The Datapath Merging Problem in Reconfigurable Systems: Lower Bounds and Heuristic Evaluation

Cid C. de Souza[1], André M. Lima[1], Nahri Moreano[2], and Guido Araujo[1]

[1] Institute of Computing, State University of Campinas,
C.P. 6176, 13084-970 Campinas, Brazil.
{cid,andre.lima,guido}@ic.unicamp.br
[2] Department of Computing and Statistics, Federal University of Mato Grosso do Sul,
79070-900 Campo Grande, Brazil.
moreano@dct.ufms.br

Abstract. In this paper we investigate the datapath merging problem (DPM) in reconfigurable systems. DPM is in $\mathcal{NP}$-hard and it is described here in terms of a graph optimization problem. We present an Integer Programming (IP) formulation of DPM and introduce some valid inequalities for the convex hull of integer solutions. These inequalities form the basis of a branch-and-cut algorithm that we implemented. This algorithm was used to compute lower bounds for a set of DPM instances, allowing us to assess the performance of the heuristic proposed by Moreano et al. [1] which is among the best ones available for the problem. Our computational experiments confirmed the efficiency of Moreano's heuristic. Moreover, the branch-and-cut algorithm also was proved to be a valuable tool to solve small-sized DPM instances to optimality.

1 Introduction

It is well known that embedded systems must meet strict constraints of high-throughput, low power consumption and low cost, specially when designed for signal processing and multimedia applications [2]. These requirements lead to the design of application specific components, ranging from specialized functional units and coprocessors to entire application specific processors. Such components are designed to exploit the peculiarities of the application domain in order to achieve the necessary performance and to meet the design constraints.

With the advent of reconfigurable systems, the availability of large/cheap arrays of programmable logic has created a new set of architectural alternatives for the design of complex digital systems [3,4]. Reconfigurable logic brings together the flexibility of software and the performance of hardware [5,6]. As a result, it became possible to design application specific components, like specialized datapaths, that can be reconfigured to perform a different computation, according to the specific part of the application that is running. At run-time, as each portion of the application starts to execute, the system reconfigures the datapath

C.C. Ribeiro and S.L. Martins (Eds.): WEA 2004, LNCS 3059, pp. 545–558, 2004.

so as to perform the corresponding computation. Recent work in reconfigurable computing research has shown that a significant performance speedup can be achieved through architectures that map the most time-consuming application kernel modules or inner-loops to a reconfigurable datapath [7,8,9].

The reconfigurable datapath should have as few and simple hardware blocks (functional units and registers) and interconnections (multiplexors and wires) as possible, in order to reduce its cost, area, and power consumption. Thus hardware blocks and interconnections should be reused across the application as much as possible. Resource sharing has also crucial impact in reducing the system reconfiguration overhead, both in time and space.

To design such a reconfigurable datapath, one must represent each selected piece of the application as a control/data-flow graph (CDFG) and merge them together, synthesizing a single reconfigurable datapath. The control/data-flow graph merging process enables the reuse of hardware blocks and interconnections by identifying similarities among the CDFGs, and produces a single datapath that can be dynamically reconfigured to work for each CDFG. Ideally, the resulting datapath should have the minimum area cost. Ultimately, this corresponds to minimize the amount of hardware blocks and interconnections in the reconfigurable datapath. The datapath merging problem (DPM) seeks such an optimal merging and is known to be in $\mathcal{NP}$-hard [10].

To minimize the area cost one has to minimize the total area required by both hardware blocks and interconnections in the reconfigurable datapath. However, since the area occupied by hardware blocks is typically much larger than that occupied by the interconnections, the engineers are only interested in solutions that use as few hardware blocks as possible. Clearly, the minimum quantity of blocks required for each type of hardware block is given by the maximum number of such block that is needed among all CDFGs passed at the input. The minimum amount of hardware blocks in the reconfigurable datapath can be computed as the sum of these individual minima. As a consequence, DPM reduces to the problem of finding the minimum number of interconnections necessary to implement the reconfigurable datapath.

Fig. 1 illustrates the concept of control/data-flow graph merging and the problem we are tackling. For simplicity, the multiplexors, who select the inputs for certain functional blocks, are not represented. The graphs G' and G represent two mappings of the CDFGs G_1 and G_2. In both these mappings, vertices a_1 and a_5 from G_1 are mapped onto vertices b_1 and b_3 from G_2, respectively, while vertex a_4 of G_1 has no counterpart in G_2. The difference between the two mappings is that, in G' vertex b_2 of G_2 is mapped onto vertex a_2 of G_1, while it is mapped onto a_3 in G. The mappings G' and G are both feasible since they only match hardware blocks that are logically equivalent. Though their reconfigurable datapaths have the same amount of hardware blocks, in G' no arcs are overlapped while in G the arcs (a_3, a_5) and (b_2, b_3) coincide (see the highlighted arc in Fig. 1). In practical terms, this means that one less multiplexor is needed and, therefore, G is a better solution for DPM than G'.

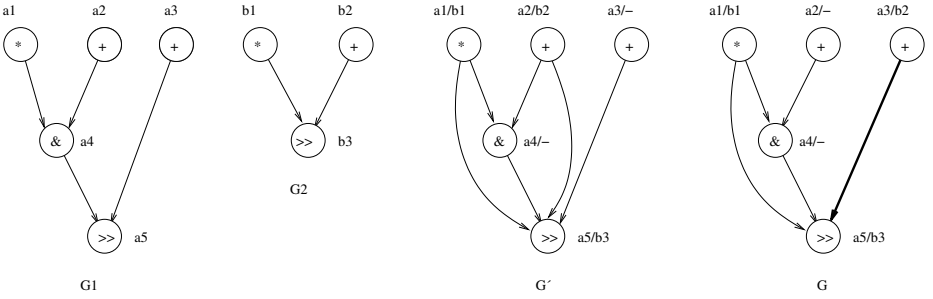

Fig. 1. Example of a DPM instance.

In this paper we present an Integer Programming (IP) formulation for DPM and introduce some valid inequalities for the convex hull of integer solutions. These inequalities form the basis of a branch-and-cut (**B&C**) algorithm that we implemented. The contributions of our work are twofold. First the **B&C** algorithm was able to compute lower bounds for a set of DPM instances, allowing us to assess the performance of the heuristic proposed by Moreano et al. [1], one of the best suboptimal algorithms available for DPM. Secondly, the **B&C** also proved to be a valuable tool to solve small-sized DPM instances to optimality.

The paper is organized as follows. The next section gives a formal description of DPM in terms of Graph Theory. Section 3 briefly discusses Moreano's heuristic. Section 4 presents an IP formulation for DPM, together with some classes of valid inequalities that can be used to tighten the original model. In Sect. 5 we report our computational experiments with the **B&C** algorithm and analyze the performance of Moreano's heuristic. Finally, in Sect. 6 we draw some conclusions and point out to future investigations.

2 A Graph Model for DPM

In this section we formulate DPM as a graph optimization problem. The input is assumed to be composed of n datapaths corresponding to application loops of a computer program. The goal is to find a merging of those datapaths into a reconfigurable one that is able to work as each individual loop datapath alone and has as least hardware blocks (functional units and registers) and interconnections as possible. That is, the reconfigurable datapath must be capable of performing the computation of each loop, multiplexed in time.

The i-th datapath is modeled as a directed graph $G_i = (V_i, E_i)$, where the vertices in V_i represent the hardware blocks in the datapath, and the arcs in E_i are associated to the interconnections between the hardware blocks. The types of hardware blocks (e.g. adders, multipliers, registers, etc) are modeled through a labeling function $\pi_i : V_i \to \mathbb{T}$, where $\mathbb{T}$ is the set of labels representing hardware block types. For each vertex $u \in V_i$, $\pi_i(u)$ is the type of the hardware block associated to u. A reconfigurable datapath representing a solution of DPM can

also be modeled as a directed graph $G = (V, E)$ together with a labeling function $\pi : V \to \mathbb{T}$. In the final graph G, given $i \in \{1, \ldots, n\}$, there exists a mapping μ_i which associates every vertex of V_i to a distinct vertex in V. This mapping is such that, if $v \in V_i$, $u \in V$ and $\mu_i(v) = u$, then $\pi_i(v) = \pi(u)$. Moreover, whenever the arc (v, v') is in E_i, the arc $(\mu_i(v), \mu_i(v'))$ must be in E. If G is an optimal solution for DPM it satisfies two conditions: *(a)* for all $T \in \mathbb{T}$, the number of vertices of G with label T is equal to the maximum number of vertices with that label encountered across all datapaths G_i; and *(b)* $|E|$ is minimum. Condition *(a)* forces the usage of as few hardware blocks as possible in the reconfigurable datapath. As cited before, this is a requirement of the practitioners.

3 Moreano's Heuristic for DPM

Since DPM is $\mathcal{NP}$-hard, it is natural to devise suboptimal algorithms that can solve it fast, preferably in polynomial time. In Moreano et al. [1], the authors proposed a heuristic for DPM and give comparative results showing that it outperforms other heuristics presented in the literature. Moreano's heuristic (MH) is briefly described in this section. In Sect. 5, rather than assess the efficiency of MH using upper bounds generated with other methods, we compare its solutions with strong lower bounds computed via the IP model discussed in Sect. 4.

For an integer $k > 1$, define k-DPM as the DPM problem whose input is made of k loop datapaths. Thus, the original DPM problem would be denoted by n-DPM but the former notation is kept for simplicity. MH is based on an algorithm for 2-DPM, here denoted by `2DPMalg`, that is presented below.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_1)$ be the input graphs and π_1 and π_2 their respective labeling functions. A pair of arcs $\{(u, v), (w, z)\}$ in $E_1 \times E_2$ is said to form a *feasible mapping* if $\pi_1(u) = \pi_2(w)$ and $\pi_1(v) = \pi_2(z)$. The first step of `2DPMalg` constructs the *compatibility graph* $H = (W, F)$ of G_1 and G_2.The graph H is undirected. The vertices in W are in one-to-one correspondence with the pairs of arcs in $E_1 \times E_2$ which form feasible mappings. Given two vertices a and b in W represented by the corresponding feasible mappings, say $a = \{(u, v), (w, z)\}$ and $b = \{(u', v'), (w', z')\}$, the edge (a, b) is in F except if one of the following conditions hold: *(i)* $u = u'$ and $w \neq w'$ or *(ii)* $v = v'$ and $z \neq z'$ or *(iii)* $u \neq u'$ and $w = w'$ or *(iv)* $v \neq v'$ and $z = z'$. If the edge (a, b) is in F, the feasible mappings that they represent are *compatible*, explaining why H is called the compatibility graph. Now, as explained in [1], an optimal solution for 2-DPM can be computed by solving the maximum clique problem on H. The solution of DPM is easily derived from an optimal clique of H since the feasible mappings associated to the vertices of this graph provide the proper matchings of the vertices of G_1 and G_2. However, it is well-known that the clique problem is $\mathcal{NP}$-hard. Thus, the approach used in MH is to apply a good heuristic available for cliques to solve 2-DPM. Later in Sect. 5, we discuss how this is done in practice.

Before we continue, let us give an example of the ideas discussed in the preceding paragraph. To this end, consider the graphs G_1 and G_2 in Fig. 2 representing an instance of 2-DPM. According to the notation used in this figure,

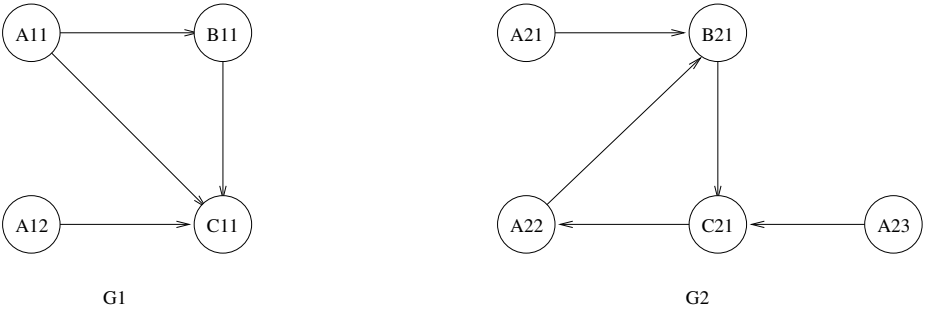

Fig. 2. Example of a 2-DPM instance.

each vertex u in a graph G_i is identified with a label T_{ij}, which denotes that u is the j-th vertex of G_i and $\pi_i(u) = T$. For instance, A_{12} is the second vertex of G_1 which have type A. This notation is used to other figures representing DPM instances and solutions throughout. Figure 3 depicts the compatibility graph H of G_1 and G_2. Consider, for example, the feasible mappings $(A_{11}, B_{11}), (A_{21}, B_{21})$ (vertex w_1 in H) and $(B_{11}, C_{11}), (B_{21}, C_{21})$ (vertex w_5 in H). For those mappings, no vertex from G_1 maps onto two distinct vertices in G_2 and vice-versa. As a result, these two mapping are compatible, and an edge (w_1, w_5) is required in H. On the other hand, no edge exists in H between vertices w_2 and w_3. The reason is that the mappings represented by these vertices are incompatible, since otherwise vertex A_{11} in G_1 would map onto both A_{22} and A_{23} in G_2.

A maximum clique of the compatibility graph H in Fig. 2 is given by vertices w_1, w_4 and w_5. An optimal solution G for 2-DPM can be easily built from this clique. The resulting graph G is shown in Fig. 3 and is obtained as follows. First, we consider the vertices of the clique. For instance, for vertex w_1 represents the feasible mapping $\{(A_{11}, B_{11}), (A_{21}, B_{21})\}$, we add to G two vertices u_1 and u_2 corresponding respectively to the mapped vertices $\{A_{11}, A_{21}\}$ and $\{B_{11}, B_{21}\}$. Moreover, we also include in G the arc (u_1, u_2) to represent the feasible mapping associated to w_1. Analogous operations are now executed for vertices w_4 and w_5. The former vertex is responsible for the addition of vertices u_4 and u_5 and of arc (u_4, u_5) in G while the latter gives rise to the addition of arc (u_2, u_4). Finally, we add to G the vertex u_3 corresponding to the non-mapped vertex A_{22} from

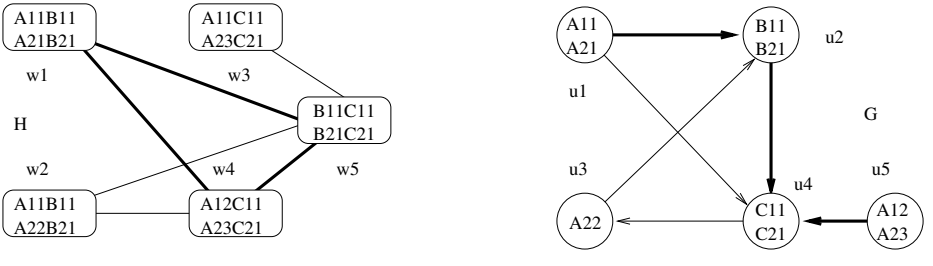

Fig. 3. Compatibility graph and an optimal solution for the 2-DPM instance of Fig. 2.

G_2, and the arcs (u_1, u_4), (u_4, u_3) and (u_3, u_2) corresponding respectively to arcs (A_{11}, C_{11}) from G_1 and arcs (C_{21}, A_{22}) and (A_{22}, B_{21}) from G_2.

Back to MH, we now show how it uses algorithm `2DPMalg` as a building-block for getting suboptimal solutions for DPM. MH starts by applying `2DPMalg` to graphs G_1 and G_2 with labeling functions π_1 and π_2, respectively. The output is a graph G and a labeling function π. At each iteration i, $i \in \{3, \ldots, n\}$, MH applies `2DPMalg` to graphs G and G_i and their functions π and π_i. After all these pairwise matchings have been completed, the graph G is returned.

4 Integer Linear Programming Exact Solution

A natural question that arises when one solves a hard problem heuristically is how far the solutions are from the true optimum. For 2-DPM, algorithm `2DPMalg` from Sect. 3 can be turned into an exact method, provided that an exact algorithm is used to find maximum cliques. However, this approach only works when merging two datapaths. A naive extension of the method to encompass the general case requires the solution of hard combinatorial problems on large-sized instances which cannot be handled in practice. As an alternative, in this section we derive an IP model for DPM. The aim is to compute that model to optimality via IP techniques whenever the computational resources available permit. When this is not the case, we would like at least to generate good lower bounds that allow us to assess the quality of the solutions produced by MH.

Let us denote by α_i the i-th type of hardware block and assume that $\mathbb{T}$ has m elements, i.e., $\mathbb{T} = \{\alpha_1, \ldots, \alpha_m\}$. Moreover, for every $i \in \{1, \ldots, n\}$ and every $t \in \{1, \ldots, m\}$, let us define b_{it} as the number of vertices in V_i associated with a hardware block of type α_t and let $q(t) = \max\{b_{it} : 1 \leq i \leq n\}$. Then, the solutions of DPM are graphs with k vertices, where $k = \sum_{t=1}^{m} q(t)$. In the remainder of the text, we denote by K and N the sets $\{1, \ldots, k\}$ and $\{1, \ldots, n\}$, respectively. Besides, we assume that for every hardware block of type α_t in $\mathbb{T}$ there exists $i \in N$ and $u \in V_i$ such that $\pi_i(u) = \alpha_t$.

When V is given by $\{v_1, v_2, \ldots, v_k\}$, we can assume without loss of generality that $\pi(v_1) = \ldots = \pi(v_{q(1)}) = \alpha_1$, $\pi(v_{q(1)+1}) = \ldots = \pi(v_{q(1)+q(2)}) = \alpha_2$ and so on. In other words, V is such that the first $q(1)$ vertices are assigned to label α_1, the next $q(2)$ vertices are assigned to label α_2 and so on. This assumption reduces considerably the symmetry of the IP model increasing its computability. Below we use the notation J_t to denote the subset of indices in K for which $\pi(v_i) = \alpha_t$ (e.g., $J_1 = \{1, \ldots, q(1)\}$ and $J_2 = \{q(1) + 1, \ldots, q(1) + q(2)\}$).

We are now ready to define the binary variables of our model. For every triple (i, u, j) with $i \in N$, $u \in V_i$ and $j \in J(\pi_i(u))$, let x_{uij} be one if and only if the vertex u of V_i is mapped onto the vertex v_j of V. Moreover, for any pair (j, j') of distinct elements in K, let $y_{jj'}$ be one if and only if there exists $i \in N$ and an arc in E_i such that one of its end-vertices is mapped onto vertex v_j of V while

the other end-vertex is mapped onto $v_{j'}$. The IP model is then the following.

$$\min \quad z = \sum_{\forall j} \sum_{\forall j' \neq j} y_{jj'} \tag{1}$$

$$x_{uij} + x_{u'ij'} - y_{jj'} \leq 1 \qquad \forall i \in N, \forall (u,u') \in E_i, \forall j \in J(\pi_i(u)), \forall j' \in J(\pi_i(u')), j \neq j' \tag{2}$$

$$\sum_{u \in V_i | j \in J(\pi_i(u))} x_{uij} \leq 1 \qquad \forall i \in N, \forall j \in K \tag{3}$$

$$\sum_{j \in J(\pi_i(u))} x_{uij} = 1 \qquad \forall i \in N, \forall u \in V_i \tag{4}$$

$$y_{jj'} \in \{0,1\} \qquad \forall j, j' \in K, j \neq j' \tag{5}$$

$$x_{uij} \in \{0,1\} \qquad \forall i \in N, \forall u \in V_i, \forall j \in J(\pi_i(u)) \tag{6}$$

Equation (1) expresses the fact that an optimal solution to DPM is a graph with as few arcs as possible. Constraints (2) force the existence of arcs in the output graph. Constraints (3) avoid multiple vertices in one input graph to be mapped to a single vertex of the output graph. Finally, (4) guarantees that any vertex in any input graph is mapped to exactly one vertex of V.

Notice that (5) can be replaced by inequalities of the form $0 \leq y_{jj'} \leq 1$ for all $j \neq j'$ with $(j, j') \in K \times K$. This is so because the objective function together with (2) force the y variables to assume values in the limits of the interval $[0, 1]$ and, therefore, to be integer-valued. This remark is important for computational purposes. The most successful algorithms implemented in commercial solvers for IP are based on branch-and-bound (**B&B**) algorithms. The size of the solution space increases exponentially with the number of integer variables in the model. Thus, relaxing the integrality constraints on the y variables in our model, we reduce the search space and increase the chances of success of the algorithm.

The solution of hard combinatorial problems through IP algorithms relies largely on the quality of the dual bounds produced by the linear relaxation of the model at hand. To improve the dual bounds, the relaxation can be amended with additional constraints that are valid for integer solutions of the relaxation but not for all the continuous ones. This addition of valid inequalities tightens the relaxation for its feasibility set strictly decreases. The new constraints, typically chosen from a particular class of valid inequalities, can be either included *a priori* in the model, which is then solved by a standard **B&B** algorithm, or generated on the fly during the enumeration procedure whenever they are violated by the solution of the current relaxation. The latter method gives rise to **B&C** algorithms for IP. Quite often the use of **B&C** is justified by the number of potential inequalities that can be added to the model which, even for limited classes of valid inequalities, is exponentially large. On the other hand, when inequalities are generated on the fly, algorithms that search for violated inequalities are needed. These algorithms solve the so-called *separation problem* for classes of valid inequalities and are named *separation routines*. For a thorough presentation of the Theory of Valid Inequalities and IP in general, we refer to the book by Nemhauser and Wolsey [11]. In the sequel we present two classes of valid inequalities that we use to tighten the formulation given in (1)-(6).

4.1 The Complete Bipartite Subgraph (CBS) Inequalities

The idea is to strengthen (2) using (3). This is done through special subgraphs of the input graphs. Given a directed graph D, we call a subgraph $H = (W_1, W_2, F)$ a CBS of D if, for every pair of vertices $\{w_1, w_2\}$ in $W_1 \times W_2$, (w_1, w_2) is in F. Now, consider an input graph G_i, $i \in N$, of a DPM instance and two distinct labels $\alpha_{t_1}, \alpha_{t_2} \in \mathbb{T}$. Let $H_i^{t_1,t_2} = (V_i^{t_1}, V_i^{t_2}, E_i^{t_1,t_2})$ be a CBS of G_i such that all vertices in $V_i^{t_1}$ ($V_i^{t_2}$) have label α_{t_1} (α_{t_2}). Suppose that $H_i^{t_1,t_2}$ is maximal with respect to vertex inclusion. Assume that v_j and $v_{j'}$ are two vertices in V, the vertex set of the resulting graph G, with labels α_{t_1} and α_{t_2}, respectively. The CBS inequality associated to $H_i^{t_1,t_2}$, v_j and $v_{j'}$ is

$$\sum_{u \in V_i^{t_1}} x_{uij} + \sum_{v \in V_i^{t_2}} x_{vij'} - y_{jj'} \leq 1. \tag{7}$$

Theorem 1. *(7) is valid for all integer solutions of the system (2)-(6).*

Proof. Due to (3), the first summation in the left-hand side (LHS) of (7) cannot exceed one. A similar result holds for the second summation. Thus, if an integer solution exists violating (7), both summations in the LHS have to be one. But then, there would be a pair of vertices $\{u, u'\}$ in $V_i^{t_1} \times V_i^{t_2}$ such that u is mapped onto vertex v_j and u' onto vertex $v_{j'}$. However, as $H_i^{t_1,t_2}$ is a CBS of G_i, $(v_j, v_{j'})$ must be an arc of E, the arc set of the output graph G, i.e., $y_{jj'}$ is one. □

Clearly, if (u, u') is not a maximal CBS of G_i, then (2) is dominated by some inequality in (7) and, therefore, superfluous. Our belief is that the number of CBS inequalities is exponentially large which, in principle, would not recommend to add them all to the initial IP model. However, the DPM instances we tested reveal that, in practical situations, this amount is actually not too large and the CBS inequalities can all be generated via a backtracking algorithm. This allows us to test **B&B** algorithms on models with all these inequalities present.

4.2 The Partition (PART) Inequalities

The next inequalities generalize (2). Consider the i-th input graph $G_i = (V_i, E_i)$, $i \in \{1, \ldots, n\}$. Let u and u' be two vertices in V_i with labels α and α', respectively, with $\alpha \neq \alpha'$ and $(u, u') \in E_i$. Again assume that $G = (V, E)$ is the output graph and that v_j is a vertex of V with label α. Finally, suppose that A and B form a partition of the set $J(\pi_i(u'))$ (see definition in Sect. 4). The PART inequality corresponding to u, u', v_j, A and B is

$$x_{uij} - \sum_{j' \in A} y_{jj'} - \sum_{j' \in B} x_{u'ij'} \leq 0 \tag{8}$$

Theorem 2. *(8) is valid for all integer solutions of the system (2)-(6).*

Proof. If u' is mapped onto a vertex $v_{j'}$ of the resulting graph G and j' is in B, (8) reduces to $x_{uij} - \sum_{j' \in A} y_{jj'} \leq 1$ which is obviously true since $x_{uij} \leq 1$ and $y_{jj'} \geq 0$ for all $j' \in A$. On the other hand, if u' is mapped onto a vertex $v_{j'}$ of G with j' in A, the last summation in (8) is null and the inequality becomes $x_{uij} - \sum_{j' \in A} y_{jj'} \leq 0$. If vertex u is not mapped onto vertex v_j, the latter inequality is trivially satisfied. If not, then necessarily there must be an arc in G joining vertex v_j to some vertex $v_{j'}$ of G with j' in A. This implies that the second summation in (8) is at least one and, therefore, the inequality holds. □

Notice that using (4) we can rewrite (8) as $x_{uij} - \sum_{j' \in A} y_{jj'} + \sum_{j' \in A} x_{u'ij'} \leq 1$ which, for $A = \{j'\}$, is nothing but (2). Moreover, since the size of $J(\pi_i(u'))$, in the worst case, is linear in the total number of vertices of all input graphs, there can be exponentially many `PART` inequalities. However, the separation problem for these inequalities can be solved in polynomial time. This is the ideal situation for, according to the celebrated Grötschel-Lovász-Schrijver theorem [12], the dual bound of the relaxation of (2)-(4) and all inequalities in (8) is computable in polynomial time using the latter inequalities as cutting-planes. A pseudo-code for the separation routine of (8) is shown in Fig. 4 and is now explained.

Given an input graph G_i, $i \in N$, consider two vertices u and u' such that (u, u') is in G_i and a vertex v_j of G whose label is identical to that of u. Now, let $(x^\star, y^\star)$ be an optimal solution of a linear relaxation during the **B&C** algorithm. The goal is to find the partition of the set $J(\pi_i(u'))$ that maximizes the LHS of (8). It can be easily verified that, with respect to the point $(x^\star, y^\star)$ and the input parameters i, u, u' and j, the choice made in line 4 ensures that the LHS of (8) is maximized. Thus, if the value of LHS computed for the partition returned in line 7 is non positive, no constraint of the form (8) is violated, otherwise, (A, B) is the partition that produces the most violated `PART` inequality for $(x^\star, y^\star)$. This routine is executed for all possible sets of input parameters. The number of such sets can be easily shown to be polynomial in the size of the input. Moreover, since the complexity of the routine is $O(J(\pi_i(u')))$ which, in turn, is $O(\sum_{i \in N} |V_i|)$, the identification of all violated `PART` inequalities can be done in polynomial time.

```
procedure part-separation-routine(x*, y*, i, u, u', j);
1    A ← ∅;
2    B ← ∅;
3    for all j' ∈ J(π_i(u')) do{
4        if (y*_{jj'} ≤ x*_{u'ij'}) then A ← A ∪ {j'};
5        else B ← B ∪ {j'};
6    }
7    return (A, B);
end part-separation-routine.
```

Fig. 4. Separation routine for `PART` inequalities.

5 Computational Experiments

We now report on our computational tests with a set of benchmark instances generated from real applications from the MediaBench suite [13]. All programs were implemented in `C++` and executed on a DEC machine equipped with an `ALPHA` processor of 675 MHz, 4 GB of RAM and running under a native Unix operating system. The linear programming solver used was `CPLEX 7.0` and separation routines were coded as *callback functions* from the solver's callable library.

The program implementing heuristic MH resorts to the algorithm of Battiti and Protasi [14] to find solutions to the clique problem. The author's code, that was used in our implementation, can be downloaded from [15] and allows the setting of some parameters. Among them, the most relevant to us is the maximum computation time. Our tests reveal that running the code with this parameter set to one second produce the same results as if we had fixed it to 10 or 15 seconds. Unless otherwise specified, all results exhibited here were obtained for a maximum of computation time of one second. This means that, the MH heuristic as a whole had just a couple of seconds to seek a good solution.

The **B&B** and **B&C** codes that compute the IP models also had their computation times limited. In this case, the upper bound was set to 3600 seconds. **B&B** refers to the basic algorithm implemented in `CPLEX` having the system (1)-(6) as input. The results of **B&B** are identified by the "P" extension in the instance names. The **B&C** algorithms are based on a naive implementation. The only inequalities we generated on the fly are the `PART` inequalities. The separation routine from Fig. 4 is ran until it is unable to encounter an inequality that is violated by the solution of the current relaxation. So, new `PART` constraints are generated exhaustively, i.e., no attempt is done to prevent the well-known stalling effects observed in cutting plane algorithms. A simple rounding heuristic is also used to look for good primal bounds. The heuristic is executed at every node of the enumeration tree. The two versions of **B&C** differ only in the input model which may or may not include the set of `CBS` inequalities. As mentioned earlier, when used, the `CBS` inequalities are all generated a priori by a simple backtracking algorithm. The first (second) version **B&C** algorithm uses the system (1)-(6) (amended with `CBS` inequalities) as input and its results are identified by the "HC" ("HCS") extension in the instance names. It should be noticed that, both in **B&B** and in **B&C** algorithms, the generation of standard valid inequalities provided by the solver is allowed. If fact, Gomory cuts were added by `CPLEX` in all cases but had almost no impact on the dual bounds.

Table 1 summarizes the characteristics of the instances in our data set. Columns "k" and "ℓ" refer respectively to the number of vertices and labels of the output graph G. Columns "G_i", $i \in \{1, \ldots, 4\}$ display the features of each input graph of the instance. For each input graph, the columns "k_i", "e_i" and "ℓ_i" denote the number of vertices, arcs and different labels respectively.

Table 2 exhibits the results we obtained. The first column contains the instance name followed by the extension specifying the algorithm to which the data in the row correspond. The second column reports the `CPU` time in seconds. We do not report on the specific time spent on generating cuts since it is negli-

Table 1. Characteristics of the instances.

Instance name	k	ℓ	G_1			G_2			G_3			G_4			$\sum k_i$	$\sum e_i$
			k_1	e_1	ℓ_1	k_2	e_2	ℓ_2	k_3	e_3	ℓ_3	k_4	e_4	ℓ_4		
adpcm	108	20	97	138	20	78	103	18	–	–	–	–	–	–	175	241
epic_decode	24	6	16	17	3	16	15	4	15	13	4	12	11	5	59	56
epic_encode	39	8	36	43	7	16	17	3	13	13	5	11	10	4	76	83
g721	57	6	57	66	6	19	18	4	–	–	–	–	–	–	76	84
gsm_decode	92	8	91	102	8	65	69	8	20	20	5	19	18	4	195	209
gsm_encode	48	8	46	57	7	41	48	7	20	21	5	19	17	6	126	143
jpeg_decode	104	5	101	111	5	61	68	5	–	–	–	–	–	–	162	179
jpeg_encode	47	7	46	55	7	31	32	5	28	32	6	–	–	–	105	119
mpeg2_decode	34	6	32	31	4	24	29	6	15	15	4	11	10	4	82	85
mpeg2_encode	32	7	31	39	6	30	37	6	20	22	5	18	16	5	99	114
pegwit	41	7	40	46	7	27	28	6	27	30	5	–	–	–	94	104

gible compared to that of the enumeration procedure. Third and fourth columns contain respectively the dual and primal bounds when the algorithm stopped. Column "MH" displays the value of the solution obtained by Moreano's heuristic. To calculate these solutions, MH spent no more than 15 seconds in each problem and `mpeg2_encode` was the only instance in which the clique procedure was allowed to run for more than 10 seconds. Column "gap" gives the percentage gap between the value in column "MH" and that of column "DB" rounded up. Finally, the last two columns show respectively the total numbers of nodes explored in the enumeration tree and of `PART` inequalities added to the model. For each instance, the largest dual bound and the smallest gap is indicated in bold. Ties are broken by the smallest `CPU` time.

By inspecting Tab. 2, one can see that MH produces very good solutions. It solved 4 out of the 11 instances optimally. We took into account here that further testing with the IP codes proved that 60 is indeed the optimal value of instance `pegwit`. When a gap existed between MH's solution and the best dual bound, it always remained below 10%. Additional runs with larger instances showed that the gaps tend to increase, though they never exceeded 30%. However, this is more likely due to the steep decrease in performance of the IP codes than to MH. Instance `epic_decode` was the only case where an optimal solution was found that did not coincided with that generated by MH. Nevertheless, the gap observed in this problem can be considered quite small: 3.39%.

Comparing the gaps computed by the alternative IP codes, we see that the two **B&C** codes outperform the pure **B&B** code. Only in two instances the pure **B&B** code beat both code generation codes. The strength of inequalities `PART` and `CBS` can be assessed by checking the number of nodes explored during the enumeration. This number is drastically reduced when cuts are added as it can be observed, for instance, for problems `adpcm` and `epic_decode` where the use of cuts allowed the computation of the optimum at the root node while **B&B** explored thousands or hundreds of nodes. Instance `g721` was also solved

Table 2. Computational results.

Instance.extension	time	DB	PB	MH	gap	#nodes	#cuts
adpcm.P	3604	165.50	170	168	1.20	9027	0
adpcm.HC	37	**168.00**	168	168	**0.00**	1	813
adpcm.HCS	78	168.00	168	168	0.00	1	848
epic_decode.P	5	33.00	33	33	0.00	143	0
epic_decode.HC	0	**33.00**	33	33	**0.00**	1	90
epic_decode.HCS	0	**33.00**	33	33	**0.00**	1	74
epic_encode.P	2221	**59.00**	59	61	**3.39**	299908	0
epic_encode.HC	3603	57.00	60	61	7.02	1055	4108
epic_encode.HCS	3082	59.00	59	61	3.39	1197	4016
g721_.P	460	70.00	70	70	0.00	50702	0
g721_.HC	880	70.00	70	70	0.00	371	1673
g721_.HCS	112	**70.00**	70	70	**0.00**	116	1259
gsm_decode.P	3616	105.75	–	120	13.21	1601	0
gsm_decode.HC	3614	**112.09**	–	120	**6.19**	0	3655
gsm_decode.HCS	3607	110.25	–	120	8.11	0	2002
gsm_encode.P	3604	67.12	80	72	5.88	23346	0
gsm_encode.HC	3606	**68.24**	73	72	**4.35**	97	6126
gsm_encode.HCS	3604	68.16	79	72	4.35	23	6094
jpeg_decode.P	3606	117.70	163	137	16.10	10651	0
jpeg_decode.HC	3620	**126.20**	–	137	**7.87**	1	4569
jpeg_decode.HCS	3623	125.10	–	137	8.73	1	3669
jpeg_encode.P	3608	61.00	83	71	16.39	63302	0
jpeg_encode.HC	3604	62.92	–	71	12.70	1	5419
jpeg_encode.HCS	3604	**64.10**	–	71	**9.23**	1	5620
mpeg2_decode.P	3613	47.03	51	51	6.25	220173	0
mpeg2_decode.HC	3603	46.09	51	51	8.51	310	4657
mpeg2_decode.HCS	3605	**47.06**	52	51	**6.25**	455	4555
mpeg2_encode.P	3608	46.50	60	53	12.77	86029	0
mpeg2_encode.HC	3603	47.59	55	53	10.42	68	7420
mpeg2_encode.HCS	3603	**48.67**	55	53	**8.16**	115	7778
pegwit.P	3607	**58.22**	60	60	**1.69**	81959	0
pegwit.HC	3604	55.69	62	60	7.14	289	6920
pegwit.HCS	3604	57.77	61	60	3.45	202	5843

to optimality by the **B&C** codes with much fewer nodes than **B&B**, however, when the `CBS` inequalities were not added a priori, this gain did not translate into an equivalent reduction in computation time. In the remaining cases, where optimality could not be proved, again we observed that **B&C** codes computed better dual bounds whereas the number of nodes visited were orders of magnitude smaller than that of **B&B**.

6 Conclusions and Future Research

In this paper we presented an IP formulation for DPM and introduced valid inequalities to tighten this model. Based on this study, we implemented **B&C** and **B&B** algorithms to assess the performance of Moreano's heuristic (MH) for DPM, which is reported as being one of the best available for the problem. Our computational results showed that MH is indeed very effective since it obtains high-quality solutions in a matter of just a few seconds of computation.

The cut generation codes also proved to be a valuable tool to solve some instances to optimality. However, better and less naive implementations are possible that may turn them more attractive. These improvements are likely to be achieved, at least in part, by adding tuning mechanisms that allow for a better trade off between cut generation and branching. For instance, in problems `gsm_decode`, `jpeg_decode` and `jpeg_encode` (see Tab. 2), the **B&C** codes seemed to get stuck in cut generation since they spent the whole computation time and were still at the root node. Other evidences of the need of such tuning mechanisms are given by instances `pegwit` and `g721` were the pure **B&B** algorithm was faster than at least one of the **B&C** codes.

Of course, a possible direction of research would be to perform further polyhedral investigations since they could give rise to new strong valid inequalities for the IP model possibly resulting into better **B&C** codes. Another interesting investigation would be to find what actually makes a DPM instance into a hard one. To this end, we tried to evaluate which of the parameters displayed in Tab. 1 seemed to affect most the computation time of the IP codes. However, our studies were inconclusive. Probably, the structures of the input graphs play a more important role than the statistics that we considered here.

Acknowledgments. This work was supported by the Brazilian agencies FAPESP (grants 02/03584-9, 1997/10982–0 and 00/15083–9), CAPES (grants Bex04444/02–2 and 0073/01–6) and CNPq (grants 302588/02–7, 664107/97–4, 552117/02–1, 301731/03–9 and 170710/99–8).

References

1. Moreano, N., Araujo, G., Huang, Z., Malik, S.: Datapath merging and interconnection sharing for reconfigurable architectures. In: Proceedings of the 15th International Symposium on System Synthesis. (2002) 38–43
2. Wolf, W.: Computers as Components – Principles of Embedded Computing System Design. Morgan Kaufmann Publishers (2001)
3. DeHon, A., Wawrzynek, J.: Reconfigurable computing: What, why, and implications for design automation. In: Proceedings of the Design Automation Conference (DAC). (1999) 610–615
4. Schaumont, P., Verbauwhede, I., Keutzer, K., Sarrafzadeh, M.: A quick safari through the reconfiguration jungle. In: Proceedings of the Design Automation Conference (DAC). (2001) 172–177

5. Compton, K., Hauck, S.: Reconfigurable computing: A survey of systems and software. ACM Computing Surveys **34** (2002) 171–210
6. Bondalapati, K., Prasanna, V.: Reconfigurable computing systems. Proceedings of the IEEE (2002)
7. Callahan, T., Hauser, J., Wawrzynek, J.: The Garp architecture and C compiler. IEEE Computer (2000) 62–69
8. Singh, H., Lee, M., Lu, G., Kurdahi, F., Bagherzadeh, N., Filho, E.: MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. IEEE Transactions on Computers **49** (2000) 465–481
9. Schmit, H., et al.: PipeRench: A virtualized programmable datapath in 0.18 micron technology. In: Proceedings of the IEEE Custom Integrated Circuits Conference (CICC). (2002) 63–66
10. Moreano, N., Araujo, G., de Souza, C.C.: CDFG merging for reconfigurable architectures. Technical Report IC-03-18, Institute of Computing, University of Campinas SP, Brazil (2003)
11. Nemhauser, G.L., Wolsey, L.: Integer and Combinatorial Optimization. Wiley & Sons (1988)
12. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1** (1981) 169–197
13. *MediaBench benchmark* `http://cares.icsl.ucla.edu/MediaBench/`.
14. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29** (2001) 610–637
15. Clique code. `http://rtm.science.unitn.it/intertools/clique/`.