Frequenz
60 (2006)
9-10

Dynamic Resource Allocation in Software-Defined Radio – The Interrelation between Platform Architecture and Application Mapping*

By V. Marojevic, X. Revés, and A. Gelonch

Abstract *– One of the main features of software-defined radio (SDR) is the dynamic switch from one radio access technology to another. This requires a partial or total reconfiguration of an SDR platform, which contains a cluster of heterogeneous processors with limited processing and dataflow capacities. SDR applications, modeled as dynamic acyclic graphs, require a certain amount of computing resources for real-time execution. The allocation of these resources or, in other words, the mapping of an SDR application to an SDR platform is the subject of this paper. In particular, given an SDR application and the remaining computing resources of an SDR platform, the objective is to find a mapping that meets all system constraints. In this sense, a number of simulations with two mapping algorithms and a parametric cost function, which guides the mapping, show the interrelation between platform architecture and application mapping.*

Index Terms *– Applications, mapping, dynamic computing resource allocation, software-defined radio (SDR)*

1. Introduction

Software radio, or software-defined radio (SDR), is an emerging concept that characterizes the implementation of signal processing chains in software rather than in dedicated hardware [1]–[3]. Therefore, reconfigurable devices, including digital signal processors (DSP's) and field-programmable gate arrays (FPGA's), will be the main processing entities of future *SDR platforms*. An SDR platform stands for an SDR mobile terminal or base stations.

We introduce the term *SDR application* as the part of the signal processing chain of a radio transceiver that is implemented in software. An *SDR application* comprises several *SDR functions* that process and propagate data. An SDR function represents a software-defined signal processing block, such as a filter, a decoder, or a RAKE receiver.

Future SDR applications will be no longer specifically tailored, but rather will have similarities with today's massive computing applications. Therefore we argue that general-purpose computing methods, mapping and scheduling, in particular, should be considered in SDR contexts.

Mapping describes the assignment of software modules to hardware resources. Scheduling determines the execution intervals of mapped software modules. The related contributions propose algorithms that jointly address the mapping and scheduling problems to minimize the overall execution time of an application [4]–[11]. In software-defined radio, however, the principal objective is to meet all system constraints in hard-to-meet cases [12].

This paper addresses the *SDR mapping problem*: An SDR application, which requires a certain amount of computing resources for real-time processing, has to be mapped to an SDR platform with limited computing capacity. Appropriate software and hardware abstractions provide the necessary information on the required and the available computing resources. We consider time as an implicit resource and assume that the SDR application may be executed in a pipelined fashion; this, by and large, solves the scheduling problem. In other words, a mapping that meets all system constraints indicates that the SDR application can be processed within its maximum allowed time frame. The grey shaded blocks in Fig. 1 show the scope of the paper within the context of the SDR mapping problem.

We model SDR applications as directed acyclic graphs (DAG's) of particular characteristics. We randomly generate a number of such DAG's and map them individually to several platform architectures. These platform architectures represent different resource occupations of a dynamically reconfigurable SDR platform. We use two simple mapping algorithms that are guided by a parametric cost function. The contribution of this paper is to show the influence of the platform architecture, the cost function parameter, and the mapping algorithm on the application mapping.

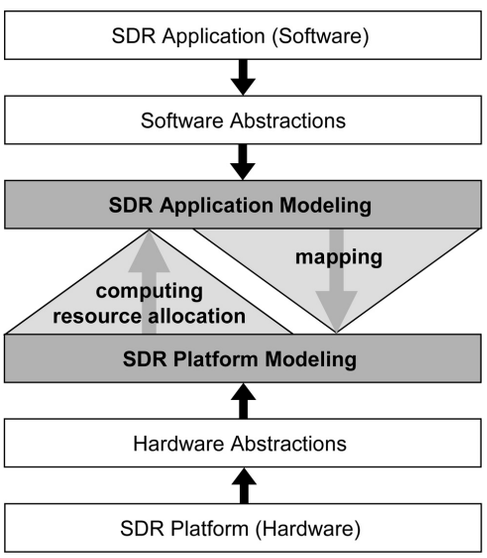

Fig. 1: The scope of the paper within the SDR mapping problem.

The rest of the paper is organized as follows: In section 2 we present the SDR system modeling. Section 3 briefly describes the mapping algorithms and introduces the cost function. In section 4 we discuss the simulation set-up, analyze the results, and derive conclusions.

2. SDR System Modeling

The SDR system modeling encompasses the modeling of SDR platforms and applications. Our modeling accounts for the timing constraints of SDR applications and the limited computing resources of SDR platforms.

2.1 Modeling of SDR Platforms

The computing resources in of an SDR platform are the processing powers of N heterogeneous processors and the available bandwidths between them. N may be as small as 1–3 for a single-user

* This work has been supported by CYCIT (Spanish National Science Council) under grant TIC2003-08609, which is partially financed from the European Community through the FEDER program, the DURSI of the Catalonian Government, and European social funds.

mobile terminal and as large as 10–20 for a multi-user base station. A processor represents an SDR-specific processing device, such as a DSP or an FPGA. The processing powers in MOPS (Million Operations Per Second) of processors P_1 to P_N are resumed in

$$\mathbf{C} = (C_1, C_2, ..., C_N) \text{ [MOPS]}. \quad (1)$$

The available bandwidth in MBPS (Mega-Bits Per Second) between P_i and P_j is B_{ij} ($i,j \in 1, 2, \ldots, N$). We assume a shared memory (of unlimited capacity) for processor-internal communication. Matrix **B** can then be written as

$$\mathbf{B} = \begin{pmatrix} \infty & B_{12} & \cdots & B_{1N} \\ B_{21} & \infty & \cdots & B_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & B_{N2} & \cdots & \infty \end{pmatrix} \text{ [MBPS]}. \quad (2)$$

Without loss of generality, we label processors in order of descending processing capacities; that is, $C_1 \geq C_2 \geq ... \geq C_N$. Furthermore, if $C_i = C_j$ and $i < j$, then

$$\sum_{x=1,x\neq i}^{N} (B_{ix} + B_{xi}) \geq \sum_{x=1,x\neq j}^{N} (B_{jx} + B_{xj}) \quad (3)$$

2.2 Modeling of SDR Applications

We model an SDR application as a cluster of M SDR functions f_1, f_2, …, f_M, where M may be in the order of tens. Any SDR function f_i ($i \in 1, 2, ..., M$) belongs to a chain of at least two SDR functions. Directed acyclic graphs (DAG's) model these SDR function chains, where a node in a DAG stands for an SDR function and an arc for a non-zero bandwidth requirement. SDR functions are logically numbered: if f_i sends data to f_j, than $i < j$ [13].

The modeling of an SDR application features

$$\mathbf{c} = (c_1, c_2, ..., c_M) \text{ [MOPS]}, \quad (4)$$

which absorbs the computing demands, and

$$\mathbf{b} = \begin{pmatrix} 0 & b_{12} & \cdots & b_{1M} \\ 0 & 0 & \cdots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \text{ [MBPS]} \quad (5)$$

which specifies the bandwidth requirements.

3. Mapping

M SDR functions can be mapped to N processors in N^M different ways. The problem of finding an optimal solution is NP hard in general [5]. Therefore, the applied mapping algorithm must be efficient in terms of computing time and mapping results. Here we consider the ordered version of the *t*-mapping [12] and the corresponding greedy approach. The two dynamic programming approaches, which are apt for any cost function, are briefly described in 3.1. A cost function proposal follows thereafter.

3.1 Mapping algorithms

The *t*-mapping systematically maps one process at a time, starting with f_1 and finishing with f_M, to each one of the N processors. The decisions are taken as a function of the accumulated mapping cost due to some cost function. These decisions discard mapping combinations to such a degree that before addressing SDR function f_i ($i \in 2, 3, \ldots, M$), N (partly) different mapping combinations of the first (i–1) SDR functions are available. The algorithm then adds the mapping of f_i to processor P_k ($k \in 1, 2, \ldots, N$) to one of the N mapping combinations of size (i–1).

After finishing the processing of SDR function f_M, the algorithm chooses the mapping combination of minimum cost. This combination represents the mapping proposal due to the particular problem and cost function. The *t*-mapping's computing complexity is of order $M \cdot N^2$.

The greedy or *g*-mapping is a simplification of the *t*-mapping. It maps one process at a time to the processor that is associated with the minimum accumulated cost due to some cost function. That is, the algorithm maps f_i ($i \in 1, 2, \ldots, M$) to either P_1, P_2, …, or P_N and adds it to the mapping combination of size (i–1). Its complexity order is $M \cdot N$.

3.2 Cost Function

The purpose of the cost function is to guide the mapping process so that the mapping proposal meets all system constraints. Hence, the cost function has to manage the limited computing resources of an SDR platform. In [12] we introduced a cost function that seems suitable for this purpose. Parameter q extends this cost function to

$$\text{cost}(k, i) = q \cdot \text{cost}_{\text{comp}}(k, i) + (1-q) \cdot \text{cost}_{\text{comm}}(k, i). \quad (6)$$

The term $cost(k,i)$ represents the cost of mapping f_i to P_k ($i \in 1, 2, \ldots, M$; $k \in 1, 2, .., N$) and is for $i > 1$ a function of the corresponding previous mapping decisions. The computation cost $cost_{comp}(k,i)$ is obtained as the quotient between the required processing power c_i of SDR function f_i and the remaining processing capacity of processor P_k. The sum of up to (i–1) quotients between the required bandwidths (for the data transfers between f_1 and f_i, f_2 and f_i, …, and f_{i-1} and f_i) and the corresponding currently available bandwidths defines the communication cost $cost_{comm}(k,i)$.

Throughout the mapping process the algorithm dynamically updates the remaining processing and bandwidth capacities. This way the algorithm recognizes and discards any *infeasible allocation*, an allocation that reserves more than 100% of any computing resource.

The weight q in (6) may take any real value in [0...1] and specifies the relative importance of the computation cost in respect to the communication cost.

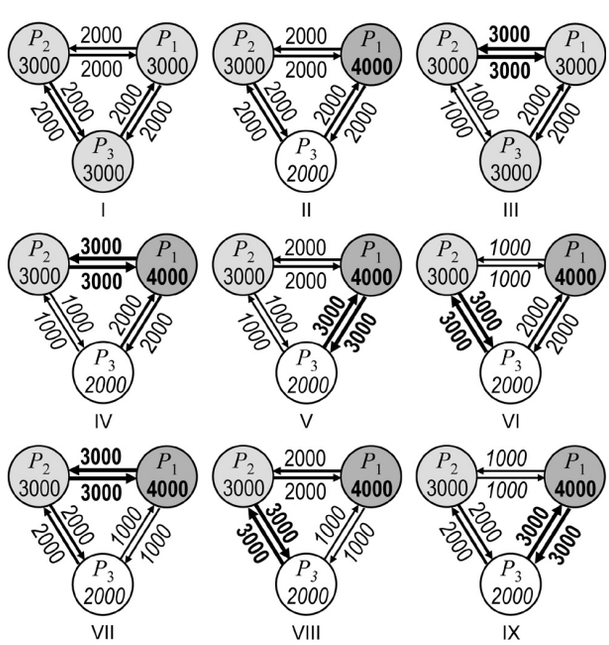

Fig. 2: SDR platform architectures I – IX.

4. Simulations

4.1 SDR Platforms

A future SDR platform will be subject to the dynamic reconfiguration of its functionality [1]–[3]. In other words, the available resources of an SDR platform will be partially or totally de- and reallocated in a dynamic fashion.

We model a partially and dynamically reconfigurable SDR platform as a cluster of three fully interconnected processors. This cluster is representative for an SDR mobile terminal or the minimum computing cell within an array of processors, which models an SDR base station. The partial deallocation of resources immediately before their reallocation leaves the processing platform in one of the nine states shown in Fig. 2. An *SDR platform state*, or *architecture*, abstracts the available computing resources of an SDR platform at some time instance.

Homogeneous processing and bandwidth capacities characterize platform architecture I, heterogeneous processing capacities platform architecture II, heterogeneous bandwidth capacities platform architecture III, and heterogeneous processing and bandwidth capacities platform architectures IV–IX (Fig. 2). Any platform state s ($s \in$ I, II, …, IX) offers a total processing capacity of 9000 MOPS and a total inter-processor bandwidth of 12 000 MBPS.

4.2 SDR Applications

In order to avoid a particular implementation and to provide statistically representative results, we generate 10 million random DAG's with the following parameters:

- $M = 25$,
- $con = 0.2$,
- c_i uniformly distributed in [1, 2, ..., 500] MOPS,
- b_{ij} uniformly distributed in [1, 2, ..., 500] MBPS.

Parameter *con* indicates the probability of drawing an arc between f_i and f_j ($i < j$); no arc between f_i and f_j means $b_{ij} = 0$. Any of the random DAG's consists of one or several components. (A component is a connected subgraph [13]). Several components stand for parallel function chains. A two-component DAG, for example, perfectly models an SDR transceiver with one function chain for the transmit and one for the receive path.

A random DAG requires 6262.5 MOPS in the mean, which is 70% of a platform's remaining computing capacity. The probabilities that the compound processing requirement of an SDR application be larger than 4500 MOPS and 9000 MOPS are 0.99 and $5 \cdot 10^{-5}$, respectively.

The total bandwidth demand of a random DAG is $[con \cdot (M^2 - M)/2] \cdot (500 \text{ MBPS} + 1 \text{ MBPS})/2 = 15\,030$ MBPS in the mean. 93.6% of the DAG's require more bandwidth than the 12 000 MBPS that are available for inter-processor data flow. This can be solved by mapping (highly) communicating SDR functions to the same processor.

4.3 Results and Discussion

For each $q \in$ 0, 0.05, 0.1, …, 1, the two mapping algorithms individually compute the mapping of a DAG to any SDR platform architecture. Figs. 3 and 4 show the percentage of unfeasibly mapped DAG's as a function of the platform architecture and the cost function parameter.

First of all we notice the interrelation between the number of infeasible allocations and the platform architecture: The minimum number of infeasible allocations is achieved for platform states I and IV, the maximum number of infeasible allocations for VI and VIII. We explain this using the notations $P_k^{(s)}$ and $B_{ij}^{(s)}$, where s identifies the platform state.

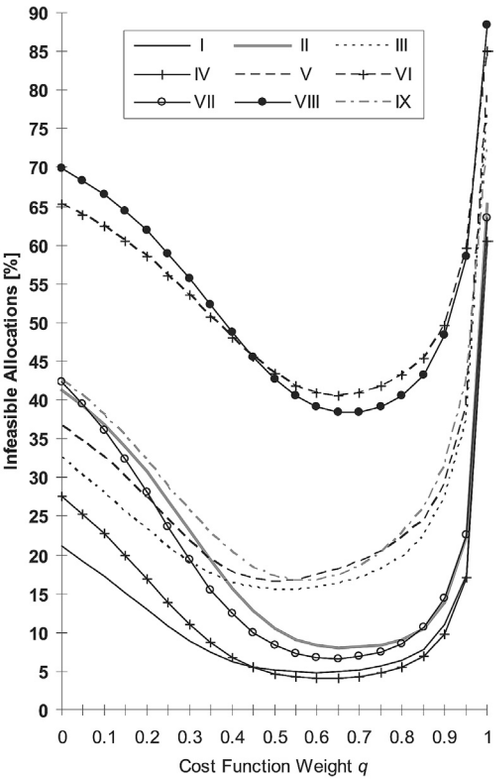

Fig. 3: *g*-mapping results for platform architectures I – IX.

Platform state I works well because the components of the SDR application can be well distributed between the homogeneous processing and link capacities. State III lacks the homogeneous communication network and complicates such a distribution.

Most of the processing load is likely to be distributed between $P_1^{(s)}$ and $P_2^{(s)}$ ($s \in$ II, IV, V, …, IX). States IV and VII are favorable, because $B_{12}^{(IV)} = B_{21}^{(IV)} = B_{12}^{(VII)} = B_{21}^{(VII)} = 3000$ MBPS,

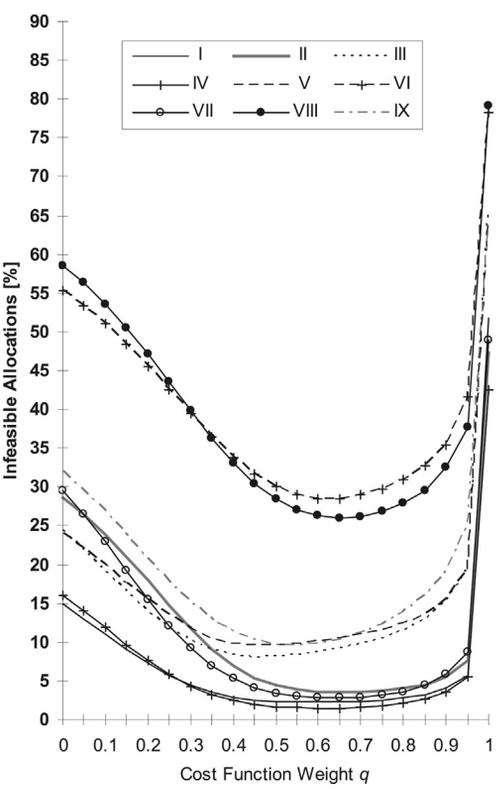

Fig. 4: *t*-mapping results for platform architectures I – IX.

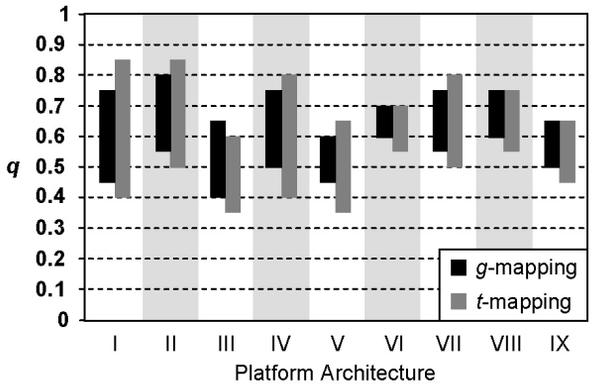

Fig. 5: The *q-range* as a function of the platform architecture and the mapping algorithm.

whereas the corresponding bandwidths of II, V, and VIII (VI and IX) are merely 2000 (1000) MBPS.

$P_1^{(VI)}$ and $P_1^{(VIII)}$ have less communication capabilities than P_1 of any other platform state. The fact that P_1 generally executes more SDR functions than P_2 or P_3 explains the overall inferiority of platform states VI and VIII. In practice we should, therefore, try to avoid these two platform states.

Figs. 3 and 4 also show that any architecture has an optimal q: $q_{opt}^{(I)} = 0.6$, $q_{opt}^{(II)} = 0.7$, $q_{opt}^{(III)} = 0.45$, and $q_{opt}^{(IV–IX)} = 0.45–0.65$. Recall that the higher the q, the more decisive is the computation cost in respect to the communication cost and vice versa. Platform state II, which differs from I in the computing capacities, leads to more infeasible allocations than platform state I; therefore, $q_{opt}^{(II)} > q_{opt}^{(I)}$. Similarly, platform state III, which differs from I in the bandwidth capacities, is inferior to platform state I; therefore, $q_{opt}^{(III)} < q_{opt}^{(I)}$. Platforms states IV–IX are a combination of II and III, and so are their optimal q values.

The two local maxima of the 18 curves confirm that the limited processing and bandwidth capacities require a composite load balancing, that is, $0 < q < 1$. In section 4.2 we have mentioned that the inter-processor bandwidths are the major bottleneck in this study. The cost function with $q = 1$, which balances the processing load and does not care about (excessive) data flow between processors, explains the global maximum at $q = 1$ (Figs. 3 and 4).

As regards the mapping algorithm, the results show that the *g*-mapping is always inferior to the *t*-mapping. The relative inferiority is a function of q and s. In respect to $q_{opt}^{(s)}$, the *g*-mapping leads to about 50% more infeasible allocations for s = VI and VIII, almost twice as many infeasible allocations for s = III, V, and IX, and more than twice as many infeasible allocations for s = I, II, IV, and VII.

If, on the other hand, we require a feasible allocation for at least 90% of the DAG's, both algorithms are suitable for $q = 0.6$ and s = I, II, IV, and VII, only the *t*-mapping for $q = 0.5$ and s = III, V, and IX, and neither the *g*-mapping nor the *t*-mapping for s = VI and VIII.

Finally, we study the robustness of the mapping algorithms against variations of q. Therefore we compute the range of q instances (*q-range*) with less than 100 000 additionally infeasible mappings in respect to the optimal result. That is, if the optimal result for platform state s is $x^{(s)}$ [%], than all instances of q with less than $(x^{(s)} + 1)$ [%] infeasible allocations define the corresponding *q-range*. Fig. 5 shows the *q-range* as a function of the platform architecture and the mapping algorithm.

First we observe that the *q-range* of the *t*-mapping is mostly higher than the *q-range* of the *g*-mapping. Thus, the *t*-mapping is more roust than the *g*-mapping. Fig. 5 further shows that the *q-range* is a function of the platform architecture. In this scenario however, $q = 0.6$ works for any of the nine platform architectures with any of the two mapping algorithms. Hence, the importance of adjusting q to its optimal value is not that critical here; the mapping algorithm and, moreover, the platform architecture condition the application mapping (Figs. 3 and 4). Nevertheless, q_{opt} could be of great importance in another scenario.

References

[1] J. Mitola, "The software radio architecture," *IEEE Commun. Mag.*, vol. 33, no. 5, pp. 26–38, May 1995.

[2] W. H. W. Tuttlebee, "Software-defined radio: facets of a developing technology," IEEE Pers. Commun., vol. 6, iss. 2, pp. 38–44, April 1999.

[3] E. Buracchini, "The software radio concept," *IEEE Commun. Mag.*, vol. 38, iss. 9, pp. 138–143, Sept. 2000.

[4] S.-Y. Lee, J. K. Aggarwal, "A mapping strategy for parallel processing," *IEEE Trans. Comput.*, vol. C-36, no. 4, pp. 433–442, April 1987.

[5] V. Chaudhary, J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms," *IEEE Trans. on Parallel Distrib. Syst.*, vol. 4, iss. 3, pp. 328–346, March 1993.

[6] M. Tan, H. J. Siegel, J. K. Antonio, Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, iss. 8, pp. 857–871, Aug. 1997.

[7] A. H. Alhusaini, V. K. Prasanna, C. S. Raghavendra, "A framework for mapping with resource co-allocation in heterogeneous computing systems," *Proc. 9th IEEE Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, May 2000, pp. 273–286.

[8] H. Topcuoglu, S. Hariri, Min-You Wou, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. on Parallel and Dist. Syst.*, vol. 13, iss. 3, pp. 260–274, March 2002.

[9] R. Bajaj, D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Trans. Parallel Distr. Syst.*, vol. 15, no. 2, Feb. 2004.

[10] A.-R. Rhiemeier, F. Jondral, "Mathematical modeling of the software radio design problem," *IEICE Trans. Commun.*, vol. E86-B, no. 12, Dec. 2003.

[11] A.-R. Rhiemeier, "A comparison of scheduling approaches in modular software defined radio," *Proc. 3rd Karlsruhe Workshop on Software Radios (WSR'04)*, Karslruhe, Germany, March 17/18, 2004.

[12] V. Marojevic, X. Revés, A. Gelonch, "Computing resource management for SDR platforms," *Proc. 16th IEEE Int'l Symp. Personal, Indoor and Mobile Radio Communications (PIMRC 2005)*, Berlin, Sept. 11–14, 2005.

[13] D. F. Robinson, L. R. Foulds, *Digraphs: Theory and Techniques*. Gordon and Breach Science Publisher Inc., 1980.

Vuk Marojevic
Dept. of Signal Theory and Communications
Universidad Politècnica de Catalunya
08034 Barcelona
Spain
Fax: +34 934017200
E-mail: marojevic@tsc.upc.edu

Xavier Revés
Dept. of Signal Theory and Communications
Universidad Politècnica de Catalunya
08034 Barcelona
Spain
Fax: +34 934017200
E-mail: xavier.reves@tsc.upc.edu

Antoni Gelonch
Dept. of Signal Theory and Communications
Universidad Politècnica de Catalunya
08034 Barcelona
Spain
Fax: +34 934017200
E-mail: antoni@tsc.upc.edu

(Received on July 7, 2006)
(Revised on July 15, 2006)