Y_i$ and $c_y \notin Y_i$ then
14. $P_i = P_i \cup \{x, y\}$
15. $Y_i = Y_i \setminus \{x, y\}$
16. $Y_i = Y_i \setminus \{x, y\}$
17. goto $L_1$
18. end if
19. end for
20. end if
21. if one of $x, y$ is unassigned, say $x$ then
22. Determine $y \in P_i$
23. $c_x = \sigma(x)$
24. if $c_x \notin Y_i$ then
25. $P_i = P_i \cup \{x\}$
26. $Y_i = Y_i \setminus \{x\}$
27. goto $L_1$
28. end if
29. end if
30. end if
31. goto $L_1$
32. $L_2$
33. Get an unassigned vertex $v$ in $V$
34. $c_v = \sigma(v)$
35. for $i = 1$ to $K$ do
36. if $c_v \notin Y_i$ then
37. $P_i = P_i \cup \{v\}$
38. $Y_i = Y_i \setminus \{c_v\}$
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 $\Pi$. One technique
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 \( E_\alpha \) and \( D_\alpha \) denote the external cost and difference of vertex \( v \) with respect to each subset \( P_i \) where \( u \in P_i \). Algorithm 2 is our constrained k-way partitioning algorithm. Lines 1-11 compute the initial \( D_v \) values. Basically, there are 4 cases need to be considered when updating \( D_v \) 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 \( P_i \) and we want to recalculate \( D_v \) with respect to \( P_i \). The first 2 cases (lines 32 and 34) are the same as in the 2-way partitioning. The third case is explained by Figures 4 and 5. Figure 4 shows the case when \( P_i = P_j \) but \( P_k \neq P_j \), while Figure 5 shows the case when \( P_k = P_j \) but \( P_i \neq P_j \). Line 36 consider both cases. In Figure 4, after exchanging \( x \) and \( y \), since \( D_v = E_v - I_v, E_v \) remains unchanged and \( I_v \) should become \( I_v - W_x + W_y \), so \( D_v \) is recalculated by \( D_v + W_x - W_y \) as shown in line 37. Similarly in Figure 5, \( I_v \) remains unchanged, but \( E_v \) should become \( E_v + W_x - W_y \). Thus \( D_v \) 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 \( v \in V \) do
3. Find the cluster \( P_i \) containing \( v \)
4. \( I_v = \sum_{j \in P_i, j \neq v} W_{ij} \)
5. for \( i = 1 \) to \( K \) do
6. if \( i \neq i \) then
7. \( E_v = \sum_{j \in P_i} W_{ij} \)
8. \( D_v = E_v - I_v \)
9. end if
10. end for
11. end for
12. \( t = 1 \)
13. loop forever
14. \( S_t = \{ \} \)
15. \( Q = \{ (x,y) | o(x) = o(y) \text{ and } x \in P_i \text{ and } y \in P_j \text{ and } u 
eq w \text{ and } x \text{ and } y \text{ are not "locked" } \} \)
16. repeat
17. Get \( (v_1, v_2) \) from \( Q \) \( Q = Q - \{ (v_1, v_2) \} \)
18. Find \( P_i \) and \( P_j \) containing \( v_1 \) and \( v_2 \) resp.
19. \( E_{v_1} = D_{v_1} + D_{v_1} - 2W_{v_1v_2} \)
20. \( E_{v_2} = D_{v_2} - D_{v_1} + 2W_{v_1v_2} \)
21. \( E = E_{v_1} \cup E_{v_2} \)
22. \( Q = Q - \{ (v_1, v_2) \} \)
23. until \( Q = \{ \} \)
24. Find the biggest element \( (v_p, v_q) \) in \( S_t \)
25. if \( S_{t} = \{} \) or \( S_{t} < 0 \) then goto Loop 1 end if
26. \( G = (v_p, v_q, y) \) \( t = t + 1 \)
27. Mark \( x \) and \( y \) as "locked"
28. Find \( P_i \) and \( P_j \) containing \( x \) and \( y \) resp.
29. for each unlocked vertex \( v \in V \) do
30. Find \( P_i \) containing \( v \)
31. for \( k = 1 \) to \( k \neq i \) do
32. if \( i = i \) and \( \neq i \) then
33. \( D_{v} = -D_{v} + 2W_{v} - 2W_{v} \)
34. else if \( i = j \) and \( \neq k \) then
35. \( D_{v} = -D_{v} + 2W_{v} - 2W_{v} \)
36. else if \( i = j \) and \( \neq k \) then
37. \( D_{v} = -D_{v} + 2W_{v} - 2W_{v} \)
38. else if \( i = j \) and \( \neq k \) then
39. \( D_{v} = -D_{v} + 2W_{v} - 2W_{v} \)
40. end if
41. end for
42. end loop
43. end loop
44. \( L_1: \)
45. \( G = \max \{ \sum_{i \in \Omega} G_i(\Pi) \leq \leq \} \)
46. if \( G < 0 \) then goto Loop 2 end if
47. for \( i = 1 \) to \( k \) do
48. \( (v_p, v_q) = G_x \)
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 \( \Pi \) such that, without increasing \( ||L|| \), \( \chi(\Pi) \) can be further reduced. For example, Figure 6 shows a portion of a colored graph in which, \( P_i \) is of type \( M_1 = \{ r, b, g \} \), \( P_2 \) is of type \( M_2 = \{ g, b, g \} \), and \( P_3 \) is of type \( M_3 = \{ r, b, g \} \). Note that the vertices are in two vertices in \( P_2 \). Therefore, one of the components \( b \) in \( M_2 \) is unused. Assume \( M_4 = \{ g, g, g \} \) and \( M_5 = \{ r, b, b \} \) 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 \( M_5 \) 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 \( \Pi \) 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 \( \Omega \) (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-
can’t exceed the original size, and equation 9 implies that the components provided by $X$ and $Y$ respectively vector component represents the sum of number of color $r$ ($g$, $b$ respectively). The vector $\vec{V}$ obtained by $\sum_{r} X_r = s$ and $\sum_{r} Y_r = t$ We also represent each $M_i$ as vector $\vec{V}_M$, in the same way. If $|C| = n$ and $|\Pi| = m$, then let $\vec{V} = (v_1, v_2, \ldots, v_n)$, $\vec{V}_M = (a_{11}, a_{12}, \ldots, a_{1n})$, $1 \leq i \leq m$. We iteratively consider a pair of vector $\vec{V}_M \rightarrow \vec{V}_M'$ (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.

$$X + Y \leq s \quad (8)$$

$$X \left( \begin{array}{c} a_{11} \\ a_{12} \\ \vdots \\ a_{1n} \end{array} \right) + Y \left( \begin{array}{c} a_{21} \\ a_{22} \\ \vdots \\ a_{2n} \end{array} \right) \leq \left( \begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_n \end{array} \right) \quad (9)$$

X and Y are nonnegative integers denoting the number of $M_a$’s and $M_b$’s respectively. Equation 8 implies that the new size can’t exceed the original size, and equation 9 implies that the components provided by $X$ $M_a$’s and $Y$ $M_b$’s are enough in the sense that all the vertices in $\Pi$ 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 $\Omega$.

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 (|\Pi|) 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 colors.

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

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

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


1781