Reducing the complexity of ILP formulations for synthesis

Anne Mignotte Olivier Peyran
Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon
46, Allée d'Italie, 69364 LYON Cedex 07, France
Anne.Mignotte@ens-lyon.fr, Olivier.Peyran@ens-lyon.fr

Abstract

Integer Linear Programming (ILP) is commonly used in high level and system level synthesis. It is an NP-Complete problem (in general cases). There exists some tools that give an optimal solution for small ILP formulations. Nevertheless, these tools may not give solutions for complex formulations. In this paper, we present a solution to overcome the problem of complexity in ILP formulations. We propose a partitioning methodology based on a constraint graph representing all the constraints included in any ILP formulation. To direct the partitioning, the constraint graph nodes are grouped to represent Data Flow Graph (DFG) nodes. This reduced constraint graph can be used to partition any ILP formulation based on DFG. We illustrate this method on ILP formulation for scheduling. Results on ILP scheduling formulations are promising.

1 Introduction

Integer linear programming (ILP) is a widely used technique in high level and system level synthesis. Common applications are resolution of NP-complete problems. For instance, [5] uses ILP for throughput and latency optimization when algorithm/architecture matching, retiming and pipelining are considered simultaneously. ILP is also used for DSPs code generation and embedded systems. For instance, [14] gives a solution to the problem of code compaction with real-time constraints for processors offering instruction-level parallelism; [21] presents an ILP-based code placement method for embedded software to maximize hit ratios of instructions caches. ILP is also widespread for HW/SW partitioning [2, 11, 17]. In the field of system level synthesis, one can also cite [20, 1], which deal with the optimization of heterogeneous multiprocessors systems. Another example is [19], whose method generates a static task execution schedule along with the structure of the multiprocessor system and a mapping of subtasks to processors. A part of the previous problem, scheduling, is a very common application of ILP: [10] presents different formulations for different usual scheduling problems; [8], [13] propose solutions to the more general problem of performing scheduling and resource allocation simultaneously; [3] describes a methodology to solve a scheduling problem in a 3-dimensional design space, including the usual area and schedule length dimensions plus the clock length dimension, using module libraries.

In [15], we consider the problem of performing scheduling and operator type selection simultaneously. Operands can be represented in various number systems (redundant or non redundant ones). This leads to architectures with different type of arithmetics (*mixed arithmetics*), and the insertion of conversion nodes between redundant output and non redundant input of operators is needed. As conversions are "real" operations (in fact additions with carry propagation), the operator type selection has to be done while scheduling. A heuristic have been proposed [16], but the results were not convincing. Thus, we explored the ILP approach. Our ILP formulation has been tested using *LP_ SOLVE* (see table 1). The results are optimum, and the computation times remain small for small examples.

Nevertheless, ILP is a NP-complete problem, and our ILP solver could not find any solution for reasonable size examples, because of numerical instability. Complexity is the major drawback of ILP. There are solutions to overcome this problem by directly modifying the formulation. In the next section, we present some of these techniques. However, they usually do not largely decrease the complexity, and are always specific to each problem. Thus, some partitioning techniques have been proposed in the literature. However they usually do not take into account all the ILP constraints (resource constraints and dependency constraints) as well as the ILP size. In section 3, we propose a general solution which partitions the problem into several small ILP formulations, separately solved taking all the constraints into account. Section 4 discusses the results and extensions of this method.

2 Shortening the ILP formulation

In this section, we present solutions that are specific to our resource constraint scheduling problem using mixed arithmetic. More precisely, all these solutions have been tested on the 5^{th} order elliptic wave filter benchmark.

Our ILP formulation for scheduling is based on two main constraints: resource constraints, and data dependency constraints which are given by a DFG. The formulation is inspired by Hwang *et al.* one [10]. The variables and constants used are the following:

benchmarks	Nb equations	Nb variables	Nb operations	Nb cycles	CPU
2 way method	53	62	6	5	6.2 s
Fast Fourier Transform	51	78	7	4	3.6 s
Differential equation	99	132	11	5	17.6 s
3 way method	248	410	25	?	no sol.
3 way method classic	91	126	25	10	51.1 s
5^{th} order elliptic wave filter (EWF)	383	991	34	?	no sol.
5^{th} order EWF, $k = -1/0$	258	586	34	?	no sol.
5^{th} order EWF, $k = \pm 1$, red. outputs	287	665	41	?	no sol.
5^{th} order EWF classic	119	133	34	16	4 min 44.8 s

Table 1: ILP results using LP_SOLVE.

- $x_{i,j} = 1$ if operation o_i is scheduled into cycle j; otherwise, $x_{i,j} = 0$.
- T is the final number of cycles that we wish to minimize and N_t is the number of resources of type t.
- s is an overestimation of T.
- L_i (resp., S_i) is the latest (resp. earliest) possible time to schedule operation o_i. They are computed according the classical "As Late As Possible" and "As Soon As Possible"schedules considering that we have s cycles.

We have kept the same conventions and extended them to our specific problem: if o_i is a classical operation (addition, subtraction, multiplication...), it is also related to variable $x_{i,j}$, with $j \in [S_i, L_i]$. Our model inserts a virtual conversion after each operation. Therefore we need a new variable, $x_{k,j}$, representing the conversion. Our ILP formulation will not be detailed in this paper.

The complexity of an ILP formulation is the product of the number of equations by the number of variables. Concerning scheduling, these two values depend on the number of operations, n, and on the initial bound of the number of cycles, s (in our case, both grows in $O(s * n)$). The s value is a large overestimation, and considering the operation frames (difference between L_i and S_i) is more accurate.

To reduce the ILP formulation, the number of operations could obviously not be decreased. The only solution is to shorten the operation frames. It decreases the number of variables, and also the number of equations, as some resource constraints may disappear (if some juxtaposed operation frames become disconnected). Obviously, this does not guarantee optimality, as all the possible resulting schedules are not considered anymore. The problem is to find a good shortened frame (i.e an as small as possible frame containing the optimum schedule with an high probability).

A first method consists in producing an optimal linear programming solution using real variables. From this solution, we prune the ILP formulation by considering, for the operation possible schedules, the integer values that are close to the real-number solutions. However, a real-number solution is not an approximation of an integer solution. In some cases, this reduction can lead to a solution space which does not contain any integer solution. In opposite cases, the frame still remains too large to solve the instability problem.

Thus, we tried another technique: using an heuristic solution (for instance the list scheduling) to bound the frame of each operation, the frames are redefined as $T(i) \pm k$, where $T(i)$ is the scheduled cycle of operation o_i, and k is as small as possible. Nevertheless, the frames were still too large with $k = 1$, as shown table in 1. We even tried this method with frames equal to $\{T(i) - 1, T(i)\}$, and we did not obtain any solution either.

This solution is interesting when the initial operation frames are large. However, in such a case, reducing the frames will reduce the solution space, increasing the probability of "loosing" the optimal schedule. On the opposite, if operations have small mobility, the optimal schedule has an high probability of staying in the solution space, but the complexity can not be largely decreased, as time frames are already short. Therefore, we have explored another technique, which is addressed in the following sections, that consists in partitioning the problem in order to get several smaller ILP formulations, which are separately solved.

3 Partitioning

3.1 Partitioning methodology

The initial DFG is partitioned. Each partition can be considered as a separate design, and is scheduled using a separate ILP formulation. We obtain several optimal local schedules which are concatenated in order to obtain the final global schedule. The main difficulty is to find a partitioning algorithm, as there are two constraints to deal with: all the interdependencies between partitions and their size.

Some related methods of partitioning have already been proposed. For instance, Hwang experimented an approach [9], called "zone scheduling". It partitions the control steps into zones, and decides which operation will be scheduled into a zone and which one will

be "delayed" into the next zone. Their model can be turned into an optimal ILP scheduling, a list scheduling, or one in between. However, their goal is more to find better solutions than those achieved by list scheduling rather than finding near optimal solutions when ILP does not succeeds. Depuydt et al. have a solution based on clustering techniques [4]. It does not take into account the resource constraints but variable time frames to reduce the register cost.

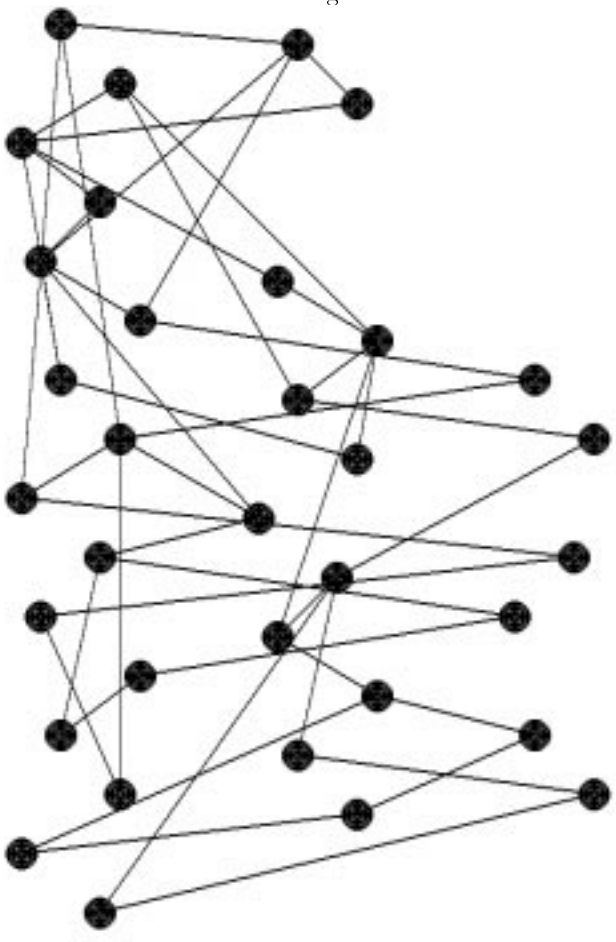

Figure 1: 5^{th} *order filter* data flow graph DFG

Therefore, we propose an **original and general approach**, based on the ILP formulation, which consists in partitioning the set of operations, each partition violating as few constraints as possible (either data dependency or resource ones) and being balanced in terms of ILP variables. Considering a simple DFG would not be satisfactory, as a DFG only reflects data dependency constraints (see for instance the $5^{th} order filter$' DFG in Fig. 1). Thus, our partitioning is based on a *reduced constraint graph* extracted from the ILP formulation, whose vertices represent operations and edges represent constraints between operations. Performing minimum edge cut partitioning creates partitions with low constraint dependencies. As each partition leads to an optimal partial schedule, the final schedule, obtained by the concatenation of the partial schedules, is a very good approximation of the optimal one.

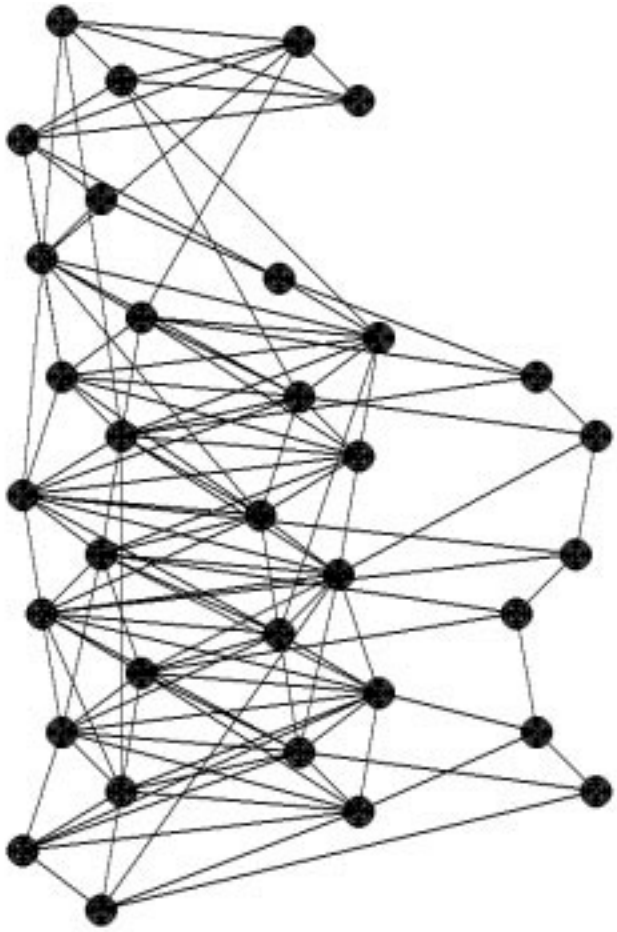

Figure 2: 5^{th} *order filter* reduced constraint graph RCG

A $k-$partition

In order to determine the best value of k, one could iteratively try several decreasing values until it leads to an infeasible ILP formulation: starting with an $n-$partition, if all the partitioned formulations succeed to find a solution, try with a $(n-1)-$partition, and so on. This solution is realistic, as the computation times are largely decreased using the partitioning method (see below, and particularly the comparison between benchmarks which had a solution with the whole formulation and their partitioned solution). Moreover, one usually knows the approximate number of variables (N_{max}) that his solver can handle. Thus, an efficient solution is to determine directly, as a starting value, a number of partition which has a good chance to be the optimal (for example it would be $\left\lceil \frac{\sum_i L_i - S_i + 1}{N_{max}} \right\rceil$ with the scheduling problem).

In our specific scheduling problem [15], we have to

benchmarks	Nb equations	Nb variables	Nb operations	NB cycles	CPU time
2 way method partition 1	30	36	4	3	1.1s
2 way method partition 2	24	24	3	2	0.2s
Fast Fourier Transform partition 1	32	50	4	1	0.4s
Fast Fourier Transform partition 2	45	54	6	3	0.5s
Differential equation partition 1	63	77	10	3	0.7s
Differential equation partition 2	56	76	8	3	1.4s
3-way method partition 1	128	187	15	6	6 hr 34 min
3-way method partition 2	102	118	11	5	12 min 56 s
3-way meth. Classic partition 1	57	68	16	6	5.7s
3-way meth. Classic partition 1	33	33	9	4	0.9s
$5^{th}O$ Elliptic wave filter partition 1	121	204	15	7	1 hr 46 min
$5^{th}O$ Elliptic wave filter partition 2	135	195	17	4	2 hrs 58 min
$5^{th}O$ Elliptic wave filter partition 3	128	175	18	5	2hrs 20 min
$5^{th}O$ EWF Classic partition 1	65	65	17	10	7.9s
$5^{th}O$ EWF Classic partition 2	51	68	17	7	2.0 s

Table 2: ILP results using LP_SOLVE after ILP based partitioning.

deal with the number systems used for representing operands: the ILP formulation aims at finding the best (redundant or non redundant) encoding. As the outputs of a partition become the inputs of the following partition, and as the encoding of the primary inputs influence the scheduling, this implies that all the precedent partitions must have already been scheduled. This communication problem is also crucial when dealing with resource allocation and scheduling: the way operators share registers in a partition highly influences the resource sharing in the following partitions. Thus, these informations have to be brought to the other partitions. Therefore, it seems more efficient to make successive bi-partition after each local schedule rather than an initial global $k-$partition.

Considering the data flow graph $DFG = (V, E)$ such that the ILP formulation related to DFG could not be solved, The $k-$partition of $DFG = (V, E)$, with k as small as possible, defines k data flow graphs $DFG_i = (V_i, E_i)$, such that $V = V_1 \cup V_2 \cup ... \cup V_k$ and $V_i \cap V_j = \emptyset$ if $i \neq j$.

We are dealing with partition i
DFG_1 to DFG_{i-1} have been scheduled.
A reduced constraint graph, RCG_i is built:
$RCG_i = (RCV_i, RCE_i, W_i)$, where
$RCV_i = V \setminus \{V_1 \cup V_2 \cup ... \cup V_{i-1}\}$.
RCE_i andW_i are defined in the next section.
W_i aims at representing the "ILP size".
RCG_i is bipartitioned according to the ILP sizes:
$S(RCG_i)$ refers to the "ILP size" of graph RCG_i.
Thus, $S(RCG_i) = \sum_{v_j \in RCV_i} w(v_j)$,
where $w(v_j)$ is the weight of v_j in RCG_i
We create a bi-partition of RCG_i **with minimum edge cut**, of respective size,
$\frac{S(RCG_i)}{k-i}$ and $\frac{(k-i-1).S(RCG_i)}{k-i}$.

Partitioning with minimum cut is known to be a NP-complete problem [7], but there are some efficient heuristics [6, 12]. Our method has been implemented using the Fiduccia and Mattheyses heuristic, which is an improvement of Kernighan and Lin Min-Cut heuristic.

As edges represent constraints, the idea behind min-cut partitioning is to minimize the constraint violations. Thus, this algorithm is efficient if RCG is a good representation of the different constraints. We will now address the problem of the reduced constraint graph definition.

3.2 Reduced constraint graph

The previous method is not specific to any particular problem. It has been tested with our scheduling problem, but it could also be used to any other ILP based problem. Therefore, the definition of the reduced constraint graph, RCG, should not depend on any particular problem. Nevertheless, there are a few observations that the graph should match.

- The input of the ILP formulation is a DFG. Thus, the graph vertices represent operations of the DFG.
- Our goal is to create partitions whose ILP formulations would take comparable computation times. As every operation does not have the same influence over the ILP computation time, the vertices must have a weight $w(e_j)$ reflecting their influence over this computation time.
- Edges must represent **any** constraint. In fact, this solution should not even be related to a scheduling problem, but, more generally, to the problem of resolving large ILP formulations.

We based our solution on a graph used by Pan, Dong and Liu [18] to solve a problem of constraint reduction in symbolic layout compaction: from a set, S, of linear programming constraints of the form $x_i - x_j \geq b$ (we will say that $x \in cn$ if variable x appears

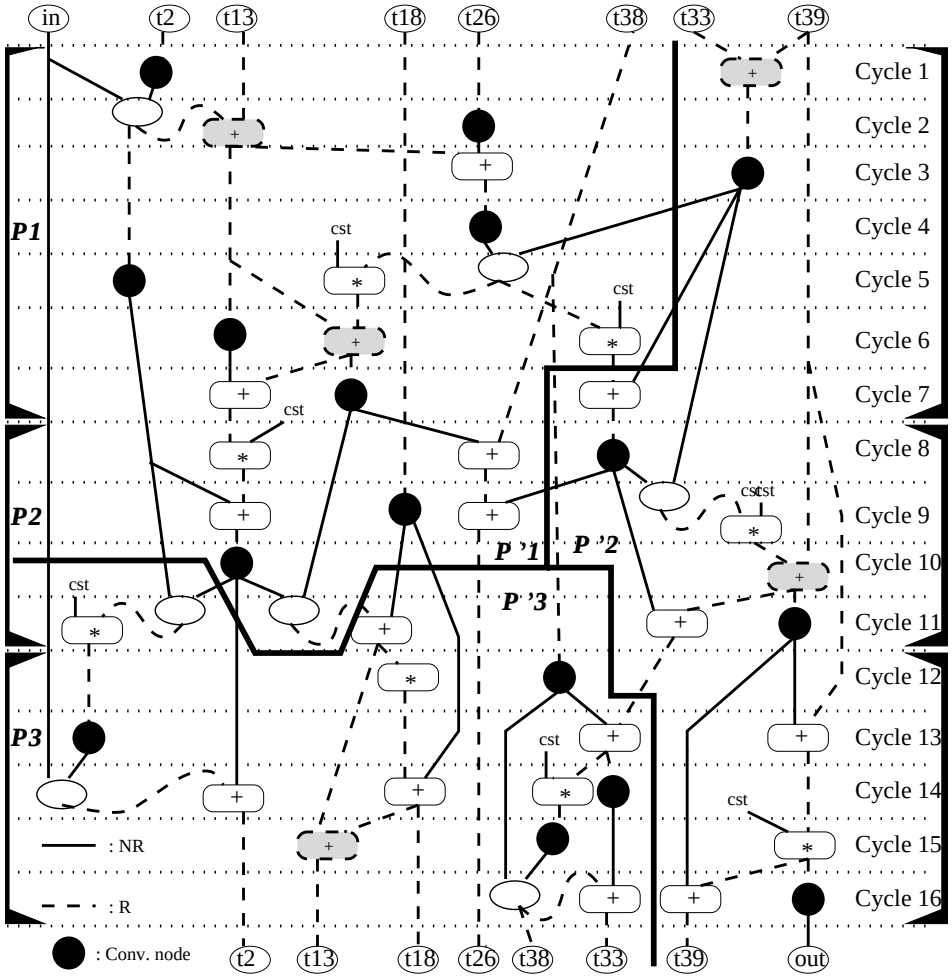

Figure 3: Result of the 5^{th} *order filter* scheduling after a 3-partition (partitions are P1, P2 and P3 with a constraint graph partition, and P'1, P'2 and P'3 with a DFG partition).

in constraint cn), they create a directed graph, $G = (V, E)$, such that each variable x_i which appears in S is related to a vertex $v_i \in V$, and such that each constraint $cn : x_i - x_j \geq b, cn \in S$ is related to an edge $e : v_i \mapsto v_j, e \in E$, of weight b. From this graph, they solve a problem of subgraph reduction (i.e finding an equivalent graph with less edges).

We have extended this representation to ILP: from a constraint $\sum_{i,j} a_{i,j}.x_{i,j} \geq b$, where $x_{i,j}$ represents a variable related to operation o_i, we construct a complete graph $CG = (CV, CE)$, where a vertex $v_{i,j} \in V$ is related to the ILP variables $x_{i,j}$. It makes a constraint graph **of variables**. From this graph, we perform a coarsening phase which creates clusters $C_i = \{x_{p,q} | p = i\}$ (C_i contains all the variables related to operation o_i), in order to get a graph **of operations**. This defines a *reduced constraint graph* $RCG = (RCV, RCE, W)$ as follows:

From a set, ILP, of ILP constraints,

- $\forall i$, if $\exists cn \in ILP, j \in N | x_{i,j} \in cn$, we construct a vertex $v_i \in RCV$, weighted by $w(v_i) = Card_j(x_{i,j})$.
- If $\exists cn \in ILP, i_1, i_2, j_1, j_2 \in N | i_1 \neq i_2, x_{i_1,j_1} \in cn, x_{i_2,j_2} \in cn$, we construct an edge $e_{1,2} \in RCE$ between v_{i_1} and v_{i_2}.

This definition fits the previous observations, as RCG is an operation graph, whose vertices are weighted by the number of ILP variables linked to an operation, which has a great influence over the ILP computation time. Furthermore, the edges are constructed by each ILP constraint, explicitly and equally treated. This method could even be used for other problems than scheduling ones. Fig. 2 shows the reduced constraint graph for the 5^{th} *order filter* design.

4 Results

We have tried this solution with the *5th order elliptic wave filter*, using a 3-partition, for our scheduling

and operator type selection problem. The *5th order filter* 3-partition of Fig 3 has been obtained. It defines partitions P1, P2 and P3 and the resulting schedule is optimal for our specific problem. Compared to our partitions, the partitions P'1, P'2 and P'3 were obtained with a 3-partition based upon the DFG. The DFG based partition could not be exploited, as the partition is not "horizontal". This shows that, even if our method does not introduce any information specific to a scheduling method, it gives exploitable solutions.

Examples that did not need partitioning have also been tested, in order to get an idea of the degradation compared to the optimal. Only two examples had more cycles than the optimal: the differential equation design and the $5^{th} order filter$ EWF one, in its classical version, without redundant operators and operator type selection (3+3 instead of 5 and 10+7 instead of 16). However, in each case, the extra cycle is due to the junction between the two partitions, and a method closed to one presented in section 2 manages to find the optimal schedule. This solution consists in making a formulation of the whole design where the frames of the second partition operations are reduced to $\{T(i)-1, T(i)\}$ (where $T(i)$ is the schedule obtained by the partitioning solution), whereas first partition operations are fixed to the partitioning solution schedule. This solution allows to use the available resources of the last cycle of partition 1, and can be considered as a "smart" concatenation. All the others examples managed to find the optimal schedule. Besides, on every benchmark, the CPU time is largely decreased (see table 2 compared to table 1). This is particularly impressing with large examples (10 and 28 times faster for the 3-way method and the $5^{th} order filter$).

We have experimented this method with other examples, and a problem occurred when the operation time frames are too large. In such a case, the resource constraints become too numerous compared to data dependency ones, and the partitions obtained can not be exploited (for example, additions and multiplications are separated without regards of the precedence). This problem disappears if we manage to reduce the initial operation frames. A better solution would be to have a balanced influence of the different kinds of constraints. Therefore we plan to introduce a weight on the constraint graph edges. We are currently experimenting weighting functions which would balanced the influence of resource and data dependency constraints, even with large operation frames. Of course, this solution is not general anymore, and becomes specific to a scheduling problem.

5 Conclusion

To overcome the problem of ILP complexity, we have defined a partitioning methodology which constructs a reduced constraint graph $RCG = (V, E, W)$ to successively bi-partition an ILP formulation. V represents the DFG nodes. E is built according to all the ILP constraints. W represents the ILP size.Therefore, this graph can be used to partition any ILP formulation based on a DFG, which are common in high level and system level synthesis. This method have been tested for the scheduling problem. It seems that a better partition can be obtained by weighing the edges of RCG according to the type of constraints (resource sharing or data dependency). We are currently working on this weight definition.

References

[1] A. Bender. Milp based task mapping for heterogeneous multiprocessor systems. In *proceedings of the European Design Automation Conference 96 (EuroDAC'96)*, 1996.

[2] N. N. Binh, M. Imai, and A. Shiomi. A new hw/sw partitioning algorithm for synthesizing the highest performance pipelined asips with multiple identical fus. In *proceedings of the European Design Automation Conference 96 (EuroDAC'96)*, 1996.

[3] S. Chaudhuri, S. A. Blythe, and R. A. Walker. An exact methodology for scheduling in a 3d design space. In *Proceedings of the International Symposium on System synthesis (ISSS 95)*, 1995.

[4] F. Depuydt, G. Goossens, and H. De Man. Clustering techniques for register optimization during scheduling preprocessing. In Louise Goto, Satoshi; Trevillyan, editor, *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 280–283, Santa Clara, CA, November 1991. IEEE Computer Society Press.

[5] Y.G DeCastelo-Vide e Souza, M.Potkonjak, and A. C. Parker. Optimal ilp-based approach for throughput optimization using simultaneous algorithm/architecture matching and retiming. In *Proceedings of the 32nd Design Automation Conference*, 1995.

[6] C.M. fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, 1982.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Ed. W.H. Freeman and Co., New-York, 1979.

[8] C. H. Gebotys and M. I. Elmasry. Optimal synthesis of high-performance architectures. *IEEE Journal of Solid-State Circuits*, 27(3), 92.

[9] C-T. Hwang and Y-C. Hsu. Zone scheduling. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 12(7):926–934, 1993.

[10] C-T. Hwang, J-H. Lee, and Y-C. Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transaction on Computer Aided Design*, 10(4):464–475, April 1991.

[11] I. Karkowski and R.H.J.M Otten. An automatic hardware/software partitioner based on the possibilistic programming. In *proceedings of the European Design and Test Conference 96 (EDTC'96)*, 1996.

[12] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49, 1970.

[13] B. Landwehr, P. Marwedel, and R. Dömer. Oscar: Optimum simultaneous scheduling allocation and resource binding based on integer programming. In *proceedings of the EuroDAC'94*, 1994.

[14] R. Leupers and P. Marwedel. Time-constrained code compaction for dsps. In *proceedings of International Symposium on system synthesis 95 (ISSS'95)*, 1995.

[15] A. Mignotte and O. Peyran. Scheduling using mixed arithmetic: an ilp formulation. In *proceedings of the European Design and Test Conference 1997*, 1997.

[16] Anne Mignotte, Jean-Michel Muller, and Olivier Peyran. Mixed arithmetics: Introduction and design structure. In *proceedings of MPCS'96*, 1996.

[17] R. Niemann and P. Marwedel. Harware/software partitioning using integer programming. In *proceedings of the European Design and Test Conference 96 (EDTC'96)*, 1996.

[18] Peichen Pan, Sai-Keung Dong, and C.L. Liu. Optimal graph constraint reduction for symbolic layout compaction. In *Proceedings of the 30th Design Automation Conference*, 1993.

[19] S. Prakash and A. C. Parker. Sos: Synthesis of application-specific heterogeneous multiprocessor systems. *Journal of Parallel and Distributed Computing*, 16:338 – 351, 1992.

[20] M. Schwiegershausen and P. Pirsch. A system level design methodology for the optimization of heterogeneous multiprocessors. In *proceedings of the International Symposium on System Level Synthesis 95 (ISSS'95)*, 1995.

[21] H. Tomiyama and H. Yasuura. Otpimal code placement of embedded software for instruction caches. In *Proceedings of the European Design and Test Conference (EDTC'96)*, 1996.