Constraint-driven System Partitioning*

M. L. López-Vallejo†, J. Grajal‡ and J. C. López**

† Dept. Ing. Electrónica
ETSIT, Univ. Politécnica Madrid
E–28040 Madrid, Spain
marisa@die.upm.es

‡ Dept. Señales, Sistemas y Radiocomunicación
ETSIT, Univ. Politécnica Madrid
E–28040 Madrid, Spain
jesus@gmr.ssr.upm.es

** Dept. Informática
ESI, Univ. Castilla La Mancha
E–13071 Ciudad Real, Spain
lopez@uclm.es

Abstract

This paper describes how optimization techniques can be applied to efficiently solve the constrained co-design problem. This is performed by the formulation of different cost functions which will drive the hardware-software partitioning process. The use of complex cost functions allows us to capture more aspects of the design. Besides, the appropriate formulation of this kind of functions has a great impact on the results that can be obtained regarding both quality and algorithm convergence rate. A strong point of the proposed formulation is its generality. Therefore, it does not depend on the problem and can be easily extended for considering new design constraints.

1. Introduction

Design constraints are a key issue to consider during the synthesis of complex systems. They must be taken into account not only in the specification phase, but also through the whole synthesis process. One of the main stages in the design cycle is the system partitioning phase. During this stage it is established which parts of the system specification will be implemented as software running in a standard processor and which parts will have to be implemented in special purpose hardware. Constraints should always be present in this phase.

To automate the partitioning process many algorithms and techniques have been developed in different co-design environments [1, 2, 3, 4, 5]. All these approaches can perfectly work within their own co-design environments, proving the suitability of the methods. Nevertheless, most of them do not consider design goals and constraints or just take into account some aspects in a particular way. In this work we have integrated system constraints and goals in the hardware-software partitioning process, obtaining valid results in a robust and efficient way. Even more, due to the generality of the proposed technique, this can be applied to any optimization procedure.

The partitioning problem can be classified as a classical combinatorial optimization problem, since it can be reduced to a discrete, but huge, space of solutions. The way a tool can perform an effective optimization is drastically influenced by the algorithm or procedure on which it is based. Two main issues must be considered in any optimization procedure:

- The control scheme of the algorithm (the optimization algorithm itself), which is normally well established (for instance, simulated annealing, genetic algorithms, etc.) and can be used for many different optimization problems.
- The cost function that guides the process, which must be provided by the user for the specific problem.

In this paper we propose a general formulation of the cost function for system partitioning which includes design constraints. The proposed cost function preserves the control scheme of the optimization algorithm, thus guaranteeing the right work of the procedure, while guiding the process always within the allowed design space. The paper is structured as follows. First, other approaches found in the literature will be examined. After that, the partitioning problem will be introduced. Next section is devoted to the cost function formulation. Finally, the results obtained will be analyzed and some conclusions will be drawn.

2. Related Work

Hardware-software partitioning, as optimization procedure, can be performed by means of very different algorithms, like adaptation of classical circuit partitioning algorithms (min-cut [4], clustering [6], etc.), general optimization methods (simulated annealing [2], tabu search [5], etc.). In this section we will review some of the previous works on hardware-software partitioning.

*This work was funded by the CICYT project TIC97-0928

The two first algorithms that tackled with the problem were presented by Cosyma and Vulcan. In Cosyma [2], a simulated annealing algorithm with fine granularity is applied, concentrating on the optimization of a single design attribute (the design execution time). Vulcan [1] also uses fine granularity with a partitioning algorithm based on iterative improvement, and extracts software blocks from an initial all-hardware solution considering the timing constraints.

Eles *et al.* [5] also apply simulated annealing and tabu-search to optimize the sum of several interesting objectives without considering constraints in the cost function. Thus, the control schedule of the algorithm must be modified, but the way the procedure works is not described.

A coarse grain constructive algorithm was developed in Ptolemy [3] extending list-based scheduling. Two important points characterize this algorithm: (1) it is able to adapt the objective function to global or critical measures, and (2) different hardware implementations are considered. Design constraints modify the algorithm control scheme.

An important work in functional partitioning has been done by Prof. Vahid [4], applying classical partitioning algorithms to hardware-software architectures (including the K&L heuristic, simulated annealing, clustering, etc.), but design constraints are not specifically considered.

Finally, the application of artificial intelligence techniques has proven its suitability when dealing with the system partitioning problem. The DDEL group [7] performs system partitioning using a genetic algorithm that includes hardware space exploration. The cost function considers hardware area and latency constraints to guide the process. But constraints can only be considered in a simple way. Later on we will analyze this formulation in depth.

3. Problem Description

The global information flow of the partitioning procedure presented here is depicted in Figure 1. The input to the partitioning process is an execution flow graph which comes from the initial system specification, described using high-level specification languages. This is a directed and acyclic graph where vertices stand for basic computation units and edges represent data and control dependencies. Thus, vertices can be large pieces of information (tasks, processes, etc.) or small ones (instructions, operations), following respectively a coarse or fine granularity approach.

Every graph node is labeled with additional information about the processes. In detail, these pieces of information for a node i are: hardware area (ha_i), hardware execution time (ht_i), software execution time (st_i), software size (ss_i) and the average number of times the task is executed (n_i). Edges have also associated a communication value ($t_{transf}(i,j)$) which is only considered in case the edge crosses the hardware-software boundary.

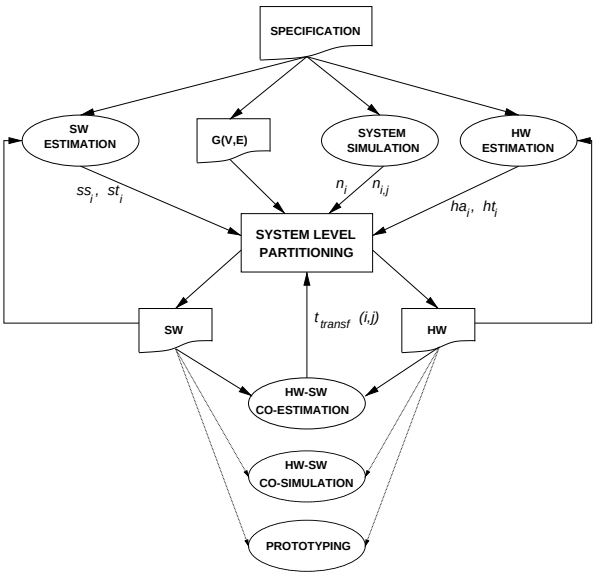

Figure 1: Partitioning Model.

As it is well known, system partitioning is clearly influenced by the target architecture onto which the hardware and the software will be mapped. Here the target architecture considered consists of one processor running the software, several dedicated co-processors (based on ASIC or FPGA) and shared memory accessed through a common bus. Interface modules are used to connect the processor and the co-processors to the bus. This is a general model in which for the sake of simplicity we have used just one hardware co-processor during our experiments. Hardware and software processes can execute concurrently in the standard processor and the application-specific co-processor.

The outcome of the partitioning tool is not only an assignment of blocks to hardware or software implementations, but also their starting and finishing time (computed by means of static scheduling) and the communication values produced in the interface.

The validity of the solution is measured by means of some *design-quality attributes* which must perfectly describe the solution. These attributes are normally design costs and performance parameters. In particular, we have used as quality attributes the required hardware area for the co-processor, A_p, the design latency, T_p, calculated by scheduling the system graph, and the required memory space, M_p. A design constraint is associated with each attribute. In our case, we will have the maximum available area, A, the maximum allowed execution latency T, and the maximum memory space, M. All these attributes and constraints will be used to characterize every solution by means of the cost function.

4. Cost Function Formulation

The main goal of a cost function must be to measure the quality of a given solution. For instance, in system level partitioning, the important information related to the process is:

- The global cost associated to the solution, provided by several attributes like hardware area, memory size, power consumption, timing related costs (design latencies, throughput, ...), etc.
- The design constraints (maximum available hardware area, maximum available memory) and goals (minimum precessing latency, etc.).

This information should be considered not only in a quantitative but also in a qualitative way, taking into account its nature. In this sense, design constraints must define the search space and cost issues must be used to characterize the quality of the solution. The type of cost related to the quality attributes should also be taken into account: fixed costs must be considered in a different way than variable costs. We propose a unified formulation for including all these issues.

Cost functions are usually expressed as a sum of weighted costs or objectives, as follows:

$$\mathcal{F}(\mathcal{P}) = \sum_i k_i \times \frac{C_i(\mathcal{P})}{N_i} \tag{1}$$

where $\mathcal{P}$ is the solution under evaluation, $C_i(\mathcal{P})$ is the value of a particular objective or cost, k_i is the weight factor applied to this objective in the solution evaluation and N_i is a normalization parameter needed to sum relative values. But this expression has a clear drawback: optimum values can be found out of the valid search space, since the violation of constraints either is not checked or it is performed modifying the control scheme.

We propose the use of different correction terms (one per constraint) to correctly guide the search in the design space. In the following, analytic expressions will be proposed for these correction terms, denoted $\mathcal{F}_C(C_i(\mathcal{P}))$:

$$\mathcal{F}(\mathcal{P}) = \sum_i k_i \times \frac{C_i(\mathcal{P})}{C_i} + \sum_i k_{c_i} \mathcal{F}_C(C_i(\mathcal{P})) \tag{2}$$

where as normalization parameter, N_i, has been used C_i, the i-th design constraint applied to the quality attribute $C_i(\mathcal{P})$ in a given partition $\mathcal{P}$, and k_{c_i} is the weight factor for every correction term.

Three different expression can be used to correct the objective function:

- Mean Square Error based Terms: which help the algorithm in finding a solution tuned to the constraints.
- Barrier Terms: which forbid the exploration of solutions out of the allowed search space.
- Penalty Terms: which strongly punish exploring solutions that would produce medium or large constraints overhead, but allow exploring points close to the constraint boundary.

4.1. Mean Square Error based Functions

A typical way of tuning system constraints is minimizing the mean square error that results in a quality attribute. The use of this kind of expressions tries to adjust the attributes to the design goals and constraints. These functions can be applied to the particular goals that should be completely fulfilled instead of minimized.

The general expression of this type of correction terms is the following:

$$\mathcal{F}_C(C_i, C_i(\mathcal{P})) = \sum_i \frac{(C_i(\mathcal{P}) - C_i)^2}{C_i^2} \tag{3}$$

where C_i is the i-th design constraint applied to the quality attribute $C_i(\mathcal{P})$, in a given partition $\mathcal{P}$. This correction term, added to the general expression shown in equation 2, works as follows. When a solution produces a value $C_i(\mathcal{P})$ different from the constraint value C_i, the cost associated with the solution is incremented with the deviation from this constraint in a quadratic way. Only when the attribute adopts a value tuned with the constraint will not contribute to the final solution cost.

This kind of correction term is suitable for those attributes with a *fixed cost*, as it is an FPGA-based coprocessor. It is a good policy to fully exploit all the resources the FPGA provides, since its maximum exploitation generally results in performance improvement. Besides, if the FPGA is not completely fulfilled, part of this resource is wasted and another resource will have to be used instead, increasing the design cost. The use of this kind of terms is strongly recommended for including these constraints, because the search space is considerably reduced and, consequently, the computation time of the algorithm is shorter. Nevertheless, it would not be suitable for those resources whose quality attributes have variable cost, as it is an ASIC area or timing measures.

Another function that works in a similar way as the mean square error based function is the one used in [7] with area and latency attributes:

$$\mathcal{F}_C(C_i, C_i(\mathcal{P})) = \sum_i \frac{|C_i(\mathcal{P}) - C_i|}{C_i} \tag{4}$$

but as stated before, this is not the best formulation for the optimization of the latency attribute.

4.2. Barrier Functions

A smart way of introducing design constraints in a cost function is the use of barrier methods [8]. Barrier functions provide a clear boundary in the design space, in such a way that the cost associated to a solution out of this region will be infinity. This is performed placing asymptotes in the constraint values. The analytical expression for these terms is:

$$\mathcal{F}_C(C_i, C_i(\mathcal{P})) = \sum_i \frac{1}{b[C_i(\mathcal{P}), C_i]} \tag{5}$$

where $b[C_i(\mathcal{P}), C_i]$ is the barrier function, which can be, for instance:

$$b[C_i(\mathcal{P}), C_i] = max\{0, [C_i - C_i(\mathcal{P})]\} \tag{6}$$

We have used this type of correction terms for very hard design constraints, because it ensures that the constraint will never be exceeded. Nevertheless its application contributes to the final cost even inside the allowed exploration regions, requiring the modification of the interpretation of the resulting cost.

4.3. Penalty Functions

When the system constraints are not too hard, the use of penalty functions [8] can be more suitable. Penalty functions do not contribute to the cost function when the solution is in the allowed search space. This kind of functions are not as restrictive as the barrier functions, since solutions around the border of the allowed exploration region can be accepted if they are really close. At this point, the weight factor k_{c_i} plays an important role.

The general expression commonly used for penalty functions is the following:

$$\mathcal{F}_C(C_i, C_i(\mathcal{P})) = \sum_i r^2[C_i(\mathcal{P}), C_i] \tag{7}$$

where it has been introduced the function $r[C_i(\mathcal{P}), C_i]$, which corresponds to:

$$r(C_i(\mathcal{P}), C_i) = \max\{0, \frac{[C_i(\mathcal{P}) - C_i]}{C_i}\} \tag{8}$$

Figure 2 clearly depicts the way the three different terms (mean square, penalty and barrier-based) contribute to the cost when considering a single quality attribute.

5. Analysis of Results

The performance of our approach guiding the hardware-software partitioning process has been tested by means of two classical algorithms: the Kernighan&Lin partitioning heuristic and the stochastic procedure simulated annealing. The K&L heuristic has required a strong modification to deal with the system partitioning problem [9]. The cost function used to guide the process has taken into account three quality attributes of the current partition that must be minimized simultaneously: the area of the hardware co-processor, A_P, the memory space required by the software running in the standard processor, M_P, and the global latency of the design, T_P (computed by static scheduling). The resulting cost function is:

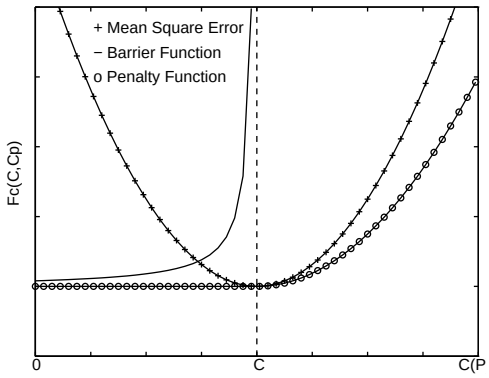

Figure 2: Comparison of the three correction terms: quadratic, penalty-based and barrier-based.

$$\mathcal{F}(\mathcal{P}) = k_A \times \frac{A_p}{A} + k_T \times \frac{T_p}{T} + k_M \times \frac{M_p}{M} + k_{c_A}\mathcal{F}_C(A, A_p) + k_{c_T}\mathcal{F}_C(T, T_P) + k_{c_M}\mathcal{F}_C(M, M_P) \tag{9}$$

where A, T and M are respectively the area, latency and memory constraints.

The following weight factors have been applied in all the tests: $k_A = 0.3$, $k_T = 0.4$, $k_M = 0.3$ and $k_{C_i} = 150$. These factors allow the designer to put emphasis on the desired design attribute. In out tests, the time goal is slightly emphasized. If these factors are appropriately chosen, they can also help in the result interpretation process. We have chosen $\sum_i k_i = 1$, being consequently $\mathcal{F}(\mathcal{P}) = 1$ a figure of merit, since (1) constraint overheads will produce cost values much greater than 1 (due to the weight factor $k_{C_i} = 150$), (2) attribute values tuned to the constraints will produce costs close to 1, and, (3) in the case the solution could be optimized (penalty terms) the cost value will be lower than 1.

Both algorithms have been applied to a set of examples provided by V. Srinivasan [7] using two cost functions: the first one is composed of three penalty terms and the other is based on mean square error based functions. Space requirements preclude us to present all the results. We will only

Examples		*Constraints*			*Penalty Method*				*Square Error Method*			
Name	#	A	T	M	A_p	T_p	M_p	Cost	A_p	T_p	M_p	Cost
LU	9	50000	200	12000	48898	126	1367	0.5796	46167	203	11434	2.2178
FFT	15	60000	400	25000	58676	267	8213	0.6589	60309	412	24077	1.3459
DCT	9	16000	5000	100000	13720	3384	66955	0.7288	16464	4455	100785	2.8850
DCT16	36	10000	15000	1000000	6552	15080	813591	0.5824	9828	15460	897901	2.7255
Laplace	9	20000	250	16000	16059	251	8997	0.8136	21627	259	15699	2.2733
Mean	9	60000	300	20000	55241	285	14383	0.8719	60633	304	20076	1.0552
Reg	8	2200	3000	25000	2167	2891	8323	0.6479	2407	2590	24674	5.1248
Sobel	20	60000	500	20000	59362	418	21384	0.7290	59267	504	19978	1.0314

Table 1: Results obtained after executing simulated annealing with several examples applying penalty and mean square error methods (weights $k_A = 0.3$, $k_T = 0.4$, $k_M = 0.3$ and $k_{C_i} = 150$).

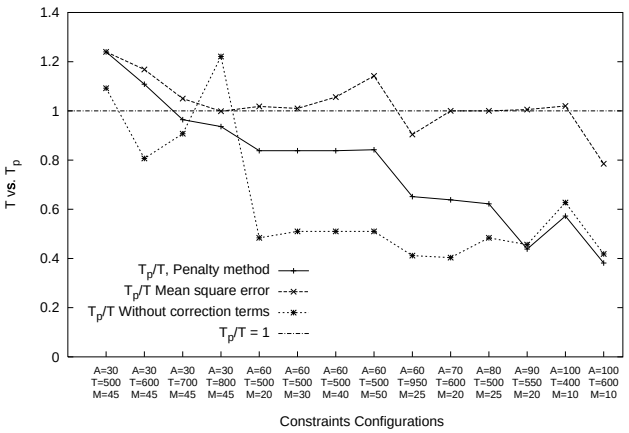

Figure 3: Latency attribute (T_p) vs. latency constraint T for the three functions under study.

comment the results obtained with simulated annealing.

The results obtained with a given set of constraints for this algorithm are shown in Table 1. It can be clearly seen how the mean square method adjusts the attributes to their constraints, while the penalty method optimizes the design attributes. This fact is clearly shown in the cost value associated to the different solutions. For the penalty based method, cost values are always lower than 1, and thus the solution found has largely fulfilled design constraints. The mean square error based method obtains values around one when the design constraints are tuned. It is very difficult to adjust three design constraints when the solution space is discrete. This is the reason why only two examples (Mean and Sobel) present $\mathcal{F}(\mathcal{P}) \approx 1$, while the rest of examples have cost values larger than 1.

A medium size example with 20 tasks (Sobel edge detection algorithm) will be analyzed in depth. Figure 3 illustrates some of the results obtained, plotting the latency attribute T_p related with its corresponding constraint, T. In this figure the solution provided by the cost function without correction terms seems to be the best, because it has the lowest latency values. However, this is obtained while other constraints are not met, what must be forbidden in any case. It can also be seen how results from mean square terms are around 1, while penalty based values try to minimize the latency constraint.

For the penalty based cost function, it can be seen that constraints can be met for all sets except the two first cases, in which time and area constraints are incompatible. We should remark here that the algorithm always tries to use as much hardware as possible, since the time weight is greater than the others. The algorithm, which has no knowledge of the problem it solves, "discovers" that latency can be minimized using more hardware. For the mean square error based function, the cost results show how difficult it is to tune all the constraints.

Different kind of correction terms can be combined in a single cost function. We have tried several combinations, each one trying to reflect a different system partitioning problem. For example, if the design latency must be perfectly adjusted, a correction term based in the minimization of the mean square error of the latency attribute must be used. If penalty terms are used with the other design constraints their corresponding quality attributes will be optimized. The cost function can be expressed as:

$$\mathcal{F}(\mathcal{P}) = k_A \times \frac{A_p}{A} + k_T \times \frac{T_p}{T} + k_M \times \frac{M_p}{M} + k_C \times [r^2(A, A_p) + (\frac{T_p - T}{T})^2 + r^2(M, M_p)] \quad (10)$$

Applying this cost function (called *mixed method*) to the system partitioning problem the results shown in Table 2 are obtained. A graphical interpretation of these results appears in Figure 4.

As a result of the experiments, it has been observed that the execution time is much shorter for the mean square correction terms. This is due to the reduced search space considered in this case. The results obtained with the Kernighan&Lin heuristic are very close to the ones presented here.

Constraints			*Penalty Method*					*Mixed Method*				
A	T	M	A_p	T_p	M_p	$Cs.$	$CPU(s)$	A_p	T_p	M_p	$Cs.$	$CPU(s)$
30000	500	45000	33594	620	32714	11.8	976.745	33594	620	32714	11.8	689.641
30000	600	45000	30830	665	31737	2.84	8911.263	30830	665	31737	2.84	3345.417
30000	700	45000	29130	675	33176	0.90	16529.984	28801	702	34155	0.92	11624.843
60000	500	20000	58299	419	18112	0.90	9787.185	58452	496	15412	0.93	1214.739
60000	500	40000	58299	419	18112	0.76	3293.860	53318	497	20661	0.82	8821.625
60000	950	25000	58299	419	18112	0.68	1852.365	46329	793	26544	5.55	3045.082
70000	600	20000	70028	383	11033	0.72	13177.488	65348	598	15454	0.91	7308.038
80000	500	25000	80640	311	8225	0.66	7840.264	65963	499	13144	0.80	4455.059
90000	550	20000	89330	241	6091	0.56	6049.002	61362	546	14720	0.83	965.369
100000	400	10000	98287	229	3928	0.64	8354.604	85646	397	8295	0.91	1347.516
100000	600	10000	98287	229	3928	0.56	16511.467	80852	588	10908	1.21	2785.782

Table 2: Results obtained for the penalty based cost function and the mixed cost function (equation (10)).

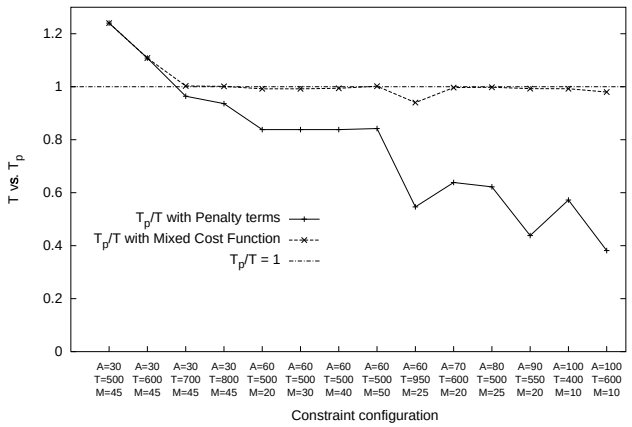

Figure 4: Latency attribute (T_p) vs. latency constraint T for the penalty based cost function and the mixed cost function.

6. Conclusions

In this paper, we have proposed the use of powerful cost functions to consider system constraints in the hardware-software partitioning process. The complete fulfillment of specification constraints is an essential requirement for many designs, being therefore their consideration a key issue during the synthesis process, and in particular in the partitioning phase. The main advantages of our approach are:

- The generality and efficiency of the proposed formulation, since it can be applied to different algorithms and does not depend on the design under optimization.
- The system constraints are considered directly in the cost function, being unnecessary the modification of the optimization algorithms.
- The combination of square, penalty and barrier terms can be used to adjust the cost function to the specific co-design problem.
- Additional system attributes can be easily introduced.

Future work covers the integration of other quality attributes in the cost function, as power consumption or communication overhead. The application of the proposed techniques in other system optimization procedures, as it is scheduling, is also under study.

Acknowledgments

The authors wish to thank V. Srinivasan from the University of Cincinnati for providing us the set of examples used for the experiments.

References

[1] R. K. Gupta and G. De Micheli. HW-SW Cosynthesis for Digital Systems. *IEEE Design & Test of Computers*, pages 29–41, 1993.

[2] R. Ernst, J. Henkel, and T. Benner. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design & Test of Computers*, pages 64–75, Dec 1993.

[3] A. Kalavade and E. A. Lee. The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling and Implementation-bin Selection. *Journal of Design Automation of Embedded Systems*, 2(2):125–164, March 1997.

[4] F. Vahid. Modifying Min-Cut for Hardware and Software Functional Partitioning. In *Proc. Workshop on HW/SW Co-Design CODES/CASHE'97*, Mar 1997. Braunschweig, Germany.

[5] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System Level Hardware/Software Partitioning based on Simulated Annealing and Tabu Search. *Design Automation for Embedded Systems*, 2(1):5–32, January 1997.

[6] M. L. López Vallejo, C. Carreras, J. C. López, and L. Sánchez. Coarse Grain Partitioning for Hardware-Software Co-design. In *Proc. 22nd Euromicro Conference*, pages 161–167, Sep 1996.

[7] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware Software Partitioning with Integrated Hardware Design Space Exploration. In *Proc. DATE'98*, pages 28–35, Paris, France, 1998.

[8] D. G. Luenberger. *Linnear and non-Linear Programming*. Addison-Wesley, 1984.

[9] M.L. López Vallejo and J.C. López. Extending classical hardware partitioning algorithms to solve system level partitioning. In *Proc. DCIS'99*, pages 321–326, Nov 1999.