The importance of Relaxations and Benders Cuts in Decomposition Techniques: Two Case Studies

Alessio Guerri (student) and Michela Milano (supervisor)

DEIS, University of Bologna
V.le Risorgimento 2, 40136, Bologna, Italy
{aguerri, mmilano}@deis.unibo.it

When solving combinatorial optimization problems it can happen that using a single technique is not efficient enough. In this case, simplifying assumptions can transform a huge and hard to solve problem in a manageable one, but they can widen the gap between the real world and the model. Heuristic approaches can quickly lead to solutions that can be far from optimality. For some problems, that show a particular structure, it is possible to use decomposition techniques that produce manageable subproblems and solve them with different approaches. Benders Decomposition [1] is one of such approaches applicable to Integer Linear Programming. The subproblem should be a Linear Problem. This restriction has been relaxed in [4] where the technique has been extended to solvers of any kind and called Logic-Based Benders Decomposition (LBBD). The general technique is to find a solution to the first problem (called Master Problem (MP)) and than search for a solution to the second problem (Sub-problem (SP)) constraining it to comply with the solution found by the MP. The two solvers are interleaved and they converge to the optimal solution (if any) for the problem overall. When solving problems with a Benders Decomposition based technique, a number of project choices arises:

- At design level, the objective function (OF) depends either on MP or SP variables, or both. This choice affects the way the two solvers interact.
- Generation of Benders Cuts; Benders Cuts are constraints, added to the MP model once a SP as been solved, that remove some solutions.
- Relaxation of the SP; to avoid the generation of trivially infeasible MP solutions, some relaxations of the SP should be added to the MP model.

We focus on two particular problems, (1) the allocation and scheduling problem on Multi-Processor System-on-Chip (MPSoC) platforms (ASP), we investigated in [2], and (2) the dynamic voltage scaling problem on energy-aware MPSoC (DVSP), we investigated in [3]. These are very hard problems and they have never been solved to optimality by the system design community. We used LBBD to solve the problems.

The aim of this paper is to show the importance of the relaxations and Benders Cuts in terms of their impact on the search time and on the number of times the two solvers iterate. In addition, we claim that, in general, a tradeoff between the complexity of the cuts and relaxations introduced and their impact on the number of iterations must be found.

1 Benders Decomposition

The Benders Decomposition (BD) technique works on problems where two loosely constrained sub-problems can be recognized. Let us consider a problem modelled using two sets of variables x and y. The Benders Decomposition technique solves to optimality the master problem (MP) involving only variables x, producing the optimal solution $\bar{x}$, then it solves the original problem where the variables x values are fixed to $\bar{x}$, namely the sub-problem (SP). Depending on the objective function (OF), two cases can appear: (i) if the OF depends only on variables x, the SP is simply a feasibility problem; if $\bar{x}$ is a feasible solution for the SP, it is the optimal solution for the original problem, otherwise the SP must communicate a no-good saying that $\bar{x}$ is not feasible and another one must be found; (ii) if the OF depends on both x and y variables (or only on y variables), the MP finds an optimal solution w.r.t. its OF (or feasible if the OF depends only on y), $\bar{x}$, then passes the solution to the SP and, when the SP finds an optimal solution w.r.t. its OF, the SP must tell the MP that the solution found is the optimal one unless a better one can be found with a different assignment to variables x. In both cases, the two solvers are interleaved and they converge to the optimal solution (if any) for the problem overall.

To avoid the inefficient *generate and test* behaviour of the MP and the SP interaction it is useful to add to the MP a relaxation of the SP. The relaxation provides a lower bound (or an upper bound if it is a maximization problem) on the SP optimal solution.

The original BD technique models both the MP and the SP using Integer Linear Programming (IP), while LBBD [4] extends BD to cope with any solver. In [5] LBBD is applied to planning and scheduling problems. A set of activities must be assigned and scheduled on a given set of homogeneous facilities. The allocation master problem is modelled using an IP approach, while the scheduling sub-problem is modelled using Constraint Programming (CP). Once the allocation problem is solved, the scheduling part becomes easier since the scheduling problem does not contain alternative resources. Precedence constraints are posted only among activities allocated to the same facility, so the scheduling SP can be decomposed in a number of simpler one machine scheduling problems, one for each facility. In both our problems the scheduling does not decompose since precedence constraints link tasks that possibly run on different processors.

2 Problem Description

We describe here the two problems we faced in [2] and [3].

Problem 1: Allocation and scheduling problem on a MPSoC (ASP)

- Given a set of tasks $t_1 \dots t_n$, with duration $d_1 \dots d_n$ and memory requirements $s_1 \dots s_n$ for the internal state, $pd_1 \dots pd_n$ for the program data and $c_1 \dots c_n$ for communication,
- given precedences and communications among tasks, and realtime constraints imposing deadlines on tasks and processors,

- given an MPSoC platform [7], where a set of homogeneous processors $p_1 \dots p_m$ each with a local memory slot, a system bus and a remote memory are integrated on the same chip,
- $\diamond$ find an allocation of tasks to processors and of memory requirements to storage devices such that the total communication on the system bus is minimized. We have a contribution to the OF each time a memory requirement is allocated on the remote memory and each time two communicating tasks execute on different processors.

Problem 2: Allocation, scheduling and voltage selection problem on an energy-aware MPSoC (DVSP)

- given an energy-aware MPSoC platform [6], where a set of homogeneous processors able to change their frequency and a system bus are integrated on the same chip,
- Given a set of tasks $t_1 \dots t_n$, each annotated with a tuple of durations $\{d_{11} \dots d_{1f}\} \dots \{d_{n1} \dots d_{nf}\}$ (one for each processor speed) and a communication requirement $c_1 \dots c_n$,
- given precedences and communications among tasks, and realtime constraints imposing deadlines on tasks and processors,
- given time and energy overhead for a processor to switch from a frequency to another,
- $\diamond$ find an allocation of tasks to processors and of frequency to task executions such that the total power consumption is minimized. We have a contribution to the objective function each time an activity (task or communication) is performed and each time two activities running at different speeds are scheduled one just after the other on the same processor.

In both problems, we model and solve the allocation using an Integer Programming approach, while we use Constraint Programming to solve the scheduling SP. We can immediately see that the main difference between the ASP and the DVSP concerns the objective function. In the ASP, when the allocation is done, we know all the contributions to the objective function and thus the SP is simply a feasibility problem. In the DVSP instead the OF depends on both the MP and the SP.

3 Improving the models

3.1 Generation of Logic-based Benders cut

In the following we describe the Benders Cuts used.
ASP: A no-good is generated when the optimal solution of the MP is not feasible for the SP. We investigated two no-goods.

– We have variables X_{ij} that assume the value 1 if task i is allocated to processor j, 0 otherwise. The no-goods impose that for each set of tasks S_p allocated to a processor p, they should not be all reassigned to the same processor in the next iteration. The resulting no-good is $\sum_{p=1}^{m} \sum_{i \in S_p} X_{ip} < n$.

– The cuts described above remove only complete solutions. It is possible to refine the analysis and to find tighter cuts that remove only the allocation of tasks to bottleneck resources. So, when a SP failure occurs, we solve a one machine scheduling for each processor p considering constraints involving only tasks running on p. For each processor p where the problem is infeasible, we generate the cut $\sum_{i \in S_p} X_{ip} < |S_p|$. Finding this cut is a NP-hard problem, but we will show experimentally when it pays off.

DVSP: Here the OF depends on both MP and SP. If there is no feasible schedule given an allocation, the cuts are the same computed for the ASP. If the schedule exists we have to produce a cut stating that the one just computed is the optimal solution unless a better one exists with a different allocation. These cuts produce a lower bound on the setup of single processors. The cuts can therefore be of two types:

– We have variables X_{tpf}, taking value 1 if task t executes on processor p at frequency f. Let us consider J_p the set of couples (Task, Frequency) allocated to processor p. No-goods are the following: $\sum_{(t,f) \in J_p} X_{tpf} < |J_p|, \forall p$.
– Suppose a SP solution has an optimal setup cost $Setup^*$. It is formed by independent setups, one for each processor $Setup^* = \sum_{p=1}^{m} Setup_p^*$. We have a bound on the setup LB_{Setup_p} on each processor and therefore a bound on the overall setup $LB_{Setup} = \sum_{p=1}^{m} LB_{Setup_p}$. The constraints introduced in the master problem are: $Setup_p \geq LB_{Setup_p}$, and $LB_{Setup_p} = Setup_p^* - Setup_p^* \sum_{(t,f) \in J_p} (1 - X_{tpf})$, where J_p has the same meaning introduced above.

The cuts described remove only one allocation. Indeed, we have also produced cuts that remove some symmetric solutions.

3.2 Relaxation of the subproblem

In the MP models, deadlines are not taken into account, so the simplest kind of relaxation is based on the tasks execution times. The sum of the execution times of all the activities (tasks and communications) allocated to the same processor must not exceed the deadline. The deadline constraint can still be violated during the scheduling, but a huge number of infeasible solutions is surely cut.

In the **DVSP** this procedure can be improved by adding other relaxations expressing bounds on the setup cost and setup time in the master problem based only on information derived from the allocation. Let us consider, for each processor, the set of frequencies appearing at least once. A bound on the sum of the energy spent during the frequency switches can be computed as follows: let us introduce in the model variables Z_{pf} taking value 1 if the frequency f is allocated at least once on the processor p, 0 otherwise. Let us call E_f the minimum energy for switching to frequency f, i.e. $E_f = min_{i, i \neq f}\{E_{if}\}$. $Setup_p \geq \sum_{f=1}^{M} (Z_{pf} E_f - max_f\{E_f | Z_{pf} = 1\})$. This bound helps in reducing the number of iterations between the master and the subproblem. Similarly, we can compute a bound on the setup time to tighten the constraints involving deadlines described above.

4 Experimental Results

We have generated 500 DVSP and 400 ASP realistic instances, with the number of tasks varying from 7 to 19 and the number of processors from 3 to 10. We consider applications with a pipeline workload. We assume for the DVSP that each processor can run at three different frequencies. All the considered instances are solvable and we found the proved optimal solution for each of them. Experiments were performed on a 2.4GHz Pentium 4 with 512 Mb RAM. We used ILOG CPLEX 8.1, ILOG Solver 5.3 and ILOG Scheduler 5.3 as solving tools.

4.1 Algorithm performances

In [2] and [3] we compared the hybrid approaches with pure approaches modelling the problem as a whole using only IP or CP. For the ASP we found that the pure approaches search times are order of magnitude higher w.r.t. the hybrid, while for the DVSP the pure approaches are not able to find even a feasible solution within the time limit. In this section we will show the effectiveness of the cuts used. We consider ASP and DVSP instances with task graphs representing a pipeline workflow. Note that here, since we are considering applications with pipeline workload, if n is the number of tasks to be allocated, the number of scheduled tasks is n^2, corresponding to n iterations of the pipeline. Results are summarized in Table 1 for the ASP and in Table 2 for the DVSP. The first three rows contain respectively the number of tasks allocated and scheduled and the number of processors considered in the instances. The last two rows represent respectively the search time and the number of iterations. Each value is the mean over all the instances with the same number of tasks and processors. We can see that for all the DVSP instances the optimal solution can be found within four minutes and the number of iterations is typically low. For the ASP instances the optimal solution can be found within one minute and the mean number of iterations is very close to 1.

To show the effectiveness of the relaxations used for the DVSP we solved the instances considering either both or only one of the two relaxations described in 3.2. Table 3 shows the percentage of occurrence of a given number of iterations when solving the DSVP with different relaxations. Using both of them (row All) we can see that the optimal solution can be found at the first step in one half of the cases and the number of iterations is at most 5 in almost the 90% of cases. We tried to solve the problems using only one relaxation; rows Time and Bound show the results when considering only the relaxation on the deadlines and on the SP OF lower bound respectively. We can see that, for most of the cases, the number of iterations is higher than 10. In addiction, the search time on average rises up to 1 order of magnitude and, in the worst cases, the solution cannot be found within two hours.

To show the effectiveness of the cuts used for the ASP, we selected a hard ASP instance with 34 activities and we solved it with different deadline values, starting from a very weak one to the tightest one. Table 4 shows the number of iterations when solving these instances respectively without (row Base) and with

(row Advanced) the second kind of cuts described in 3.1 for descending deadline values (row Deadline). We can see that, when the number of iterations is high, the cuts reduce them notably. These cuts are extremely tight, but the time to generate them is one order of magnitude greater w.r.t. the time to generate the Base cuts, therefore they are helpful only on hard instances.

We tried to introduce tighter cuts and relaxations, but we experimentally see that the computation time increases. This is because the cuts and the relaxations complicate the model too much. In general, a tradeoff between the complexity of the cuts and the reduction in terms of iterations must be found.

Alloc	7	7	9	9	11	11	11	13	13	15	15	15	17	17	19	19	19
Sched	49	49	81	81	121	121	121	169	169	225	225	225	289	289	361	361	361
Procs	3	4	4	5	4	5	6	5	6	5	6	7	6	7	4	7	9
Time(s)	0,42	0,41	0,50	0,57	0,60	0,85	1,26	2,84	6,14	0,98	9,53	14,37	7,71	9,25	3,85	27,85	46,69
Iters	1,01	1,05	1,01	1,07	1,06	1,09	1,10	1,08	1,09	1,03	1,07	1,12	1,11	1,02	1,03	1,06	1,11

Table 1. Search time and number of iterations for ASP instances

Alloc	7	7	9	9	11	11	11	13	13	15	15	15	17	17	19	19	19
Sched	49	49	81	81	121	121	121	169	169	225	225	225	289	289	361	361	361
Procs	3	4	4	5	4	5	6	3	7	4	5	7	5	6	3	6	10
Time(s)	1,43	2,24	5,65	6,69	15,25	2,17	2,14	5,90	34,53	12,34	22,65	51,07	60,07	70,40	3,07	120,1	209,4
Iters	2,91	3,47	4,80	3,41	4,66	4,50	3,66	1,90	6,34	4,45	10,53	6,98	7,15	9,20	1,96	6,23	10,65

Table 2. Search time and number of iterations for DVSP instances

Iter	1	2	3	4	5	6	7	8	9	10	11+
All	50,27	18,51	7,11	4,52	4,81	2,88	2,46	2,05	1,64	1,64	4,11
Time	35,23	10,32	3,47	4,76	3,12	2,84	2,13	2,06	1,04	1,11	33,92
Bound	28,6	10,12	5,64	3,78	4,35	2,91	1,29	1,48	1,12	0,84	39,87

Table 3. Number of iterations distribution ratio with different relaxations

Deadline	1000000	647824	602457	487524	459334	405725	357491	345882	340218	315840	307465
Base	3	1	1	18	185	192	79	6	4	2	2
Advanced	3	1	1	6	16	23	17	4	3	3	2

Table 4. Number of iterations varying the deadline and with different Benders Cuts

References

1. J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
2. L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for mpsocs via decomposition and no-good generation. In *Proceedings of CP 2005*, pages 107–121, 2005.
3. L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation, scheduling and voltage scaling on energy aware mpsocs. In *Proceedings of CPAIOR2006*, 2006.
4. J. N. Hooker. A hybrid method for planning and scheduling. In *Procs. of the 10th Intern. Conference on Principles and Practice of Constraint Programming - CP 2004*, pages 305–316, Toronto, Canada, Sept. 2004. Springer.
5. J. N. Hooker. Planning and scheduling to minimize tardiness. In *Procs. of the 11th Intern. Conference on Principles and Practice of Constraint Programming - CP 2005*, pages 314–327, Sites, Spain, Sept. 2005. Springer.
6. M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini. Application-specific power-aware workload allocation for voltage scalable mpsoc platforms. In *2005 International Conference on Computer Design*, pages 87–93, 2005.
7. W. Wolf. The future of multiprocessor systems-on-chips. In *In Procs. of the 41st Design and Automation Conference - DAC 2004*, pages 681–685, San Diego, CA, USA, June 2004. ACM.