The Datapath Merging Problem in Reconfigurable Systems: Complexity, Dual Bounds and Heuristic Evaluation

CID C. DE SOUZA, ANDRE M. LIMA and GUIDO ARAUJO
Universidade Estadual de Campinas
and
NAHRI B. MOREANO
Universidade Federal de Mato Grosso do Sul

In this paper, we investigate the data path merging problem (DPM) in reconfigurable systems. DPM is modeled as a graph optimization problem and is shown to be $\mathcal{NP}$-hard. An Integer Programming (IP) formulation of the problem is presented and some valid inequalities for the convex hull of integer solutions are introduced. 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 two heuristics proposed earlier in the literature for the problem. Moreover, the branch-and-cut algorithm also was proved to be a valuable tool to solve small-sized DPM instances to optimality.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems; G.1.6 [**Numerical Analysis**]: Optimization—*Integer Programming*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Data path merging, heuristics, lower bounds, reconfigurable systems

1. INTRODUCTION

It is well known that embedded systems must meet strict constraints of high throughput, low power consumption and low cost, especially when designed for signal processing and multimedia applications [Wolf 2001]. These requirements

Authors' address: Cid C. de Souza, André Lima, and Guido Araujo, Instituto de Computação, Universidade Estadual de Campinas, C.P. 6176, 13084-970 Campinas, SP, Brazil, email: `(cid,andre.lima,guido)@ic.unicamp.br`; Nahri B. Moreano, Departamento de Computação e Estatística, Universidade Federal de Mato Grosso do Sul, 79070-900 Campo Grande, MS, Brazil, email: `nahri@dct.ufms.br`.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2005 ACM 1084-6654/05/0001-ART2.11 $5.00

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 [DeHon and Wawrzynek 1999; Schaumont et al. 2001]. Reconfigurable logic brings together the flexibility of software and the performance of hardware [Compton and Hauck 2002; Bondalapati and Prasanna 2002]. As a result, it became possible to design application-specific components, like specialized data paths, 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 data path 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 data path [Callahan et al. 2000; Singh et al. 2000; Schmit et al. 2002].

The reconfigurable data path 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 also has crucial impact in reducing the system reconfiguration overhead, both in time and space.

To design such a reconfigurable data path, one must represent each selected piece of the application as a control/data-flow graph (CDFG) and merge them together, synthesizing a single reconfigurable data path. 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 data path that can be dynamically reconfigured to work for each CDFG. Ideally, the resulting data path should have the minimum area cost. Ultimately, this corresponds to minimize the amount of hardware blocks and interconnections in the reconfigurable data path. The data path merging problem (DPM) seeks such an optimal merging and is shown to be $\mathcal{NP}$-hard in this article. Similar complexity results are given in Moreano et al. [2003] for a more general variant of the problem.

To minimize the area cost; one has to minimize the total area required by both hardware blocks and interconnections in the reconfigurable data path. 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 blocks that are needed among all CDFGs passed at the input. The minimum amount of hardware blocks in the reconfigurable data path can be computed as the sum of these individual minima. As a consequence,

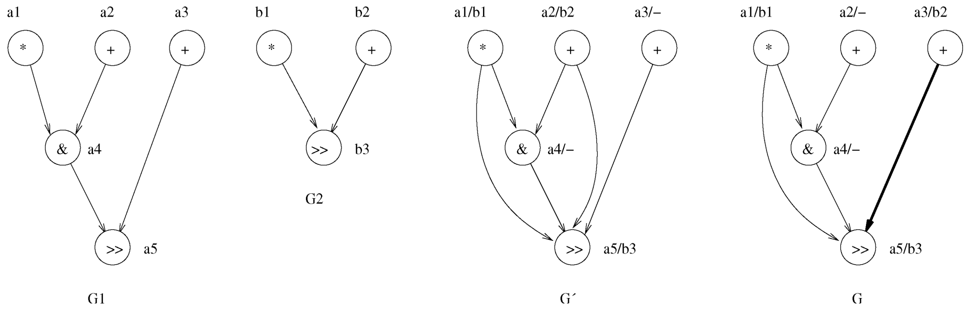

Fig. 1. Example of a DPM instance.

DPM reduces to the problem of finding the minimum number of interconnections necessary to implement the reconfigurable data path.

Figure 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. Although their reconfigurable data paths 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 Figure 1). In practical terms, this means that one less multiplexor is needed and, therefore, G is a better solution for DPM than G'.

In this paper, we present an 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 two heuristics available for the problem. Second, 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 and contains our proof that the problem is $\mathcal{NP}$-hard. Sections 3 and 4 briefly discusses the two heuristic algorithms that we used in our computations. Section 5 presents an IP formulation for DPM, together with some classes of valid inequalities that can be used to tighten the original model. In Section 6, we report our computational experiments with the B&C algorithm and analyze the performance of the heuristics. Finally, in Section 7, we draw some conclusions and point out future investigations.

2. GRAPH MODEL FOR DPM

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

The ith data path is modeled as a directed graph $G_i = (V_i, E_i)$, where the vertices in V_i represent the hardware blocks in the data path, and the arcs in E_i are associated to the interconnections between the hardware blocks. The types of hardware blocks (e.g., adders, multipliers, and registers) 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 feasible solution for DPM is given by three elements:

- A reconfigurable data path modeled as a directed graph $G = (V, E)$
- A labeling function $\pi : V \to \mathbb{T}$
- For each $i \in \{1, \ldots, n\}$, a mapping μ_i which associates every vertex of V_i to a distinct vertex in V, 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.

Finally, in a feasible solution, for all $T \in \mathbb{T}$, the number of vertices of G with label T must be equal to the maximum number of vertices with that label encountered across all data paths G_i. The latter condition forces the usage of as few hardware blocks as possible in the reconfigurable data path which, as cited before, is a requirement of the practitioners. An *optimal* solution for DPM is a feasible one for which $|E|$ is minimum.

We now prove that the decision version of DPM (DPM-D) is $\mathcal{NP}$-complete. In the decision problem, an integer k is given in the input and one has to decide whether or not there exists a feasible solution with $|E| = k$. Before, we continue, let us recall the definition of the subgraph isomorphism problem (ISO).

Definition 2.1. Given two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, does G_1 contain a directed subgraph $H = (V_H, E_H)$ isomorphic to G_2, i.e., is there a bijective function $f : V_2 \to V_H$, such that $(u, v) \in E_2 \Leftrightarrow (f(u), f(v)) \in E_H$?

It is well-known that ISO is $\mathcal{NP}$-complete (see [Garey and Johnson 1979, GT48]).

THEOREM 2.2. *DPM-D is $\mathcal{NP}$-complete.*

PROOF. Since DPM-D can be easily seen to be in $\mathcal{NP}$, we only show that the problem is $\mathcal{NP}$-hard. To do so, we reduce an arbitrary instance of ISO to an instance of DPM-D. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the graphs in the input of ISO. Assume that $|V_1| \geq |V_2|$ and $|E_1| \geq |E_2|$, since otherwise ISO can be solved trivially. For the DPM instance, we set $n = 2$ and the input graphs $G'_1 = G_1$ and $G'_2 = G_2$. Besides, the set $\mathbb{T}$ of labels is created with a single element T, which represents a type of hardware block that performs a commutative operation. The label functions π_1 and π_2 map all vertices of G'_1 and G'_2, respectively, to the unique label T. Finally, the parameter k is set to $|E_1|$. Clearly, this reduction can be done in polynomial time.

We now claim that G_1 has a subgraph isomorphic to G_2 if, and only if, there is a merged graph $G = (V, E)$ corresponding to G'_1 and G'_2, such that (a) the number of vertices of G with label T is equal to V_1, and (b) $|E| = k$.

Suppose that G_1 and G_2 is a yes-instance for ISO. Thus, there exists a subgraph $H = (V_H, E_H)$ of G_1 that is isomorphic to G_2. Let $f : V_2 \rightarrow V_H$ be the bijective function corresponding to this isomorphism. A feasible solution for DPM-D is built as follows. First, let the resulting graph $G = (V, E)$ be such that $|V| = |V_1|$. Let the mapping function μ_1 be any bijection from V_1 to V. Now, define the mapping $\mu_2 : V_2 \rightarrow V$ as the composition of μ_1 and f, i.e., $\mu_2 = \mu_1 \circ f$. Clearly, in this case, $E = E_1$. Therefore, G has $k = |E_1|$ arcs and we have a yes-instance for DPM-D.

Conversely, suppose that DPM-D has a yes answer. Assume that $G = (V, E)$ is a merged graph corresponding to a feasible solution of the problem, which also includes the two mapping functions $\mu_1 : V_1 \rightarrow V$ and $\mu_2 : V_2 \rightarrow V$. Feasibility implies that $|E| = k$ and that all vertices in V have label T. By hypothesis, since $|V_1| \geq |V_2|$ and the labels of all vertices in V_1 and in V_2 are identical to T, the sizes of V and V_1 must be equal. Since $|E| = k = |E_1|$, mapping $\mu_2 : V_2 \rightarrow V$; must be such that; for every (u, v) in E_2, $(\mu_1^{-1}(\mu_2(u)), \mu_1^{-1}(\mu_2(v)))$ is in E_1. Hence, the function $f = \mu_1^{-1} \circ \mu_2 : V_2 \rightarrow V_1$ defines an isomorphism between a subgraph of G_1 and G_2. Thus, ISO has also a yes answer. □

It should be noticed that the input graphs for DPM may contain cycles. This occurs when the selected pieces of the application have loops. Now, if we restrict these pieces to inner-loop bodies, the CDFGs will be acyclic. However, our proof remains valid even in this case since, according to Garey and Johnson [1979], ISO is also $\mathcal{NP}$-complete when G_1 is acyclic and G_2 is a directed tree. Another observation refers to the situation where G_1 is directed forest and G_2 is a directed tree, in which case, ISO can be solved in polynomial time. However, CDFGs typically have vertices with more than one incoming arc (modeling a hardware block with more than one operand) and more than one outcoming arc (modeling a common subexpression, that is, when the result of an operation is used by two or more different blocks). Thus, this polynomially solvable instance is not likely to occur in practical situations.

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. [2002], the authors proposed a heuristic for DPM. Computational tests conducted by the authors on a few instances indicated that the algorithm outperforms other heuristics presented in the literature. Moreano's heuristic (MH) is briefly described in this section. In Section 6, the efficiency of MH is assessed both using other upper bounds generated by the heuristic presented in the next section and using the strong lower bounds computed via the IP model discussed in Section 5.

For an integer $k > 1$, define k-DPM as the DPM problem whose input is made of k loop data paths. 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.

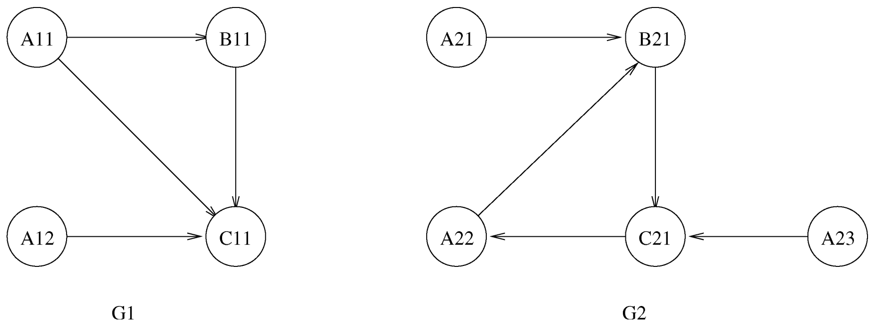

Fig. 2. Example of a 2-DPM instance.

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 arc (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 arc (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 Moreano et al. [2002], 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 $\mathcal{NP}$-complete. Thus, the approach used in MH is to apply a good heuristic available for cliques to solve 2-DPM. Later in Section 6, 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 Figure 2 representing an instance of 2-DPM. According to the notation used in this figure, 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 in 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 arc (w_1, w_5) is required in H. On the other hand, no arc 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.

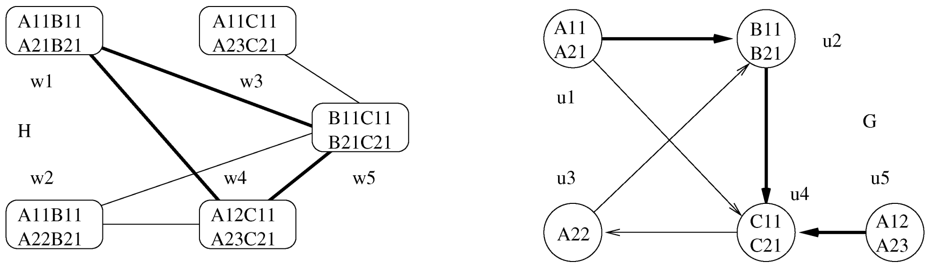

Fig. 3. Compatibility graph and an optimal solution for the 2-DPM instance of Figure 2.

A maximum clique of the compatibility graph H in Figure 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 Figure 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 nonmapped vertex A_{22} from 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 obtaining 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. BIPARTITE MATCHING HEURISTIC FOR DPM

The most commonly used method for DPM is based on the problem of finding a maximum weight matching in a bipartite graph [Geurts et al. 1997; Shirazi et al. 1998; Huang and Malik 2001]. Because of its popularity among practicioners, we decided to include this heuristic in our computational tests. Thus, in this section we briefly describe the bipartite matching heuristic (BMH). Later, in Section 6, we compare the results obtained by BMH with those of MH and with the lower bounds generated with our IP model.

As for MH, BMH is based on an algorithm for 2-DPM. This algorithm is applied iteratively in order to merge n data paths and is described below.

Given two input graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and their respective labeling functions π_1 and π_2, a bipartite undirected graph $B = (V_1 \cup V_2, E_B)$ is built. For each pair $(u_1, u_2) \in V_1 \times V_2$, the edge (u_1, u_2) is in E_B if, and only if, $\pi_1(u_1) = \pi_2(u_2)$, that is, if u_1 and u_2 can be mapped to the same vertex u of the resulting graph G. The weight assigned to this edge represents the gain

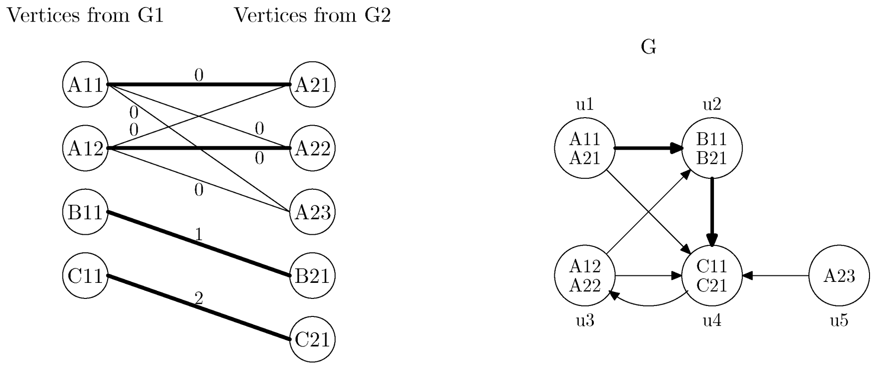

Fig. 4. Bipartite graph and suboptimal solution for the 2-DPM instance of Figure 2.

obtained with this mapping according to some predefined cost function. As an example, in Huang and Malik [2001], the cost of (u_1, u_2) corresponds to the maximum number of pair of arcs $(*, u_1) \in E_1$ and $(*, u_2) \in E_2$ that can be mapped to the same arc $(*, u)$ in G. Notice that this cost is an overestimation since, in an optimal solution, not necessarily all predecessor vertices of u_1 in G_1 are mapped to predecessors of u_2 in G_2. Other cost functions are proposed in the literature [cf. Geurts et al. 1997; Shirazi et al. 1998]. However, to compute these functions further details on the circuit architecture are required. Since no such data is available in the input of DPM, in our computational experiments, BMH was implemented with the cost function described above.

Now, since each vertex of G_1 must be mapped to, at most, one vertex of G_2 and vice versa, the maximum weight matching of B provides the vertex mappings that maximize the cost function. The merged graph G is built from these mappings. Initially, for each edge (u_1, u_2) of the matching, a vertex is added to G, corresponding to the mapped vertices $\{u_1, u_2\}$. Then, for each $u \in V_1 \cup V_2$, which is not saturated by the matching, a vertex is added to G, corresponding to the nonmapped vertex u. Finally, for each arc of G_1, a corresponding arc is inserted in G, and for each arc of G_2, which can not be mapped to an existing arc of G, a new arc is added to G.

Figure 4 illustrates this algorithm and shows the bipartite graph B obtained from the CDFGs G_1 and G_1 of Figure 2. A maximum weight matching and the resulting graph G are also depicted in Figure 4.

With respect to execution times, BMH is usually considered efficient, since a polynomial-time algorithm for the problem of finding a maximum weight matching in a bipartite graph is well-known (the Hungarian Method, described in Papadimitriou and Steiglitz [1998]). However, since the merged graph is built using only estimates about the arc mappings, the solution produced may be far from optimal.

We close this section by mentioning a few other related works in which heuristics have been proposed for the problem, although with possibly different optimization goals.

Another group of methods for the data path merging problem relies on search mechanisms that explore the solution space looking for an optimal solution. In

Werf et al. [1992] a merging technique is proposed, based on iterative improvement and simulated annealing local search algorithms, and aims at reducing the interconnection cost. Both algorithms start with an initial random assignment, and continuously step through a randomly chosen neighborhood solution set, according to an acceptance criterion. Given a solution of the problem, a two-exchange movement is obtained when the assignment of two operations from the same CDFG are exchanged. The neighborhood of the current solution is then, defined as the set of solutions that can be reached by performing a single two-exchange movement. The solution provided by the iterative improvement algorithm is the first local minimum found, so it may be far from optimal. The simulated annealing method accepts limited deteriorating transitions, trying to escape from local minima and, as a consequence, has longer execution times than classical local search algorithms. The actual complexity of the resulting algorithm depends on how some parameters are fixed.

A different heuristic for the data path merging problem is presented in Brisk et al. [2004], with the goal of minimizing the data path area. Initially, the algorithm enumerates all paths of each input data-flow graph, and then overlaps these paths iteratively, based on the longest common subsequence of each pair of paths from different graphs. Since the number of paths in a data-flow graph can be exponential with respect to the number of its vertices, this algorithm has a exponential execution time.

In Mathur and Saluja [2001], the authors propose transformations on the data-flow graphs based on information content and required precision analysis in order to reduce the width of the data path operators. They also present an iterative algorithm for partitioning the data path into clusters, so that the operators of a cluster can be merged. At each iteration, the algorithm recomputes the required precision and the information content of the signals and improves the partitioning. Experimental results reveal that area reduction can be achieved when such transformations are done. However, the approach is restricted to graphs in which each cluster represents a sum of terms derived from input signals.

5. 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 Section 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 data paths. 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 ith type of hardware block and assume that $\mathbb{T}$ has m elements, i.e., $\mathbb{T} = \{\alpha_1, \dots, \alpha_m\}$. Moreover, for every $i \in \{1, \dots, n\}$ and every

$t \in \{1, \dots, m\}$, let us define b_{it} as the number of vertices in V_i associated with a hardware block of type α_t and let $q(t) = \max\{b_{it} : 1 \le i \le n\}$. The solutions of DPM are then 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 $\{1, \dots, k\}$ and $\{1, \dots, n\}$, 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 $\{v_1, v_2, \dots, v_k\}$, we can assume without loss of generality that $\pi(v_1) = \cdots = \pi(v_{q(1)}) = \alpha_1$, $\pi(v_{q(1)+1}) = \cdots = \pi(v_{q(1)+q(2)}) = \alpha_2$ 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 considerably reduces 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., $J_1 = \{1, \dots, q(1)\}$ and $J_2 = \{q(1)+1, \dots, q(1)+q(2)\}$).

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.

$$\begin{array}{lrll}
\min & z = \sum_{\forall j} \sum_{\forall j' \neq j} y_{jj'} & & (1) \\
 & x_{uij} + x_{u'ij'} - y_{jj'} \le 1 & \forall i \in N, \forall (u, u') \in E_i, \forall j \in J(\pi_i(u)), & \\
 & & \forall j' \in J(\pi_i(u')), j \neq j' & (2) \\
 & \sum_{u \in V_i | j \in J(\pi_i(u))} x_{uij} \le 1 & \forall i \in N, \forall j \in K & (3) \\
 & \sum_{j \in J(\pi_i(u))} x_{uij} = 1 & \forall i \in N, \forall u \in V_i & (4) \\
 & y_{jj'} \in \{0, 1\} & \forall j, j' \in K, j \neq j' & (5) \\
 & x_{uij} \in \{0, 1\} & \forall i \in N, \forall u \in V_i, \forall j \in J(\pi_i(u)) & (6)
\end{array}$$

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

Notice that Eq. (5) can be replaced by inequalities of the form $0 \le y_{jj'} \le 1$ for all $j \neq j'$ with $(j, j') \in K \times K$. This is so because the objective function together with Eq. (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

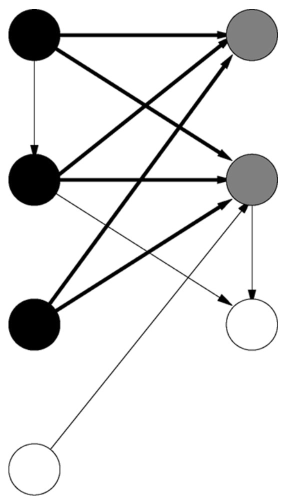

Fig. 5. Example of a CBS of a graph.

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 socalled *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 [1988]. In the sequel, we present two classes of valid inequalities that we use to tighten the formulation given in Eqs. (1–6).

5.1 The Complete Bipartite Subgraph (CBS) Inequalities

The idea is to strengthen Eq. (2) using Eq. (3). This is done through special subgraphs of the input graphs. Given a directed graph $D = (W, A)$, we call a subgraph $H = (W_1, W_2, F)$ of D a CBS if, for every pair of vertices $\{w_1, w_2\}$ in $W_1 \times W_2$, (w_1, w_2) is in F. Notice that, since H is a subgraph of D, we must have that $F \subseteq A$. In other words, this means that H has all the directed arcs going from W_1 to W_2 as illustrated in Fig. 5. In this drawing, vertices in set W_1 (W_2) are represented by black (grey) circles, while the arcs in F are indicated by the thick arrows.

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 5.1. *Equation* (7) *is valid for all integer solutions of the system Eq.* (2 − −6).

PROOF. Because of (3), the first summation in the left-hand side (LHS) of Eq. (7) cannot exceed one. A similar result holds for the second summation. Thus, if an integer solution exists violating Eq. (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 Eq. (2) is dominated by some inequality in Eq. (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.

5.2 The Partition (PART) Inequalities

The next inequalities generalize Eq. (2). Consider the ith 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 Section 5). 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 5.2. *Equation* (8) *is valid for all integer solutions of the system Eq. (2)–(6).*

PROOF. If u' is mapped onto a vertex $v_{j'}$ of the resulting graph G and j' is in B, Eq. (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 Eq. (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

```
procedure part-separation-routine($x^\star, y^\star, i, u, u', j$);
1   $A \leftarrow \emptyset$;
2   $B \leftarrow \emptyset$;
3   for all $j' \in J(\pi_i(u'))$ do{
4       if $(y^\star_{jj'} \leq x^\star_{u'ij'})$ then $A \leftarrow A \cup \{j'\}$;
5       else $B \leftarrow B \cup \{j'\}$;
6   }
7   return $(A, B)$;
end part-separation-routine.
```

Fig. 6. Separation routine for PART inequalities.

the second summation in Eq.(8) is at least one and, therefore, the inequality holds. □

Notice that using Eq. (4) we can rewrite Eq. (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 Eq. (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 [Gröptschel et al. 1981], the dual bound of the relaxation of Eq. (2)–(4) and all inequalities in Eq. (8) is computable in polynomial time using the latter inequalities as cutting-planes. Figure 6 shows a pseudocode for the separation routine of Eq. (8), which is discussed below.

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 Eq. (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 Eq. (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 Eq. (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.

6. COMPUTATIONAL EXPERIMENTS

We now report on our computational tests with a set of benchmark instances generated from real applications from the MediaBench suite [Lee et al. 1997]. 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 [2001] to find solutions to the clique problem. The author's code, which was used in our implementation, can be downloaded from the web 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 1s produce the same results as if we had fixed it to 10 or 15 s. Unless otherwise specified, all results exhibited here were obtained for a maximum of computation time of 1s. 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 s. B&B refers to the basic algorithm implemented in CPLEX having the system Eqs. (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 Figure 6 is run until it is unable to encounter an inequality that is violated by the solution of the current relaxation. Thus, new PART constraints are generated exhaustively, i.e., no attempt is made 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 Eqs. (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. In fact, Gomory cuts were added by CPLEX in all cases, but had almost no impact on the dual bounds.

Tables I and II summarize 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", $i \in \{1, \dots, 4\}$ 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 III 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 negligible compared to that of the enumeration procedure. Third and fourth columns contain, respectively, the dual and primal bounds when the algorithm stopped. Columns "MH" and "BMH" display the values of the solutions obtained by Moreano's heuristic the bipartite matching heuristic, respectively. Column "g1" gives the percentage gap between the value in column "MH" and that of column "DB" rounded up. Similarly, column "g2" gives the percentage gap between the values in columns "BMH" and "MH." 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

Table I. Characteristics of the Instances

Instance Name	k	ℓ	$\sum k_i$	$\sum e_i$
adpcm	108	20	175	241
epic_decode	24	6	59	56
epic_encode	39	8	76	83
g721	57	6	76	84
gsm_decode	92	8	195	209
gsm_encode	48	8	126	143
jpeg_decode	104	5	162	179
jpeg_encode	47	7	105	119
mpeg2_decode	34	6	82	85
mpeg2_encode	32	7	99	114
pegwit	41	7	94	104

Table II. Characteristics of the Instances (Cont.)

	G_1			G_2			G_3			G_4		
Instance Name	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	97	138	20	78	103	18	–	–	–	–	–	–
epic_decode	16	17	3	16	15	4	15	13	4	12	11	5
epic_encode	36	43	7	16	17	3	13	13	5	11	10	4
g721	57	66	6	19	18	4	–	–	–	–	–	–
gsm_decode	91	102	8	65	69	8	20	20	5	19	18	4
gsm_encode	46	57	7	41	48	7	20	21	5	19	17	6
jpeg_decode	101	111	5	61	68	5	–	–	–	–	–	–
jpeg_encode	46	55	7	31	32	5	28	32	6	–	–	–
mpeg2_decode	32	31	4	24	29	6	15	15	4	11	10	4
mpeg2_encode	31	39	6	30	37	6	20	22	5	18	16	5
pegwit	40	46	7	27	28	6	27	30	5	–	–	–

bound and the smallest gap is indicated in bold. Ties are broken by the smallest CPU time.

By inspecting Table III, one can see that MH produces very good solutions. It optimally solved 4 out of the 11 instances. 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 also, 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, although they never exceeded 30%. However, this is more likely because of 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 coincide with that generated by MH. Nevertheless, the gap observed in this problem can be considered quite small: 3.4%.

Moreover, from column "g2," we conclude that MH is much more efficient than BMH since it produced better results in all instances in our dataset. This is in accordance with what was observed in Moreano et al. [2002] on a smaller benchmark. To calculate these solutions, MH spent no more than 15 s on each problem and `mpeg2_encode` was the only instance in which the clique procedure was allowed to run for more than 10 s. On the other hand, BMH proved to be much faster than MH since the computation time always remained below 1.5 s.

Table III. Computational Results

Instance.extension	Time	DB	PB	MH	BMH	g1	g2	#Nodes	#Cuts
adpcm.P	3604	165.5	170	168	221	1.2	31.5	9027	0
adpcm.HC	37	**168.0**	168			**0.0**		1	813
adpcm.HCS	78	168.0	168			0.0		1	848
epic_decode.P	5	33.0	33	33	48	0.0	45.5	143	0
epic_decode.HC	0	**33.0**	33			**0.0**		1	90
epic_decode.HCS	0	**33.0**	33			**0.0**		1	74
epic_encode.P	2221	**59.0**	59	61	71	**3.4**	16.4	299908	0
epic_encode.HC	3603	57.0	60			7.0		1055	4108
epic_encode.HCS	3082	59.0	59			3.4		1197	4016
g721_.P	460	70.0	70	70	84	0.0	20.0	50702	0
g721_.HC	880	70.0	70			0.0		371	1673
g721_.HCS	112	**70.0**	70			**0.0**		116	1259
gsm_decode.P	3616	105.8	–	120	187	13.2	55.8	1601	0
gsm_decode.HC	3614	**112.1**	–			**6.2**		0	3655
gsm_decode.HCS	3607	110.3	–			8.1		0	2002
gsm_encode.P	3604	67.1	80	72	108	5.9	50.0	23346	0
gsm_encode.HC	3606	**68.2**	73			**4.4**		97	6126
gsm_encode.HCS	3604	68.1	79			4.4		23	6094
jpeg_decode.P	3606	117.7	163	137	171	16.1	24.8	10651	0
jpeg_decode.HC	3620	**126.2**	–			**7.9**		1	4569
jpeg_decode.HCS	3623	125.1	–			8.7		1	3669
jpeg_encode.P	3608	61.0	83	71	97	16.4	36.6	63302	0
jpeg_encode.HC	3604	62.9	–			12.7		1	5419
jpeg_encode.HCS	3604	**64.1**	–			**9.2**		1	5620
mpeg2_decode.P	3613	47.0	51	51	70	6.3	37.3	220173	0
mpeg2_decode.HC	3603	46.0	51			8.5		310	4657
mpeg2_decode.HCS	3605	**47.1**	52			**6.3**		455	4555
mpeg2_encode.P	3608	46.5	60	53	80	12.8	50.9	86029	0
mpeg2_encode.HC	3603	47.6	55			10.4		68	7420
mpeg2_encode.HCS	3603	**48.7**	55			**8.2**		115	7778
pegwit.P	3607	**58.2**	60	60	82	**1.7**	36.7	81959	0
pegwit.HC	3604	55.7	62			7.2		289	6920
pegwit.HCS	3604	57.8	61			3.5		202	5843

Despite its larger computational times, MH remains a fast algorithm and its use is justified by large gains in the quality of the solutions.

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 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, we again 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.

7. 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 two heuristics for the problem. Moreano's heuristic (MH) was confirmed to be among the best available suboptimal algorithms available for DPM, beating the popular bipartite matching heuristic (BMH) in all tested instances. Although MH required a larger computational time than BMH, it still used just a few seconds of CPU. The extra time needed relative to BMH was largely compensated by the gain in the quality of the solutions. Besides, the lower bounds produced with the IP model showed that the cost of the solutions achieved by MH are very close to the optimal values.

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 make 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 Table III), 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, which was faster than at least one B&C code.

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 in 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 Tables I and II seemed to affect most the computation time of the IP codes. However, our studies were inconclusive. Below we illustrate the difficulty of doing such analysis.

Consider the last two instances in Table I, namely: `mpeg2_encode` and `pegwit`. By inspecting the data in this table, one can see that output graph in both cases has seven labels. In addition, the total number of vertices and arcs in the input graphs are also very similar. However, the best duality gap displayed in Table III is approximately five times larger for `mpeg2_encode`. This observation apparently suggests that the number of labels is not determinant to evaluate the hardness of an instance. On the other hand, these two instances have a different number of input graphs. It is then legitimate to argue if the latter parameter is not the one that is more related to the instance difficulty. Consider the instances `jpeg_encode` and `pegwit`, whose statistics in Table I are all very close and have exactly the same number of input graphs. However, again, the best duality gaps shown in Table III for these instances are related by a factor of approximately 5.

Thus, we conclude that the intrinsic structures of the input graphs are more likely to affect the difficulty of an instance than the statistics that we considered here.

ACKNOWLEDGMENTS

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

REFERENCES

Battiti, R. and Protasi, M. 2001. Reactive local search for the maximum clique problem. *Algorithmica 29,* 4 (April), 610–637. C++ code available at `http://rtm.science.unitn.it/intertools/clique/`.

Bondalapati, K. and Prasanna, V. 2002. Reconfigurable computing systems. *Proceedings of the IEEE 90*, 7 (July), 1201–1217.

Brisk, P., Kaplan, A., and Sarrafzadeh, M. 2004. Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In *Proceedings of the Design Automation Conference (DAC)*. 395–400.

Callahan, T., Hauser, J., and Wawrzynek, J. 2000. The Garp architecture and C compiler. *IEEE Computer 33*, 4 (April), 62–69.

Compton, K. and Hauck, S. 2002. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys 34*, 2 (June), 171–210.

DeHon, A. and Wawrzynek, J. 1999. Reconfigurable computing: What, why, and implications for design automation. In *Proceedings of the Design Automation Conference (DAC)*. 610–615.

Garey, M. and Johnson, D. S. 1979. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

Geurts, W., Catthoor, F., Vernalde, S., and De Man, H. 1997. *Accelerator Data-path Synthesis for High-Throughput Signal Processing Applications*. Kluwer Academic Publishers. Boston, MA.

Grötschel, M., Lovász, L., and Schrijver, A. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1*, 169–197.

Huang, Z. and Malik, S. 2001. Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable data paths and optimized interconnection networks. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*. 735–740.

Lee, C., Potkonjak, M., and Mangione-Smith, W. 1997. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture (Micro '97)*. IEEE, Research Triangle Park, NC. 330–337. Benchmarks available at `http://cares.icsl.ucla.edu/MediaBench/`.

Mathur, A. and Saluja, S. 2001. Improved merging of data path operators using information content and required precision analysis. In *Proceedings of the Design Automation Conference (DAC)*. 462–467.

Moreano, N., Araujo, G., Huang, Z., and Malik, S. 2002. Datapath merging and interconnection sharing for reconfigurable architectures. In *Proceedings of the 15th International Symposium on System Synthesis*. 38–43.

Moreano, N., Araujo, G., and de Souza, C. C. 2003. CDFG merging for reconfigurable architectures. Tech. Rep. IC-03-18, Institute of Computing, University of Campinas SP, Brazil.

Nemhauser, G. L. and Wolsey, L. 1988. *Integer and Combinatorial Optimization*. Wiley, New York.

Papadimitriou, C. H. and Steiglitz, K. 1998. *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY: Dover Publications.

Schaumont, P., Verbauwhede, I., Keutzer, K., and Sarrafzadeh, M. 2001. A quick safari through the reconfiguration jungle. In *Proceedings of the Design Automation Conference (DAC)*. 172–177.

Schmit, H. et al. 2002. PipeRench: A virtualized programmable data path in 0.18 micron technology. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*. 63–66.

Shirazi, N., Luk, W., and Cheung, P. 1998. Automating production of run-time reconfigurable designs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 147–156.

Singh, H., Lee, M., Lu, G., Kurdahi, F., Bagherzadeh, N., and Filho, E. 2000. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers 49*, 5 (May), 465–481.

Werf, A. et al. 1992. Area optimization of multi-functional processing units. In *Proceedings of the 1992 International Conference on Computer-Aided Design (ICCAD)*. 292–299.

Wolf, W. 2001. *Computers as Components—Principles of Embedded Computing System Design*. Morgan Kaufmann, San Mateo, CA.

Received January, 2005; revised January, 2006; accepted Feburay, 2006