Dynamic Resource Allocation in Software-defined Radio – The Interrelation Between Platform Architecture and Application Mapping

V. Marojevic, X. Revés, A. Gelonch

Polythechnic University of Catalonia
Dept. of Signal Theory & Communications, C/ Jordi Girona 1-3, 08034 Barcelona, Spain
{marojevic|xavier.reves|antoni}@tsc.upc.edu

***Abstract*—This paper addresses the assignment of computing resources to software-defined radio (SDR) applications. We model SDR platforms as some interconnected processors with limited processing and data flow capacities. SDR applications, modeled as dynamic acyclic graphs, require a certain amount of these computing resources for real-time execution. The problem can be formulated as finding a mapping (of an SDR application to an SDR platform) that meets all system constraints. Several 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, resource allocation, software architectures, software defined radio (SDR).**

I. Introduction

Software radio, or software-defined radio (SDR), is an emerging technology that is characterized by the implementation of signal processing chains in software rather than in dedicated hardware [1]–[3]. As a result, programmable devices such as digital signal processors (DSP's) and reconfigurable logic devices, e.g. field-programmable gate arrays (FPGA's), may be mixed for building SDR mobile terminals and 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*. An SDR function represents a software-defined signal processing block, such as a filter, a decoder, or a RAKE receiver. Hence, 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, such as mapping and scheduling, should be considered in SDR contexts.

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

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. Further discussion is beyond the scope of this paper.

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 different platform architectures by means of the *t*-mapping [12] and a simple reference algorithm. The mapping process is guided by a parametric cost function. The platform architectures represent different resource occupations of a partially and dynamically reconfigurable SDR platform.

The contribution of this paper is twofold: We

1. analyze the influence of the platform architecture and the cost function parameter on the mapping results and

2. discuss the benefits of using a more sophisticated mapping algorithm as a function of the platform architecture.

The rest of the paper is organized as follows: Section II presents the SDR system modeling. We then briefly describe the two mapping algorithms that are considered in this work. We introduce the cost function in section IV and, finally, discuss the simulation set-up and the simulation results in section V.

II. SDR System Modeling

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

A. Modeling of SDR Platforms

In the first approximation of a SDR platform, we consider the processing powers of N heterogeneous processors and the available bandwidths between them as the main system resources. N may be as small as 1–3 for a single-user 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, P_2, …, and P_N are resumed in

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

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, thus, be written as

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

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

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

B. Modeling of SDR Applications

We model an SDR application as a cluster of M SDR functions $f_1, f_2, \ldots, f_M$, 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 applications features

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

which absorbs the computing demands, and

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

which specifies the bandwidth requirements.

III. Mapping Algorithms

M SDR functions can be mapped to N processors in N^M different ways and the problem of finding an optimal solution is NP hard in general [5]. Therefore, the applied 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 complexity orders of these two algorithms are $M \cdot N^2$ and $M \cdot N$. A brief description follows.

A. The t-mapping

The *t*-mapping is a dynamic programming approach that is apt for any cost function [12]. It 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 mappings 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 paths* of length (i–1).

After finishing the processing of SDR function f_M, the algorithm chooses the mapping path of length M of minimum cost. This mapping path represents the mapping proposal due to the particular problem and cost function.

B. The g-mapping

The *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 path of length (i–1).

IV. Cost Function

The purpose of the cost function is to guide the mapping process so that the final mapping proposal meets all system constraints. This implies the management of the limited computing resources of SDR platforms. In [12] we introduced a cost function that seems suitable for this purpose. We extend this cost function to be a weighted superposition of the computation and the communication costs:

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

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 available resource.

The weight q in (6) may take any real value in [0...1] ($q \in 0, 0.1, 0.2, \ldots, 1$ in this study) and specifies the relative importance of the computation cost in respect to the communication cost.

V. Simulations

A. 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 a future 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 constitutes an SDR base station. The partial deallocation of resources at some point in time immediately before their reallocation leaves the processing platform in one of the nine states shown in Fig. 1.

Homogeneous processing and bandwidth capacities characterize platform architecture I (Fig. 1a), heterogeneous processing capacities platform architecture II (Fig. 1b), heterogeneous bandwidth capacities platform architecture III (Fig. 1c), and heterogeneous processing and bandwidth capacities platform architectures IV–IX (Figs. 1d–i). Any platform architecture, or state, s ($s \in$ I, II, …, IX) offers a total processing capacity of 9000 MOPS and a total inter-processor bandwidth of 12000 MBPS.

B. SDR Applications

We generate 10^6 random DAG's, which represent different SDR applications, in order to avoid a particular implementation and to provide statistically representative results. The generation parameters are

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

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 the platform's remaining computing capacity. Furthermore, the probabilities that the compound processing requirement of an

SDR application be larger than 4500 and 9000 MOPS are 0.99 and $5 \cdot 10^{-5}$.

C. Simulation Results and Discussion

The two mapping algorithms individually map the 10^6 random DAG's to the nine platform architectures and return the number of *infeasible allocations* as a function of the cost function parameter q. Fig. 2 shows the results.

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, IV, and VII, the maximum number of infeasible allocations for VI and VIII. In other words, a completely homogeneous platform state and two implementations of a completely heterogeneous one are favorable over the six other states. We explain this in continuation and, therefore, introduce the notation $P_k^{(s)}$, which stands for processor P_k of platform state s.

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 25 SDR functions are mapped to P_1 and P_2 of platforms II, IV–IX. Among these, IV and VII are favorable, because the bidirectional bandwidth between P_1 and P_2 is 2·3000 MBPS, whereas the corresponding bandwidth of II, V, and VIII (VI and IX) is only 2·2000 (2·1000) MBPS.

$P_1^{(VI)}$ and $P_1^{(VIII)}$ are worst communicated with respect to P_1 of any other platform state. If we assume that P_1, generally, executes more SDR functions than P_2 or P_3, we comprehend the overall inferiority of these two architectures. In practice we should, therefore, try to avoid platform states VI and VIII.

Fig. 2 also shows that any architecture has an optimal q: $q_{opt}^{(I)} = 0.6$, $q_{opt}^{(II)} = 0.7$, $q_{opt}^{(III)} = 0.5$, and $q_{opt}^{(IV–IX)} = 0.5–0.7$. 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 values for q. Finally, q must not be a boundary value, because any (modeled and real) SDR platform (state) has limited processing and bandwidth capacities.

As regards the mapping algorithm, the results show that the *g*-mapping is always inferior to the *t*-mapping. The absolute 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, more than twice as many infeasible allocations for s = I, II, and VII, and almost three times as many infeasible allocations for s = IV.

If, on the other hand, we require a feasible allocation for more than 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.

ACKNOWLEDGMENT

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.

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.

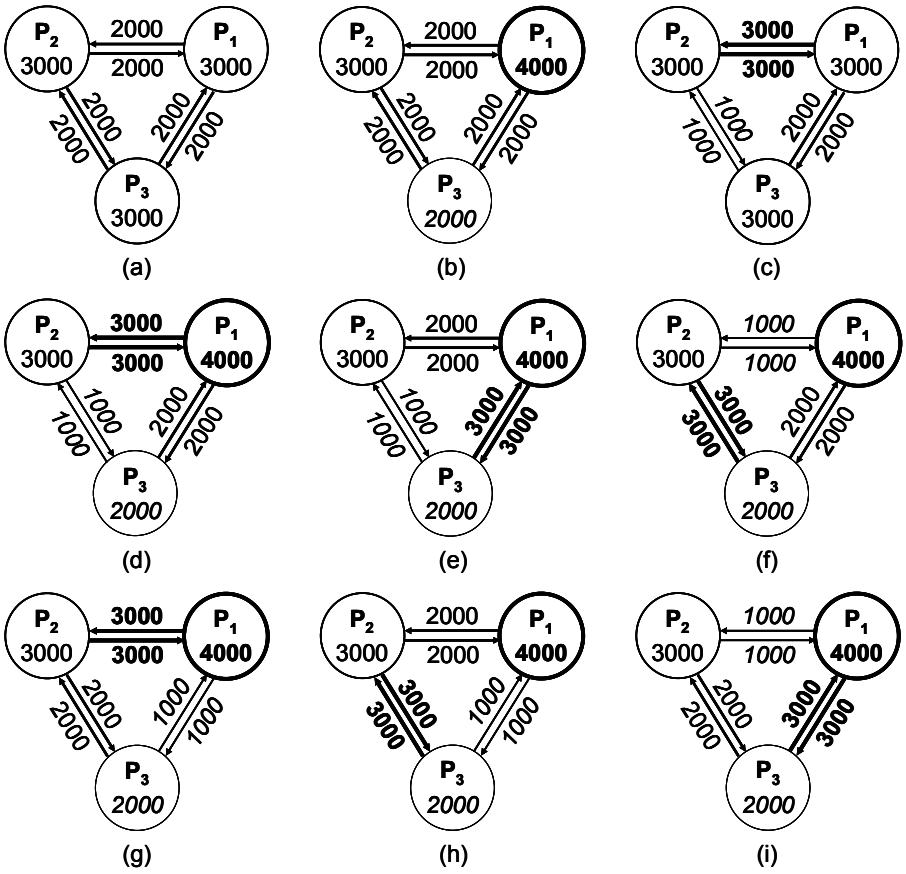

Fig. 1. SDR platform architectures I–IX.

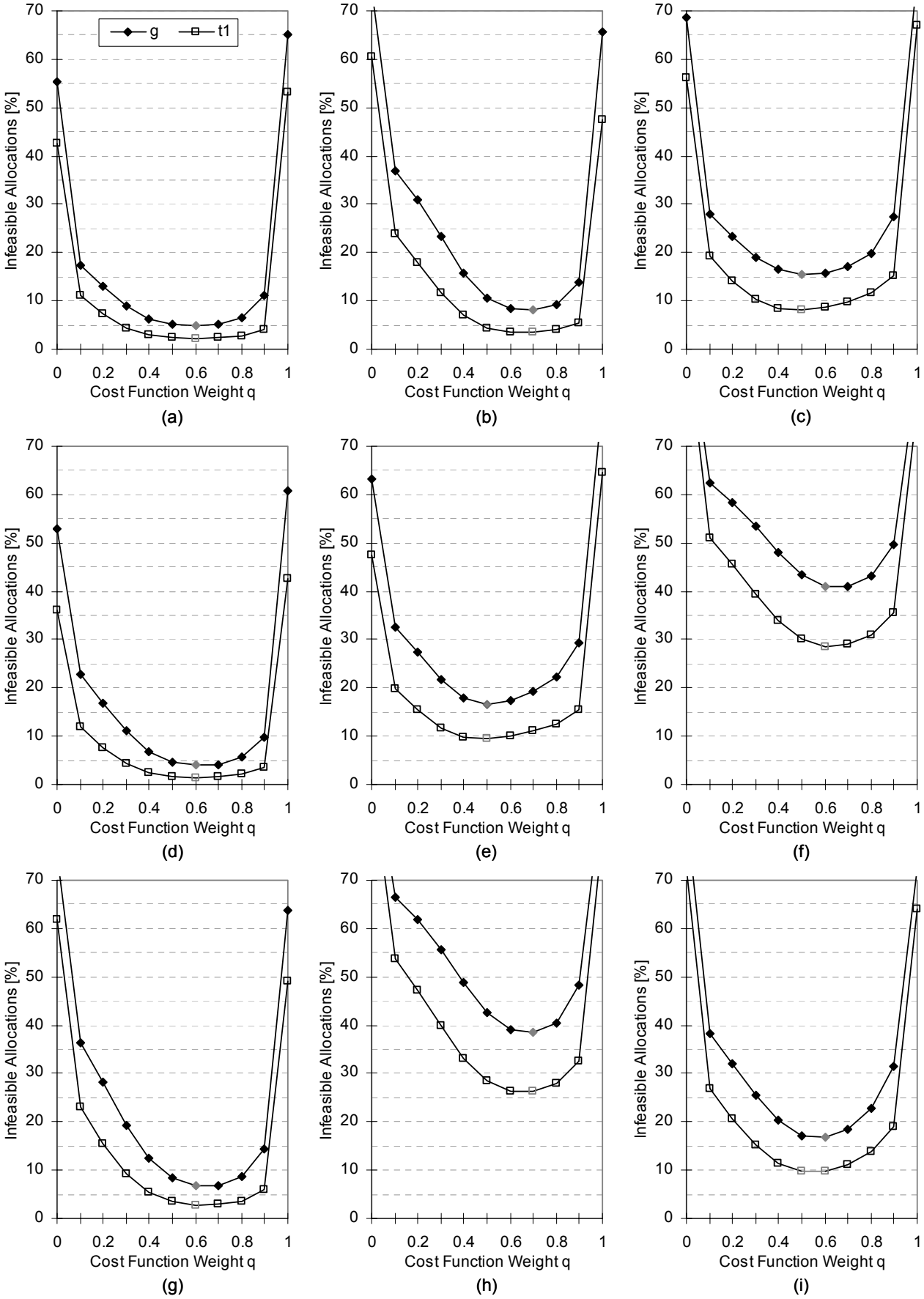

Fig. 2. Simulation results for platform architectures I–IX.