A Computing Resource Management Framework for Software-Defined Radios

Vuk Marojevic, *Student Member*, *IEEE*, Xavier Revés Ballesté, *Member*, *IEEE*, and Antoni Gelonch

Abstract—Software-defined radio (SDR) is an emerging concept that leverages the design of software-defined and hardware-independent signal processing chains for radio communications. It introduces flexibility to wireless systems, facilitating the dynamic switch from one radio access technology to another or, in other words, the de and reallocation of computing resources from one SDR application to another. This paper introduces an SDR computing resource management framework. It accounts for several SDR system characteristics, including real-time computing requirements, limited computing resources, and heterogeneous multiprocessor platforms. The framework features the t_w-mapping, a dynamic mapping algorithm that is apt for many cost functions and radio scenarios. The cost function proposal dynamically manages the available computing resources to satisfy the SDR computing constraints. Two SDR scenarios, based on representative SDR platforms and processing chains, and the corresponding simulation results demonstrate the framework's relevance and suitability for SDRs.

Index Terms—Computing resource management, framework, heterogeneous computing, mapping, scheduling, software-defined radio (SDR), system modeling.

1 Introduction

The *software radio* concept was introduced in the mid-1990s and characterizes those transmitters and receivers (transceivers) that implement the entire signal processing chain in software [1], [2]. Software-defined radio (SDR) can be considered a generalization of software radio because it characterizes a transceiver that implements one or more signal processing blocks in software [3]. SDR introduces flexibility to wireless systems: It permits the adjustment or switching of a terminal's radio access technology (RAT) implementation to adapt to changes in the radio environment of today and tomorrow.

For about a decade, SDR-related research along the whole line between the mobile terminal (MT) transceiver and the core network has been ongoing [4], [5], [6], [7]. It is motivated by the evolution of information technology: The introduction of new RATs, such as the universal mobile telecommunications system (UMTS) or the IEEE 802.11 family of wireless local area networks, the required compatibility with the existing ones, including global system for mobile communications (GSM) and general packet radio service (GPRS), and the increasing demand for new and differentiated user services call for flexible transceiver solutions.

For the above reasons, the flexibility of general-purpose processors (GPPs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs), picoArrays [8], networks-on-chip (NoCs) [9], or multiprocessor systems-on-chip (MP-SoCs) [10] is gaining interest over the energy efficiency of application-specific integrated circuits (ASICs) [11], [12], [13]. State-of-the-art reconfigurable devices, including arrays of processors, offer high computing capacities at moderate power consumptions. This permits the extension of the digital and reconfigurable radio part while reducing the analog and nonreconfigurable circuits.

An SDR processing chain (SDR application or waveform) is the part of an SDR transceiver that is implemented in software. It may be understood as a set of concurrent processes that continuously process and propagate real-time data. Such a processing chain is not specifically tailored, but, rather, executable on any general-purpose platform with sufficient computing capacity. Because of these similarities between future SDR applications and platforms and today's general-purpose computing applications and platforms, we consider general-purpose computing methods practical for SDR systems. We particularly believe that the introduction of appropriate mapping and scheduling techniques, which are essential for the dynamic switch between RATs, will leverage the design of SDR platforms and applications. *Mapping* describes the process of assigning software modules to hardware resources, whereas *scheduling* determines the execution times of these modules. We consider them as two complementary methods for computing resource management.

Wireless or SDR systems, however, reveal specific aspects, essentially regarding flexibility and efficiency, which have not been jointly considered so far in heterogeneous computing. These are given as follows:

1. time slot based division of the transmission medium (*radio time slot*),
2. continuous data transmission and reception,
3. RAT-specific quality-of-service (QoS) targets,

- *V. Marojevic and A. Gelonch are with the Departament de Teoria del Senyal i Comunicacions (TSC), Universitat Politècnica de Catalunya, Av/ Canal Olímpic s/n, 08860 Castelldefels, Barcelona, Spain. E-mail: {marojevic, antoni}@tsc.upc.edu.*
- *X.R. Ballesté is with Gigle Semiconductor S.I., Barcelona, Spain. E-mail: xavier.reves@tsc.upc.edu.*

Manuscript received 6 July 2007; revised 3 Jan. 2008; accepted 13 Mar. 2008; published online 19 May 2008.
Recommended for acceptance by V. Barbosa.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-07-0308.
Digital Object Identifier no. 10.1109/TC.2008.83.

4. real-time computing requirements and limited computing resources,
5. different computing constraints and loads for different RATs or radio conditions,
6. dynamic reconfiguration of the protocol stack, either partial or total, and
7. heterogeneous multiprocessor platforms.

The possibility of dynamically reconfiguring an SDR platform provides full flexibility in adapting an SDR application's computing requirements to the momentary radio environment and QoS demands. This requires extending the available computing resource management approaches toward a framework that links the computing with the radio resources.

This paper introduces an SDR computing resource management framework that facilitates the interrelation between SDR computing resources and wireless systems. It is capable of dynamically allocating the necessary amount of the available computing resources in real time in order to ensure reliable radio communications at adequate QoS levels. The framework trades off performance for flexibility, allowing for different management policies. This way, it can face changes in the radio and computing environments. None of the known related contributions provide the necessary degree of flexibility to properly handle the reconfiguration process of SDR platforms.

The rest of this paper is organized as follows: Section 2 examines this paper's context, its scope, and related work. Section 3 introduces an SDR computing system modeling for the computing resource management of Section 4, which features a novel mapping algorithm and cost function. Section 5 presents two SDR scenarios and simulation results and Section 6 provides the conclusions.

2 Context and Related Work

The SDR computing context principally consists of heterogeneous multiprocessor platforms, either base stations (BSs) or MTs, and software-defined digital signal processing chains. MTs are very limited in computing and energy resources, flexibility, and support for concurrent RAT implementations. On the contrary, the computing resources of BSs are less limited, their power consumption is not a constraint, and the number of concurrent RAT implementations can be as high as desired. Nevertheless, the potentially large number of users and the platforms' high degrees of flexibility, modularity, and reconfigurability make computing resource management at BSs equally important though more complex than that at MTs.

2.1 Problem Formulation

The problem consists of defining a flexible framework that interfaces the SDR computing resources on one side and the wireless system, represented by the SDR computing requirements, on the other. The framework should be able to efficiently and dynamically map precedence-constrained SDR applications to SDR platforms while meeting all SDR computing constraints. These constraints are, primarily, the SDR applications' *real-time computing requirements*, defined by the *minimum bit rate* and *maximum latency demands*, and the SDR platforms' *limited computing resources*.

2.2 Related Work

The literature contains a plentitude of contributions to multiprocessor mapping and scheduling in heterogeneous computing. These works address a wide variety of problems in general and special-purpose computing contexts. Many contributions jointly tackle the mapping and scheduling problems and present optimal or suboptimal solutions following different objectives: [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], for example, aim at minimizing the schedule length or communication overhead, [25], [26], [27], [28], [29], [30] focus on meeting real-time deadlines, whereas [24], [31], [32], [33], [34] pursue additional or other objectives. The following paragraphs depict some of these articles in more detail.

Different scheduling techniques to execute an application as fast as possible are presented in [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. Apart from minimizing the scheduling length, [24] also minimizes the application's failure probability.

Peng et al. [25] and Hou and Shin [26] address the problem of optimally allocating periodic tasks which are subject to task precedence and timing constraints to processing nodes in a distributed real-time system. The efficient local scheduling of tasks in a real-time multiprocessor system is the topic in [27]. If a task's deadline cannot be met on a particular processing node, this task can be sent to another node [28]. The task model, which is identical in both papers, accounts for worst-case computation times, deadlines, and resource requirements; no precedence constraints are assumed.

A list scheduling framework for the runtime stabilization of static and dynamic tasks with hard and soft deadlines, respectively, is described in [29]. Moreira et al. [30] address hard real-time streaming applications and assume a scenario where jobs enter and leave a certain homogeneous multiprocessor system at any time during operation. It combines global resource allocation (admission control) with local resource provisioning (scheduling).

Multirate and cyclic dependencies between tasks are dealt with in [31]. It presents a strategy that binds multiple synchronous data-flow graphs (SDFGs) with throughput constraints to a heterogeneous MP-SoC.

A performance measurement framework for quantifying the success of a resource management system is the topic in [32], whereas [33] tackles the problem of robust resource allocation in dynamic real-time systems. A robust resource allocation would ideally avoid reconfiguration due to runtime parameter variations. Seven different problems and their solutions are finally discussed in [34]. These include optimal job scheduling techniques [34, pp. 99-112] and a design methodology that minimizes contention with minimum communication resources [34, pp. 174-190].

The SDR computing resource management context requires flexibility and efficiency at the same time. It is not an optimization problem with a fixed objective, unique platform, or predefined constraints. The constraints are a function of the (highly varying) radio environment and just have to be met. Speeding up an SDR application is, particularly, not necessary. Moreover, to properly handle the radio link timing requirements, this framework treats processing time as just another limited computing resource. Nevertheless, concepts such as global mapping, followed by local scheduling [30] and robust resource allocation [33], are

practical in SDR. The latter is relevant for partial reconfigurations, which we do not specifically address in this paper.

3 SDR Computing System Modeling

This section introduces a modeling that accounts for several SDR-specific system characteristics. Section 3.1 discusses the computing resource management facilities that constitute the basis for the system models of Section 3.2. Section 3.3 describes the mechanisms that facilitate meeting the SDR computing constraints of Section 2.1. Finally, Section 3.4 exemplifies the modeling. Some definitions follow.

Definition 1. *An* SDR application *is comprised of a chain of RAT-specific SDR functions which characterize the software-defined processing layers of a transmitter or receiver or both.*

Definition 2. *An* SDR function *is a signal processing block, such as a modulator or an equalizer. It is not necessarily implemented as one monolithic piece of binary code but rather as a composition of SDR processes.*

Definition 3. *An* SDR process *is the smallest manageable unit and symbolizes an indivisible binary code.*

Although the rest of this paper addresses SDR functions, the framework may work as well on the basis of SDR processes or a mix of SDR functions and processes.

3.1 Computing Resource Management Facilities

3.1.1 Abstractions and Metrics

An SDR platform represents an MT or a BS. These platforms are comprised of a few or many heterogeneous processing devices, such as FPGAs, DSPs, and GPPs, which communicate with each other. An FPGA's prime resource is the logic area for parallel processing, which can be converted to multiply-accumulate operations (MACs) per time unit when using well-defined benchmarks (filter, FFT, and so forth). DSP, GPP, and NoC performances are typically given in million instructions per second (MIPS).

We assume that a hardware abstraction layer (HAL), middleware, or execution environment provides the necessary hardware abstractions to supply a pseudohomogeneous computing environment on top of a heterogeneous computing platform. The Platform and Hardware Abstraction Layer (PHAL) [35] can provide such an environment. It is also capable of synchronizing the execution on all processors at a suitable time granularity. Note that opting for or against hardware abstractions is a trade-off between programmability, or flexibility, and efficiency. SDR requires a flexible usage of computing resources. Therefore, this framework assumes the availability of hardware abstractions, which is not the topic of this paper.

We consider processing powers and bandwidth capacities to be the principal computing resources in SDR. Hence, the processing powers and the interprocessor bandwidths abstract an SDR platform. Mitola proposed characterizing all platform features, including the processing powers and bandwidths, in equivalent million operations per second (MOPS) [1]. We adopt this unit for characterizing the processing powers. Similarly, we quantify any communication facility in megabits per second (Mbps).

The processing requirements are a function of the processor that finally executes the software module, the module's particular implementation and bit precision, the optimization level (speed versus memory or area), and the performance demand. The interprocessor bandwidth requirement is, basically, a function of the bit precision and the data rate. Despite these dependencies, we use the same metrics to characterize the computing requirements of SDR application. The different implementations need to be available to the execution environment which provides their computing requirements in MOPS and Mbps to the computing resource management framework.

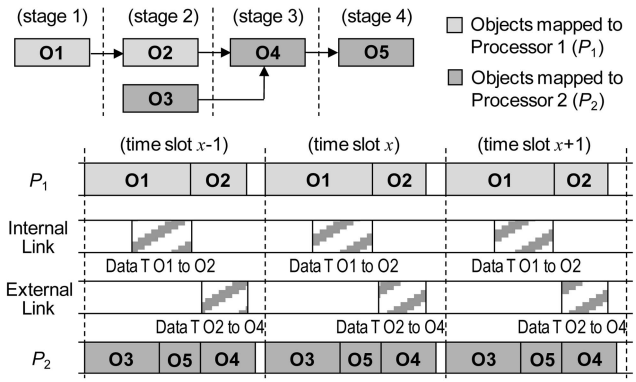


Fig. 1. Illustration of the time slot division and pipelining.

3.1.2 Time Slot Division and Pipelining

Data that is transmitted or received over the wireless link needs to be processed for as long as there is data to transmit or receive. An SDR application will typically execute during the entire user session, even though there might be periods where no user data is transmitted. The fact that an SDR application may be replaced by another during a single user session does not affect this continuous data processing. We thus propose breaking up the continuous execution into periodic executions by dividing the computing resource *time* in equidistant computing time slots and the SDR application in pipelining stages. Fig. 1 illustrates this.

The introduction of the computing time slot, time slot from here on, allows us to identify a processor's computing capacity on time slot basis. This provides the basic mechanism for an efficient computing resource management and is especially useful for satisfying the maximum latency demands of SDR applications (Section 3.3).

The pipelined execution of an SDR application establishes that, in any time slot, all SDR functions process and propagate some part of the data. That is, the same processing and data transfers repeat each time slot on a different data portion (Fig. 1). This introduces synchronization requirements, which PHAL can satisfy [35]. Pipelining also introduces latency, which must be maintained within the radio service and QoS-dependent limits. Section 3.3 explains how to achieve this.

From the above discussion, we derive the new units million operations per time slot (MOPTS) and megabits per time slot (MBPTS) as $t_{ts} \cdot \mathrm{MOPS}$ and $t_{ts} \cdot \mathrm{Mbps}$, where t_{ts} is the time slot duration that is specified in Section 3.3. MOPTS and MBPTS synchronize the available computing resources with the time slot management and are the basic units for the system models that follow.

3.2 System Models

3.2.1 Resource Models

The resource models comprise the *device* and *communication models*.

$$\boldsymbol{C} = (C_1, C_2, \ldots, C_N)\ [\text{MOPTS}] \tag{1}$$

represents the device model and absorbs the processing capacities of processors P_1 to P_N. The total processing capacity C_T of a processing platform is then $C_\text{T} = C_1 + C_2 + \cdots + C_N$. Devices are labeled in order of decreasing processing capacities.

We consider an interconnection network that consists of unidirectional communication lines between the platform's devices. The communication model features matrix $\boldsymbol{B}$, which is defined as

$$\begin{aligned}\boldsymbol{B} &= \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1N} \\ B_{21} & B_{22} & \cdots & B_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & B_{N2} & \cdots & B_{NN} \end{pmatrix} \\ &= \begin{pmatrix} \infty & B_{12} & \cdots & B_{1N} \\ B_{21} & \infty & \cdots & B_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & B_{N2} & \cdots & \infty \end{pmatrix} [\text{MBPTS}].\end{aligned} \tag{2}$$

Element B_{12}, for instance, specifies the available bandwidth per time slot of the directed communication link between P_1 and P_2. We assume direct memory access (DMA) or pointer transfers, where processor internal bandwidths are sufficiently high to be modeled as infinite. The sum of all elements in (2) excluding those on the main diagonal is equal to the platform's total bandwidth capacity B_T. Parameter CON finally captures the relation between the number of nonzero elements in B that are not on the main diagonal and $(N^2 - N)$. It describes the topology, or connectivity, of an SDR platform.

3.2.2 Processing Models

The processing models consist of the *function*, *data flow*, and *stage models*. An SDR application is comprised of the M SDR functions $f_1, f_2, \ldots, f_M$. The function model

$$\boldsymbol{c} = (c_1, c_2, \ldots, c_M)\ [\text{MOPTS}] \tag{3}$$

provides their processing requirements. An SDR application's processing requirement c_T is then the sum of the SDR functions' processing requirements, that is, $c_\text{T} = c_1 + c_2 + \cdots + c_M$.

We model SDR function chains as directed acyclic graphs (DAGs) and apply a logical numbering: If f_j sends data to f_i, then $j < i$ [36].

The data-flow model characterizes the data flow between SDR functions. It features matrix $\boldsymbol{b}$, defined as

$$\begin{aligned}\boldsymbol{b} &= \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M1} & b_{M2} & \cdots & b_{MM} \end{pmatrix} \\ &= \begin{pmatrix} 0 & b_{12} & \cdots & b_{1M} \\ 0 & 0 & \cdots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} [\text{MBPTS}].\end{aligned} \tag{4}$$

Element b_{ji} is the minimum bandwidth that is necessary for sending a certain amount of data from f_j to f_i $(j, i \in 1, 2, \ldots, M)$. As discussed in Section 3.1.1, $\boldsymbol{c}$ and $\boldsymbol{b}$ are a function of the processor type, the optimization level, and so forth. Therefore, $\boldsymbol{c}$ and $\boldsymbol{b}$ for each one of the platform's processors should be available to the framework. Without loss of generality and to improve its readability, this paper treats $\boldsymbol{c}$ and $\boldsymbol{b}$ as if they were the same for all processors of a given platform.

The sum of all elements in (4) defines the SDR application's total bandwidth requirement b_T. Parameter con is obtained as the number of nonzero elements in $\boldsymbol{b}$ divided by $(M^2 - M)/2$. It thus quantifies the connectivity of a processing chain, relating the number of arcs of the corresponding task graph to the maximum possible number of arcs in an M-node DAG.

The direct predecessors (successors) of f_i are all those SDR functions that correspond to nonzero entries in the ith column (row) of $\boldsymbol{b}$. Hence, columns (rows) of all 0s indicate *source* (*sink*) *functions* [36]. This leads to the stage model $\boldsymbol{s}$, given as

$$\boldsymbol{s} = (s_1, s_2, \ldots, s_M). \tag{5}$$

Vector $\boldsymbol{s}$ is obtained by first assigning all source functions to pipelining stage 1. An SDR function f_i is assigned to pipelining stage $s_i = x$ $(x > 1)$ if it receives data from no other SDR function than from those in stages $x - 1, x - 2, \ldots, 1$ and if at least one of its direct predecessors is in stage $x - 1$. The number of pipelining stages n_{ts} is then $n_{ts} = s_M$.

3.3 Meeting the SDR Computing Constraints

The computing resource management facilities of Section 3.1 and the system models of Section 3.2 permit mapping an SDR application to an SDR platform on the basis of a single time slot. A *feasible mapping* reserves no more than 100 percent of any available computing resource.

We assume that coprocessors facilitate the concurrent data processing and data propagation on all processor's inputs and outputs. Many related contributions, such as [15], [16], [17], make this assumption. Since repetitive operations on data samples and continuous outputs, often one per execution cycle, characterize digital signal processing, we may further assume that the software and hardware facilitate the immediate propagation of processed data samples. PHAL finally manages the synchronized execution on all processors and provides pipelining and buffering mechanisms, among others, for the proper and timely data delivery.

The usually complex scheduling process can, on the basis of a feasible mapping and under the above assumptions, be simplified to N independent local scheduling tasks.

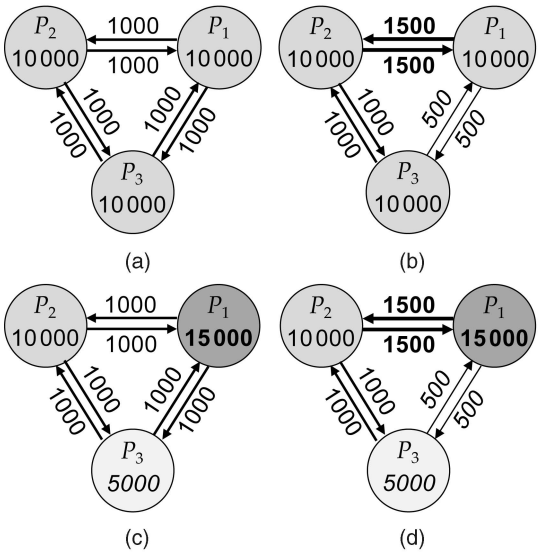


Fig. 2. Processing and bandwidth resources in MOPS and Mbps of SDR platforms I-IV, (a)-(d) respectively.

Particularly, a processor's local scheduler is capable of organizing the execution sequence of the corresponding SDR functions' portions and their data transfers within the given time slot boundaries (*feasible schedule*, see Fig. 1). This ensures that the input data of any SDR application's module or set of modules is processed according to its arrival rate so that no data is accumulated anywhere in the processing chain, meeting the *minimum bit rate requirement*.

The time slot duration t_{ts} times the number of time slots n_{ts} is the pipelining latency in case of a feasible schedule on each processor. We specify t_{ts} as

$$t_{ts} = \frac{L_{\max}}{n_{ts}} \text{ [SPTS]} \tag{6}$$

to meet the maximum allowable latency $L_{\max}$, where SPTS stands for seconds per time slot. This latency is a function of the tolerable end-to-end delay of a radio communication link due to the service and QoS agreements between the radio service provider and the user. We assume that t_{ts} is large enough for the (efficient) execution of any SDR function in the processing chain.

3.4 Modeling Examples

Fig. 2 shows four processing platforms. Three (pseudo) homogeneous processors and a homogeneous communication network characterize SDR platform I (Fig. 2a). Platform II differs from I in that its interprocessor bandwidths are heterogeneous (Fig. 2b). Platforms III and IV are equivalent to platforms I and II except for the heterogeneous processors (Figs. 2c and 2d).

P_1, P_2, and P_3 stand for three processors or three tightly coupled clusters of processors. The picoArray *PC101*, for example, embeds 430 heterogeneous processors with a total processing power of 206,000 MIPS [8].

Fig. 3 depicts the functional diagram of the digital signal processing chain at the physical layer of a software-defined UMTS downlink receiver. It is comprised of 24 SDR functions from the *digital down conversion* to the *cyclic redundancy check* (CRC) [37]. The computing requirements are estimates from [37], [38], [39] and available implementations using the *TMS320C6416* DSP and the *Code Composer Studio* from Texas Instruments [40]. In particular, the number of MACs of an SDR function's implementation times the sampling rate f_s specifies the SDR function's processing requirement. A bandwidth requirement is the product between the f_s and the bit precision of $2 \cdot 16$ bits for the real and the imaginary components.

Figs. 4 and 5 show the corresponding system models. We consider $L_{\max} = 10$ ms (the UMTS radio link uses 10 ms long frames to synchronize the data transmission with its reception) and obtain a time slot duration of $t_{ts} = 0.01/17 = 0.588 \cdot 10^{-3}$ SPTS (6), where 17 is the number of stages n_{ts} due to Fig. 5b.

4 SDR Computing Resource Management

Because different optimization criteria are conceivable in SDR, such as meeting real-time computing constraints in hard to meet conditions or optimizing energy consumption,

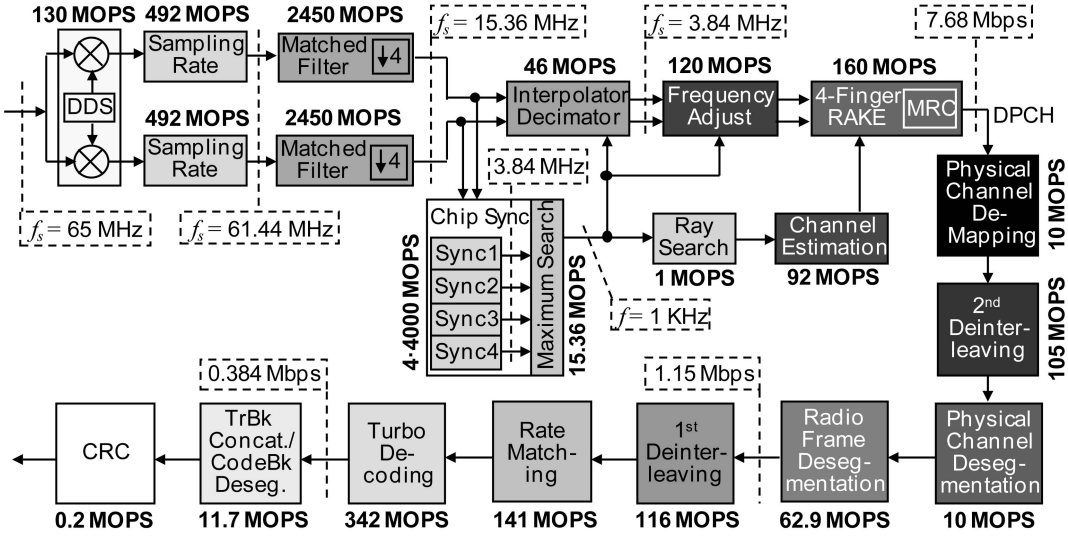
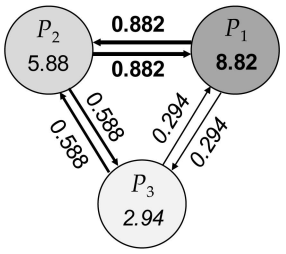


Fig. 3. Functional diagram and computing requirements of a UMTS downlink receiver.



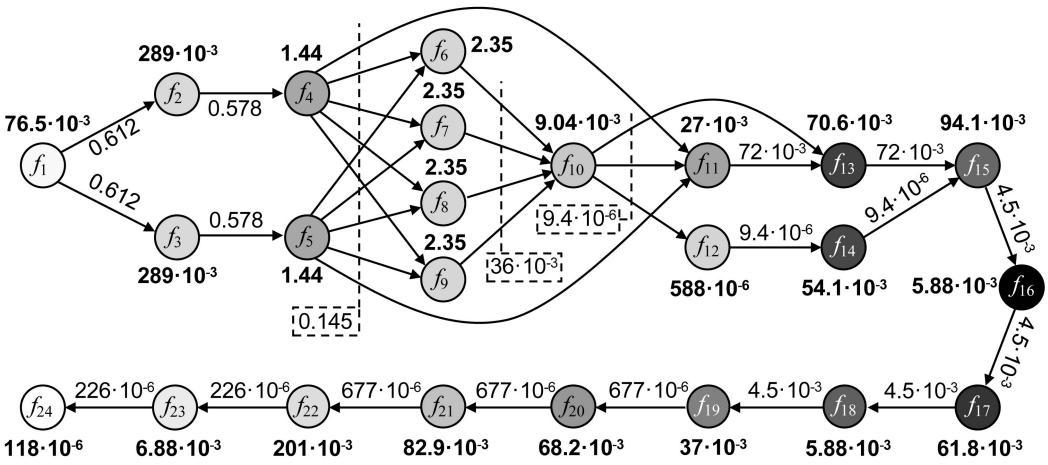
(a)

Device model: $\boldsymbol{C} = (8.82, 5.88, 2.94)$ MOPTS $\quad C_T = 17.65$ MOPTS

Communication model: $\boldsymbol{B} = \begin{pmatrix} \infty & 0.882 & 0.294 \\ 0.882 & \infty & 0.588 \\ 0.294 & 0.588 & \infty \end{pmatrix}$ MBPTS $\quad CON = 1$

(b)

Fig. 4. Resource models of SDR platform IV: (a) graphical and (b) mathematical representations.



(a)

Function model: $\boldsymbol{c} = (0.076, 0.289, 0.289, 1.44, 1.44, 2.35, 2.35, \ldots)$ MOPTS $\quad c_T = 13.675$ MOPTS

Dataflow model: $\boldsymbol{b} = \begin{pmatrix} 0 & 0.612 & 0.612 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0.578 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0.578 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0.145 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0.145 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$ MBPTS $\quad con = 0.12$

Stage model: $\boldsymbol{s} = (1, 2, 2, 3, 3, 4, 4, 4, 4, 5, 6, 6, 7, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17)$

(b)

Fig. 5. UMTS receiver's processing models: (a) graphical and (b) mathematical representations.

we suggest a general-purpose mapping algorithm. The t_w-mapping (Section 4.1) is such an algorithm. It can execute different cost functions and be dynamically adjusted to changes in the radio environment.

Section 4.2 introduces a cost function that manages the limited SDR computing resources. Section 4.3 analytically analyzes the t_w-mapping's computing complexity before presenting an SDR computing resource management example in Section 4.4. Section 5 provides more insights on the efficiency of this approach and the trade-off between the t_w-mapping's performance and computing complexity in simulated SDR environments.

4.1 The t_w-mapping

The t_w-mapping is a windowed dynamic programming algorithm, where w indicates the window size. Before proceeding with its description, we introduce some concepts first: The t_w-mapping diagram contains a matrix of $N \cdot M$ (row $\cdot$ column) t-nodes (Fig. 6). A t-node is identified as $(P_{k(l)}, f_i)$ and absorbs the mapping of SDR function f_i to processor $P_{k(l)}$. Any t-node at *step* i (column i in the t_w-mapping diagram) connects to all t-nodes at *step* $i+1$. The sequence of processors $[P_{k(0)}\ P_{k(1)} \ldots P_{k(w)}]_i$ identifies the w-path, a path of length w, which is associated with t-node $(P_{k(1)}, f_i)$. $P_{k(0)}$ is the w-path's origin processor at *step* $i-1$ and $P_{k(w)}$ is the destination processor at *step* $i+w-1$. Hence, $[P_1\ P_1\ P_N\ P_1]_2$ depicts the bold 3-path in Fig. 6. It is associated with t-node (P_1, f_2). Table 1 contains the most important variables and expressions that appear in the rest of this paper.

The t_w-mapping algorithm (Fig. 7) can be divided into three parts: *processing at step 1* (lines 1 and 2 in Fig. 7), *processing at steps* $i = 2, 3, \ldots, M-w+1$ (lines 3-18), and

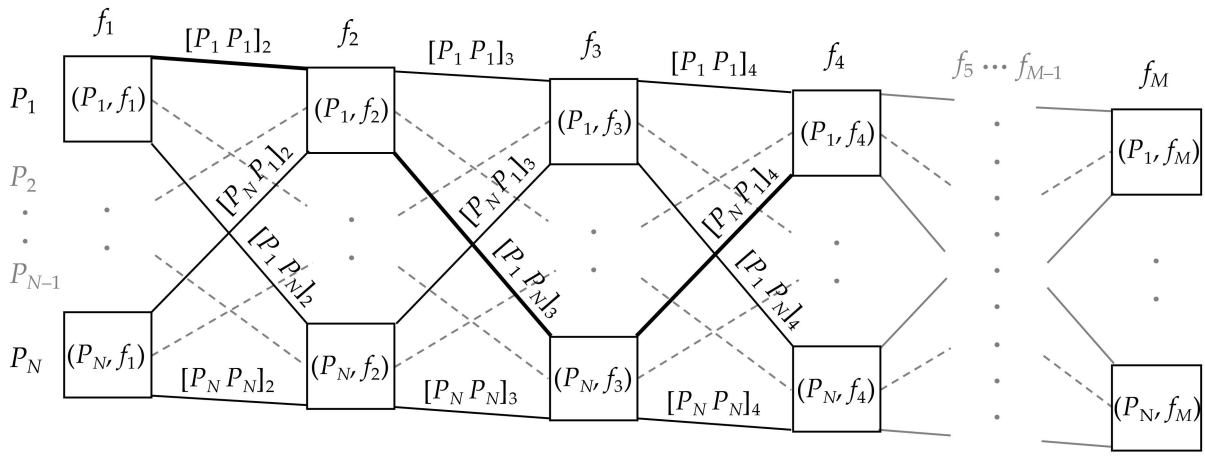


Fig. 6. The t_w-mapping diagram.

TABLE 1
Description and Ranges of Variables and Expressions

Variable or expression	Range[1]	Description
N	1, 2, …	number of processors
M	1, 2, …	number of SDR functions
w	1, 2, …, M–1	widow size
$k(l)$	1, 2, …, N; ($l \in 0, 1, \ldots, w$)	processor index $k(l)$ with its relative position l in the w-path
$P_{k(l)}$	$P_1, P_2, \ldots, P_N$	Processor
i, j	1, 2, …, M	step indexes (SDR function indexes)
f_i	$f_1, f_2, \ldots, f_M$	SDR function
$(P_{k(l)}, f_i)$		t-node describing the mapping of f_i to $P_{k(l)}$
$h = i + (l - 1)$	($l \neq 0$; $i \in 2, 3, \ldots, M–w+1$)	step index h; auxiliary variable that substitutes $i + (l - 1)$
$[P_{k(0)}\ P_{k(1)} \ldots P_{k(w)}]_i$	($i \in 2, 3, \ldots, M–w+1$)	w-path associated with t-node $(P_{k(1)}, f_i)$
$[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $i \in 2, 3, \ldots, M–w+1$)	edge between the t-nodes $(P_{k(l-1)}, f_{h-1})$ and $(P_{k(l)}, f_h)$
$\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $i \in 2, 3, \ldots, M–w+1$)	edge weight
$\mathrm{CT}[P_{k(0)}\ P_{k(1)} \ldots P_{k(w)}]_i$	($i \in 2, 3, \ldots, M–w+1$)	accumulated cost associated with w-path $[P_{k(0)}\ P_{k(1)} \ldots P_{k(w)}]_i$
$\mathrm{CT}(P_{k(l)}\ f_i)$	($l \in 1, 2$; $i \in 1, 2, \ldots, M–w+1$)	accumulated cost at t-node $(P_{k(l)}\ f_i)$
$P(f_j)$		processor allocated to SDR function f_j
$\boldsymbol{C}^{(k(l),\ i)}$		remaining processing capacities at t-node $(P_{k(l)}, f_i)$
$C_{k(l)}^{(k(l),\ i)}$		remaining processing capacity of processor $P_{k(l)}$ at t-node $(P_{k(l)}, f_i)$
$\boldsymbol{B}^{(k(l),\ i)}$		remaining bandwidths at t-node $(P_{k(l)}, f_i)$
$B_{P(f_j)P_{k(l)}}^{(k(l),\ i)}$		remaining bandwidth between processors $P(f_j)$ and $P_{k(l)}$ at t-node $(P_{k(l)}, f_i)$

[1] *Argument ranges are given in parentheses.*

postprocessing (lines 19 and 20). The t_w-mapping and thus its description do not assume a particular cost function.

4.1.1 Part I: Processing at Step 1

The first part of the algorithm addresses SDR function f_1. For any processor $P_{k(0)}$, the t_w-mapping ($w \geq 1$) premaps f_1 to $P_{k(0)}$ and stores the premapping cost $\mathrm{CT}(P_{k(0)}, f_1)$ at t-node $(P_{k(0)}, f_1)$ (lines 1 and 2). The term *premapping* indicates that the final SDR application mapping is not known until it has processed, or *premapped*, all of its SDR functions.

4.1.2 Part II: Processing at Steps $i = 2, 3, \ldots, M - w + 1$

The t_1-mapping analyzes the N ingoing edges, or 1-paths, of t-node $(P_{k(1)}, f_i)$. These are $[P_1\ P_{k(1)}]_i, [P_2\ P_{k(1)}]_i, \ldots$ and $[P_N\ P_{k(1)}]_i$ (line 5). Edge $[P_{k(0)}\ P_{k(1)}]_i$ is assigned the weight $\mathrm{WT}[P_{k(0)}\ P_{k(1)}]_i$ (line 6). This weight represents the cost of premapping SDR function f_i to processor $P_{k(1)}$ while considering the preceding decisions, which are provided by the edge's origin t-node $(P_{k(0)}, f_{i-1})$. The t_1-mapping computes the accumulated cost

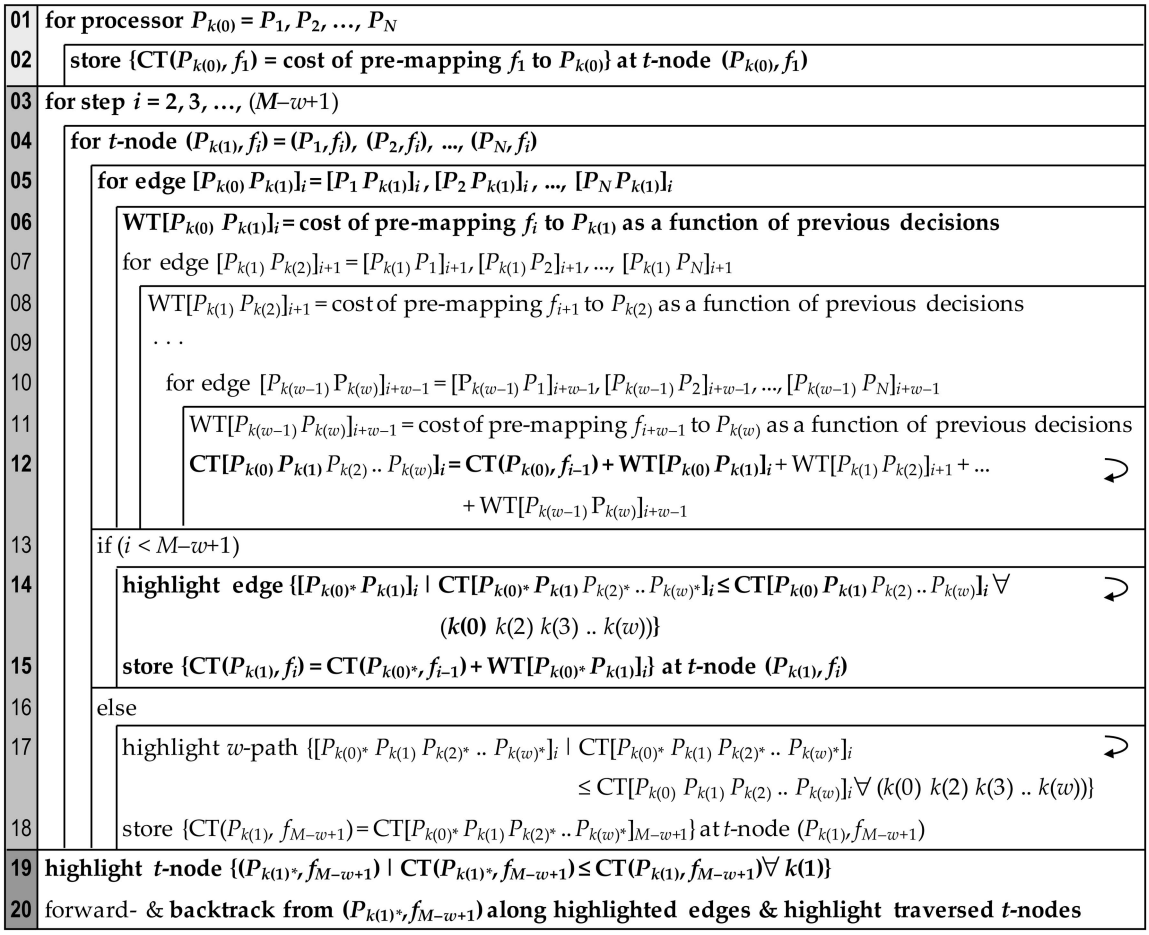


Fig. 7. t_w-mapping—pseudocode (the bold expressions describe the t_1-mapping).

$\text{CT}[P_{k(0)}\ P_{k(1)}]_i = \text{CT}(P_{k(0)}, f_{i-1}) + \text{WT}[P_{k(0)}\ P_{k(1)}]_i$ (line 12, bold) for each edge. Edge $[P_{k(0)^*}\ P_{k(1)}]_i$ is obtained from

$$k(0)^* = \underset{k(0)=1,2,\ldots,N}{\text{argmin}} \left\{ \text{CT}\left[P_{k(0)} P_{k(1)}\right]_i \right\} \tag{7}$$

and represents the premapping decision at t-node $(P_{k(1)}, f_i)$. (The function $\arg\min\{x\}$ returns the argument(s) that leads to the minimum value of x.) The algorithm finally highlights edge $[P_{k(0)^*}\ P_{k(1)}]_i$ (line 14, bold) and stores its accumulated cost at t-node $(P_{k(1)}, f_i)$ (line 15).

The t_w-mapping $(w > 1)$ analyzes the N^w w-paths that are associated with t-node $(P_{k(1)}, f_i)$. Any of these w-paths originates at a t-node at *step* $i-1$, runs through $(P_{k(1)}, f_i)$, and terminates at a t-node at *step* $i+w-1$. The algorithm computes the corresponding accumulated costs due to lines 5-12. It then solves

$$\{k(0)^*, k(2)^*, \ldots, k(w)^*\} = \underset{k(l)=1,2,\ldots,N; \forall l \neq 1}{\text{argmin}} \left\{ \text{CT}\left[P_{k(0)} P_{k(1)} P_{k(2)} \ldots P_{k(w)}\right]_i \right\}, \tag{8}$$

which returns the indices of the w-path that has the minimum accumulated cost. In the case where $i < M-w+1$, the t_w-mapping $(w > 1)$ highlights edge $[P_{k(0)^*}\ P_{k(1)}]_i$ and stores the corresponding accumulated cost at t-node $(P_{k(1)}, f_i)$ (lines 13-15). Otherwise, it highlights the entire w-path $[P_{k(0)^*}\ P_{k(1)}\ P_{k(2)^*} \ldots P_{k(w)^*}]_{M-w+1}$ and stores its accumulated cost at t-node $(P_{k(1)}, f_{M-w+1})$ (lines 16-18).

All N t-nodes at *step* i (line 4) can be processed in parallel. Once finished with their processing, the t_w-mapping $(w \ge 1)$ proceeds with *step* $i+1$ in the case where $i < M-w+1$ and with part III otherwise (line 3).

4.1.3 Part III: Postprocessing

Part III of the algorithm postprocesses the premapping decisions of parts I and II. The t_w-mapping $(w \ge 1)$ first highlights the t-node at *step* $M-w+1$ that holds the minimum cost (line 19). Starting at this t-node, it then backtracks (forward and backtracks in the case where $w > 1$) the t_w-mapping diagram along the highlighted edges while highlighting all t-nodes that are traversed (line 20). This results in M highlighted t-nodes which specify the mapping proposal due to the particular problem, cost function, and window size.

Sections 4.1.2 and 4.1.3 indicate that w controls the level of locality (local versus global scope) of the premapping decisions: The t_1-mapping is based on local decisions. It however maintains N premappings at each *step*, one per t-node, which may result in N (partially) different mapping options (bold expressions in Fig. 7).

At the other extreme, the t_{M-1}-mapping examines all N^M possible mappings of M functions to N processors. More precisely, it computes the accumulated $(M-1)$-path costs of all N^{M-1} different $(M-1)$-paths that traverse

t-node $(P_{k(1)}, f_2)$ (lines 5-12) and selects the one of minimum cost (lines 17 and 18). It does so for all N t-nodes at *step* 2 (line 4). The algorithm then highlights t-node $(P_{k(1)^*}, f_2)$ (line 19) and forwards and backtracks the t_w-mapping diagram to obtain the final mapping (line 20). It finds an optimal solution for the given problem and cost function.

Parameter w also controls the trade-off between computing efficiency and mapping performance: The higher the window size, the higher the algorithm's complexity (Section 4.3), but the better the mapping results (Section 5).

4.2 Cost Function Proposal

We propose a cost function that properly manages the limited computing resources of SDR platforms under hard real-time conditions, where the computing resources constrain the SDR application mapping. $\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$ therefore defines the cost function as a superposition between the computation and the communication costs:

$$\mathrm{WT}\big[P_{k(l-1)}\ P_{k(l)}\big]_h = \mathrm{WT}_{comp}\big[P_{k(l-1)}\ P_{k(l)}\big]_h + \mathrm{WT}_{comm}\big[P_{k(l-1)}\ P_{k(l)}\big]_h, \tag{9a}$$

$$\mathrm{WT}_{comp}\big[P_{k(l-1)}\ P_{k(l)}\big]_h = \begin{cases} c_h / C_{k(l)}^{(k(l),h)}, & \text{if } c_h / C_{k(l)}^{(k(l),h)} \le 1, \\ \infty, & \text{otherwise,} \end{cases} \tag{9b}$$

$$\mathrm{WT}_{comm}\big[P_{k(l-1)}\ P_{k(l)}\big]_h = \begin{cases} \sum_{j=1}^{h-1} b_{jh} / B_{P(f_j)P_{k(l)}}^{(k(l),h)}, & \text{if } b_{jh} / B_{P(f_j)P_{k(l)}}^{(k(l),h)} \le 1 \forall j, \\ \infty, & \text{otherwise.} \end{cases} \tag{9c}$$

Any t-node $(P_{k(l)}, f_i)$ stores the remaining processing powers $\boldsymbol{C^{(k(l),i)}}$ and bandwidths $\boldsymbol{B^{(k(l),i)}}$ as a function of the preceding premapping decisions. The cost function first computes the premapping costs at t-nodes $(P_1, f_1), (P_2, f_1), \ldots,$ and (P_N, f_1) using the right-hand side of (9b). For each t-node $(P_{k(0)}, f_1)$, the processing requirement c_1 of SDR function f_1 is then subtracted from the corresponding initial processing power $C_{k(0)}$. This updates the remaining processing powers at all t-nodes at *step* 1.

Before processing edge $[P_{k(l-1)}\ P_{k(l)}]_h$, $\boldsymbol{C^{(k(l),h)}}$ and $\boldsymbol{B^{(k(l),h)}}$ are initialized with $\boldsymbol{C^{(k(l-1),h-1)}}$ and $\boldsymbol{B^{(k(l-1),h-1)}}$. The algorithm then calculates $\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$ before updating $\boldsymbol{C^{(k(l),h)}}$ and $\boldsymbol{B^{(k(l),h)}}$. In particular, c_h is subtracted from $C_{k(l)}^{(k(l),h)}$ after computing $\mathrm{WT}_{comp}[P_{k(l-1)}\ P_{k(l)}]_h$ (9b). $\boldsymbol{B^{(k(l),h)}}$, on the other hand, is dynamically updated, subtracting any required bandwidth b_{jh} from the corresponding entry in $\boldsymbol{B^{(k(l),h)}}$ just after adding $b_{jh}/\{\cdot\}$ to $\mathrm{WT}_{comm}[P_{k(l-1)}\ P_{k(l)}]_h$ (9c).

4.3 Complexity Analysis

4.3.1 General Formulation

The computing complexity of the t_w-mapping depends on the applied cost function. For a general formulation, we assume that the complexity of calculating the cost of premapping f_i to $P_{k(l)}$ is constant and not a function of i or $k(l)$. Let this complexity be the *complexity of the cost function* (*ccf*). The computing complexity at t-node $(P_{k(l)}, f_i)$, $i \in 2, 3, \ldots, M-w+1$, can then be given as the geometric sum

$$(N + N^2 + \ldots + N^w) \cdot ccf = N \cdot \frac{N^w - 1}{N - 1} \cdot ccf. \tag{10}$$

It indicates the computing effort associated with lines 5-11 in the pseudocode of Fig. 7. There are N t-nodes per *step* (line 4) and $M - w$ steps in total (line 3). Thus,

$$\text{complexity}(t_w\text{-mapping}) \approx (M - w) \cdot N^2 \cdot \frac{N^w - 1}{N - 1} \cdot ccf. \tag{11}$$

Equation (11) takes into account the bulk processing of the t_w-mapping, neglecting the complexity that is associated with the operations of parts I and III (lines 1-2 and 19-20 in Fig. 7) and the add-compare-store operations (lines 12-18). Assuming $ccf = 1$, the t_w-mapping's complexity order becomes

$$\text{complexity-order}(t_w\text{-mapping}) = O(M \cdot N^{w+1}). \tag{12}$$

Equation (12) indicates that the t_w-mapping is not computing efficient in the case where the number of processors N is high. We therefore suggest (dynamically) dividing a large array of processors into smaller clusters of N' processors and applying the algorithm on each cluster. Without loss of generality, this paper assumes a small $N = N'$.

4.3.2 Cost Function Specific Formulation

The computing complexity of cost function (9) is not constant throughout the mapping process. Assuming no code optimizations, the number of MACs characterizes the complexity of part II of the t_w-mapping under cost function (9) as

$$\text{complexity}(t_w\text{-mapping}_{(9)}) \approx N^2 \cdot \sum_{k=1}^{w} N^{k-1} \cdot \frac{(M-w)(M-w+2k+1)}{2}. \tag{13}$$

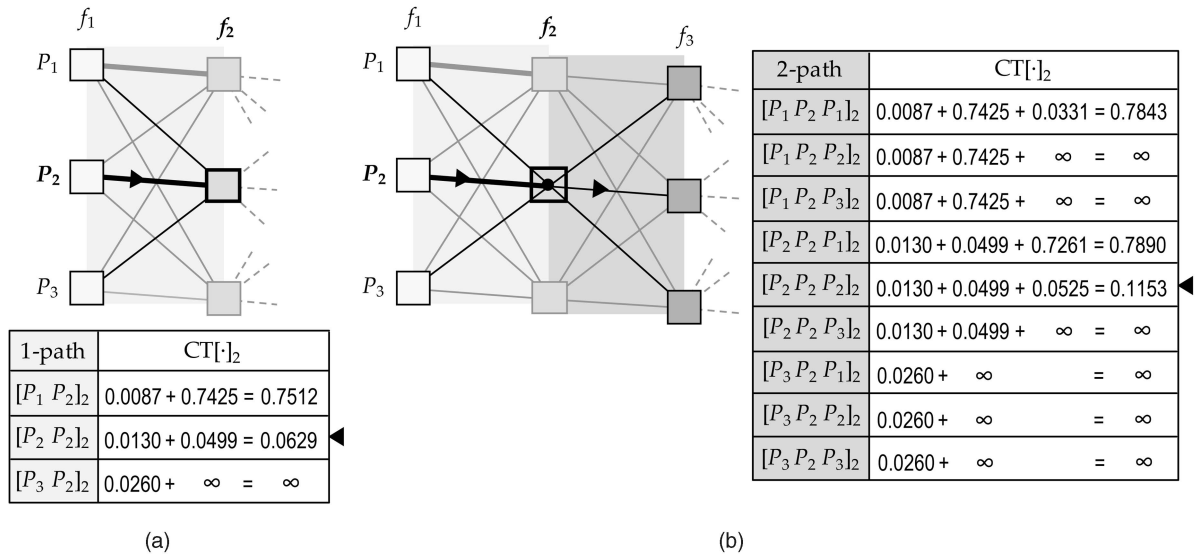
We equate the right-hand side of (11) with the right-hand side of (13) and obtain

$$\frac{N^w - 1}{N - 1} \cdot ccf = \sum_{k=1}^{w} N^{k-1} \cdot \frac{M - w + 2k + 1}{2}. \tag{14}$$

We can then derive the following: If we substitute *ccf* for $(M + w + 1)/2$, (11) becomes an upper bound formulation for complexity $(t_w\text{-mapping}_{(9)})$; it is exact for $w = 1$ and a very close approximation otherwise. For $N = 3$, $M = 24$, and $w = 3$, for instance, this upper bound deviates from the complexity due to (13) by 2.8 percent.

4.4 Resource Management Examples

Fig. 8 illustrates part II of the t_1- and t_2-mapping algorithms. It shows the processing at t-node (P_2, f_2) while mapping the SDR application model of Fig. 5 to the SDR platform model of Fig. 4 using cost function (9). Previously, and corresponding to part I of the t_w-mapping, the costs of premapping f_1 to each one of the three processors were obtained as 0.0087, 0.013, and 0.026 and stored at t-nodes (P_1, f_1), (P_2, f_1), and (P_3, f_1). These t-nodes also store the



1-path	$CT[\cdot]_2$
$[P_1\ P_2]_2$	0.0087 + 0.7425 = 0.7512
$[P_2\ P_2]_2$	0.0130 + 0.0499 = 0.0629 ◀
$[P_3\ P_2]_2$	0.0260 + ∞ = ∞

(a)

2-path	$CT[\cdot]_2$
$[P_1\ P_2\ P_1]_2$	0.0087 + 0.7425 + 0.0331 = 0.7843
$[P_1\ P_2\ P_2]_2$	0.0087 + 0.7425 + ∞ = ∞
$[P_1\ P_2\ P_3]_2$	0.0087 + 0.7425 + ∞ = ∞
$[P_2\ P_2\ P_1]_2$	0.0130 + 0.0499 + 0.7261 = 0.7890
$[P_2\ P_2\ P_2]_2$	0.0130 + 0.0499 + 0.0525 = 0.1153 ◀
$[P_2\ P_2\ P_3]_2$	0.0130 + 0.0499 + ∞ = ∞
$[P_3\ P_2\ P_1]_2$	0.0260 + ∞ = ∞
$[P_3\ P_2\ P_2]_2$	0.0260 + ∞ = ∞
$[P_3\ P_2\ P_3]_2$	0.0260 + ∞ = ∞

(b)

Fig. 8. Part II of the t_w-mapping examples: Processing at t-node (P_2, f_2) for (a) $w = 1$ and (b) $w = 2$.

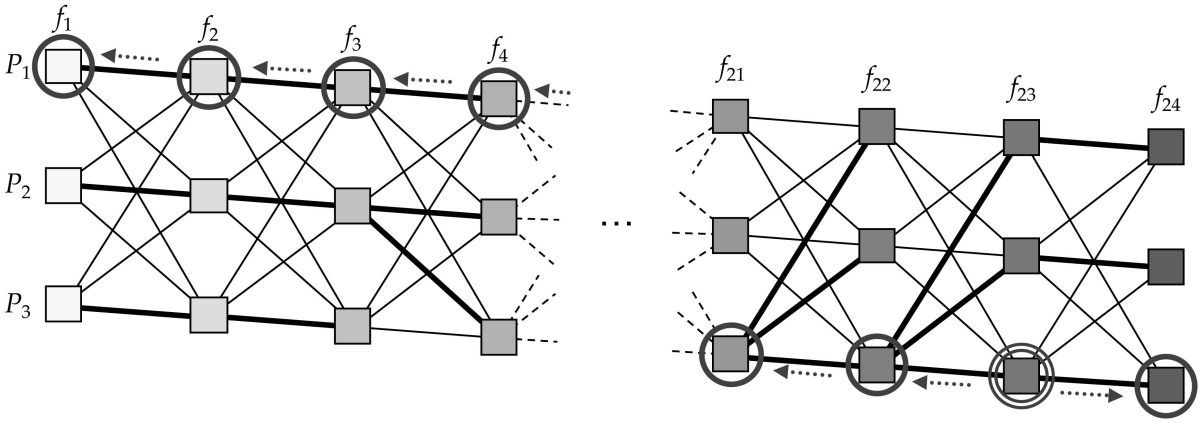


Fig. 9. Part III of the t_2-mapping example.

remaining processing powers $\boldsymbol{C}^{(\boldsymbol{k}(\boldsymbol{l}),\mathbf{1})}$ and bandwidths $\boldsymbol{B}^{(\boldsymbol{k}(\boldsymbol{l}),\mathbf{1})} = \boldsymbol{B}$.

As explained in Section 4.1.2, the t_w-mapping evaluates N^w w-paths. The tables in Fig. 8 contain the accumulated w-path costs $\mathrm{CT}[\cdot]_2$ in the format of line 12 in Fig. 7. Both algorithms highlight edge $[P_2\ P_2]_2$, resulting from the minimum-cost 1 and 2-paths $[P_2\ P_2]_2$ and $[P_2\ P_2\ P_2]_2$, respectively. The accumulated cost of 0.0629 and the remaining processing and bandwidth resources, $\boldsymbol{C}^{(\mathbf{2},\mathbf{2})} = (8.82, 5.51, 2.94)$ and $\boldsymbol{B}^{(\mathbf{2},\mathbf{2})} = \boldsymbol{B}$, are finally stored at t-node (P_2, f_2).

Due to Section 4.1.3, we obtain the final t_w-mapping by traversing the t-nodes along the highlighted edges starting at the minimum-cost t-node at step $(M - w + 1)$. Fig. 9 indicates that this is t-node (P_3, f_{23}) for this example and $w = 2$, with the costs stored at t-nodes (P_1, f_{23}), (P_2, f_{23}), and (P_3, f_{23}) being 4.3706, 4.3633, and 4.3600, respectively. Then, $f_1, f_2, \ldots, f_6, f_8, f_{10}$, and f_{11} are mapped to P_1, f_7 and f_9 to P_2, and $f_{12}, f_{13}, \ldots, f_{24}$ to P_3 (Fig. 9). Due to space limitations, we only mention that the t_1-mapping solution differs from this mapping proposal and has a slightly higher cost of 4.3891.

5 Simulations

This section attempts to evaluate the entire framework. After introducing the reference mapping algorithms in Section 5.1, we present the simulation results of two SDR scenarios in Sections 5.2 and 5.3.

5.1 Reference Mapping Algorithms

It is not feasible to adapt previously introduced algorithms to the SDR computing resource management context, as described in this paper, and to evaluate their performance within our framework. Because using these algorithms is impractical [18], we implement a baseline approach. Its simplicity makes it applicable to realistic scenarios, which are often very complex. Each baseline result is complemented with the optimal solution, obtained from an exhaustive search.

The greedy or g-mapping is a local mapping approach that does not require any postprocessing. The algorithm first maps SDR function f_1 to

$$P_1^* = \left\{ P_{k(0)^*} | CT\left(P_{k(0)^*}, f_1\right) \leq CT\left(P_{k(0)}, f_1\right) \forall k(0) \right\}, \quad (15)$$

which represents the processor associated with the minimum mapping cost.

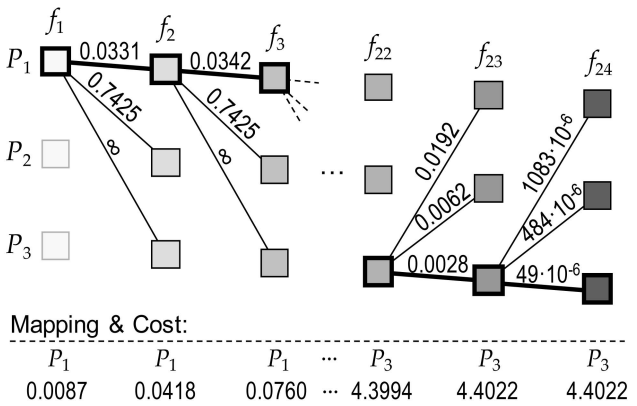


Fig. 10. The g-mapping example.

In the second part of the g-mapping process, the algorithm analyzes the outgoing edges from *the* active t-node at *step* i $(i \in 1, 2, \ldots, M-1)$, selects the one of minimum mapping cost, and passes the accumulated cost to the resulting active t-node at *step* $i+1$. The g-mapping thus maintains only one active t-node per step, making it one order of magnitude—or N times—less complex than the t_1-mapping.

Fig. 10 indicates part of the g-mapping solution to the problem of Section 4.4. The complete solution consists of mapping $f_1, f_2, \ldots, f_6, f_8, f_{10}, f_{11}, f_{13}, f_{15}$, and f_{16} to P_1, f_7 and f_9 to P_2, and $f_{12}, f_{14}, f_{17}, f_{18}, \ldots$, and f_{24} to P_3. Its cost is 4.4022 (Fig. 10).

5.2 SDR Scenario I: UMTS Task Graph

A future SDR platform will be subject to the dynamic reconfiguration of the different layers in the protocol stack that define the radio functionality. Hence, the amount of available computing resources may significantly differ from one configuration to another. To simulate this, we propose scaling a platform's computing resources as follows: sf_C scales the processing capacities and sf_B the interprocessor communication bandwidths. We obtain sf_C and sf_B from

$$(c\text{-}load, b\text{-}load) = \left(\frac{c_{\mathrm{T}}}{sf_C \cdot C_{\mathrm{T}}}, \frac{b_{\mathrm{T}}}{sf_B \cdot B_{\mathrm{T}}}\right), \tag{16}$$

where $c\text{-}load \in 0.2, 0.35, \ldots, 0.95$ and $b\text{-}load \in 0.75, 1.25, \ldots, 3$. $c\text{-}load$ specifies the relation between the total processing requirement c_{T} of the SDR application and the total processing power $(sf_C \cdot C_{\mathrm{T}})$ of an SDR platform. $b\text{-}load$ relates the total bandwidth demand b_{T} of the SDR application to a platform's bandwidth capacity $(sf_B \cdot B_{\mathrm{T}})$.

The SDR application corresponds to the UMTS downlink receiver of Fig. 3; Fig. 5 shows its modeling. The scenario considers the four SDR platforms of Fig. 2. Fig. 4 illustrates the system models of SDR platform IV.

Fig. 11 contains the simulation results. A square in each subfigure represents a particular $(c\text{-}load, b\text{-}load)$ tuple, which specifies the mapping problem. Its shading indicates whether the corresponding g- or t_w-mapping result is optimal, suboptimal, or infeasible. A mapping is optimal for a particular problem and cost function if there exists no other mapping with an inferior mapping cost. An optimal, suboptimal, or infeasible result indicates that the cost of the optimal mapping is x percent of the algorithm's mapping cost, where $x = 100$, $0 < x < 100$, and $x = 0$ describe the three cases. A cross marks a situation where none of the 3^{24} different mappings are feasible. We call this an *impossible mapping situation*.

From Fig. 11, we derive a platform's flexibility: SDR platform III is the most flexible because of the relatively few impossible mapping situations. Platforms I and II are much less flexible because more processing or bandwidth resources are needed to feasibly solve many of the given mapping problems.

We find that the t_w-mapping $(w = 1, 2, 3)$ is more robust against $c\text{-}load$ and $b\text{-}load$ variations than the g-mapping. The most critical difference between the algorithms is observed for SDR platform II: While the t_w-mapping achieves feasible results for all possible mapping situations, the g-mapping fails in nine cases (black squares in Fig. 11b). Also, the number of optimal t_w-mapping results is considerably higher than the number of optimal g-mappings (white squares in Fig. 11). We conclude that the t_1-mapping feasibly solves all but one solvable mapping problem, whereas the g-mapping fails in 10 cases. The t_4- and t_5-mapping algorithms feasibly map the resource situation $(c\text{-}load, b\text{-}load) = (0.95, 1.25)$. Their results are not shown due to space limitations.

Neither the t_4- nor the t_5-mapping algorithms achieve optimal results for all of the simulated resource conditions of architecture II. The t_2-mapping, on the other hand, achieves optimal results for (almost) all conditions of architectures I, III, and IV. The great complexity reduction of the t_2-mapping with respect to the optimal t_{23}-mapping is very important in dynamic reconfiguration scenarios. For example, the frequent initializations and terminations of sessions we currently find in BSs require a computing efficient mapping approach that is able to provide the desired performance. A feasible solution may be sufficient here, whereas an optimal solution could be desirable in another scenario. This leads to the following two conclusions:

1. There is a relation between the platform architecture and the t_w-mapping performance.
2. The t_w-mapping with small w is able to solve any possible mapping situation of this scenario.

The execution times of the g-, t_1-, t_2-, t_3-, t_4-, and t_5-mapping implementations are approximately 10, 50, 150, 430, 1,200, 3,500 μs on a 2.4 GHz GPP. The relations between these execution times are 1:5:15:43:120:350, which is in line with the corresponding relations between the approximate algorithms' complexities due to Section 5.1 and (13), 897:2,691:10,494:33,453:101,160:300,447 or 1:3:12:37:113:335.

5.3 SDR Scenario II: Random Task Graphs

An SDR platform will be dynamically reconfigured from one RAT or RAT implementation to another. This dynamism may even affect a single user session. An infeasible reconfiguration (infeasible mapping) would then mean a lost session.

The scenario considers four MTs; Fig. 2 shows their computing architectures and resources. Each terminal is reconfigured 50,000 times. A reconfiguration of an SDR platform consists of the demapping of the old SDR application and the mapping of the new one (total reconfiguration). We randomly generate 50,000 DAGs based of the following parameters:

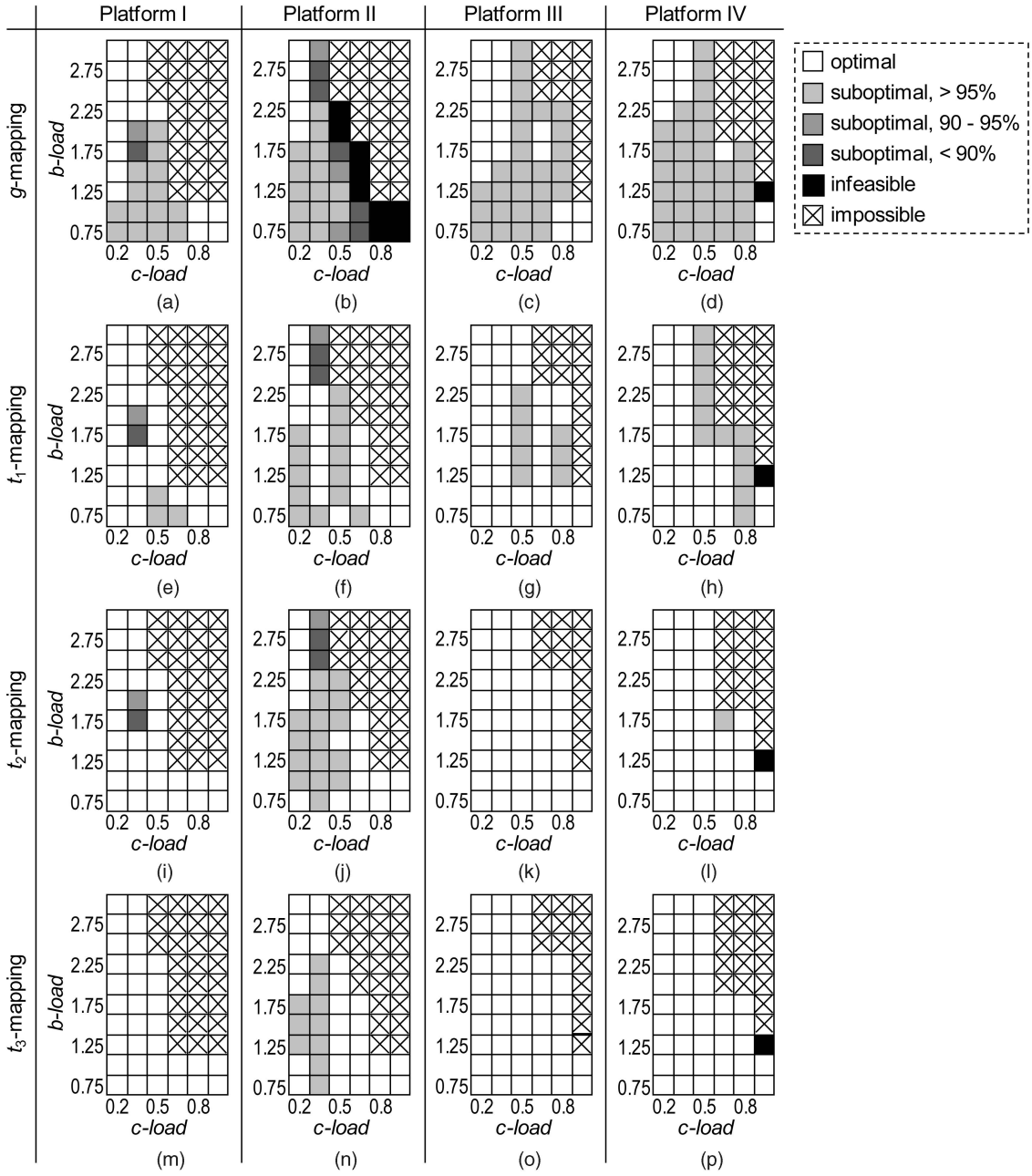


Fig. 11. (a)-(d) g-, (e)-(h) t_1-, (i)-(l) t_2-, and (m)-(p) t_3-mapping results for SDR platforms I-IV.

- the number of nodes per DAG is $M = 18$,
- the connectivity of a DAG is $con = 0.15$,
- the processing demands are uniformly distributed in $[1, 2, \ldots, 2{,}500]$ MOPS, and
- the bandwidth demands are uniformly distributed in $[1, 2, \ldots, 500]$ Mbps.

These DAGs should be understood as different SDR applications that represent different RATs or RAT variations; 99.3 percent of them require more than 50 percent of a platform's total processing resources and 75 percent in the mean. The connectivity indicates the probability of connecting, or drawing an arc between, any two nodes in a DAG. We allow disconnected graphs (a graph consisting of two or more connected subgraphs [36]), which model parallel subfunctions. Irrespective of the particular DAG, we specify the time slot duration t_{ts} as $0.5 \cdot 10^{-3}$ SPTS. Then, the latency of any of these SDR applications will be at most $18 \cdot 0.5$ ms $= 9$ ms (6).

The performance metric is the percentage of infeasible mappings. Fig. 12 illustrates the outcomes. (Since we discard those 0.7 percent of the DAGs that require more processing resources than available, the exhaustive search for a feasible mapping would lead to a negligible number of infeasible mappings.) We observe that, for any platform, the number of unfeasibly t_1-mapped DAGs is about half the number of unfeasibly g-mapped DAGs.

Fig. 12 furthermore shows that the higher the window size the better the result. If we set the limit to 7.5 percent infeasible mappings, we can say that the t_1-mapping is appropriate for SDR platform I, the t_2-mapping for platforms II and IV, whereas a window size of 3 is necessary for

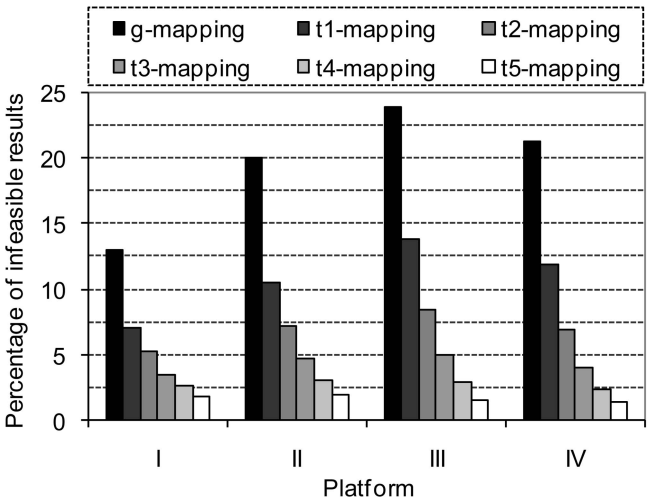


Fig. 12. Percentage of infeasible g-, t_1-, t_2-, t_3-, t_4-, and t_5-mappings for SDR platforms I-IV.

platform III. If the limit is 5 percent, the t_3-mapping is appropriate for all platforms, whereas a limit of 2.5 percent requires at least the t_4-mapping in the case of platform IV and the t_5-mapping otherwise. The g-mapping, which cannot feasibly map 13-24 percent of the DAGs, is far from suitable for any reasonable performance limit. These results qualify the algorithms' performances. Despite the more general simulation setup, we can make similar conclusions as for SDR scenario I.

6 Conclusions

The constant evolution of radio access technologies, (multimedia) services, and computing devices, among others, make the wireless environment highly dynamic and unpredictable. SDR facilitates the reconfiguration of radio equipment, introducing flexibility to wireless communications; hence the need for computing resource management in SDR and a framework that can deal with real-time constraints and changing QoS demands of wireless systems.

Our proposal consists of two parts: an *SDR computing system modeling* that facilitates the *SDR computing resource management*. The system modeling accounts for the limited computing resources of SDR platforms and the real-time requirements of SDR applications. The resource management features a mapping algorithm and a cost function: The t_w-mapping is a windowed dynamic programming approach that is apt for many cost functions or mapping policies. The cost function proposal dynamically manages an SDR platform's computing resources to satisfy an SDR application's computing and, thus, service requirements.

We have simulated two relevant SDR scenarios. The results have demonstrated the appropriateness of the entire framework. Moreover, we could observe that the t_w-mapping, as opposed to the g-mapping, achieves feasible and even optimal results for window sizes as small as 1, 2, or 3. These scenarios have also revealed that the t_w-mapping is fast enough for real implementations, predictable in terms of computing costs and mapping results, and suitable for different waveforms and platforms.

In future work, we will introduce other cost functions to manage additional computing or computing-related resources, such as memory or energy. We will simulate more SDR scenarios and analyze how to adjust the window size w to the particular problem. We will study the impact of hardware abstractions on the performance of SDR applications before addressing the implementation of our framework within the PHAL execution environment. Finally, the SDR computing resource management needs to be coordinated with the radio resource management.

Acknowledgments

This work was supported by the DURSI of the Catalonian Government and the Spanish National Science Council (CYCIT) under Grant TEC2006-09109, which is partially financed from the European Community through the FEDER program.

References

[1] J. Mitola, "The Software Radio Architecture," *IEEE Comm. Magazine,* vol. 33, no. 5, pp. 26-38, May 1995.
[2] E. Buracchini, "The Software Radio Concept," *IEEE Comm. Magazine,* vol. 38, no. 9, pp. 138-143, Sept. 2000.
[3] W.H.W. Tuttlebee, "Software-Defined Radio: Facets of a Developing Technology," *IEEE Personal Comm.,* vol. 6, no. 2, pp. 38-44, Apr. 1999.
[4] "Software Radios," *IEEE J. Selected Areas in Comm.,* J. Mitola III, V. Bose, B.M. Leiner, T. Turletti, and D. Tennenhouse, eds., vol. 17, no. 4, pp. 509-747, Apr. 1999.
[5] J. Mitola and Z. Zvonar, *Software Radio Technologies: Selected Readings.* IEEE Press, 2001.
[6] M. Dillinger, K. Madani, and N. Alonistioti, *Software Defined Radio: Architectures, Systems and Functions.* John Wiley & Sons, 2003.
[7] SDR Forum Website, www.sdrforum.org, 2008.
[8] A. Duller, G. Panesar, and D. Towner, "Parallel Processing—The picoChip Way!" *Proc. Communicating Process Architectures,* pp. 299-312, Sept. 2003.
[9] *Networks on Chip,* A. Jantsch and H. Tenhunen, eds. Kluwer Academic, 2003.
[10] T. Kogel and H. Meyr, "Heterogeneous MP-SoC—The Solution to Energy-Efficient Signal Processing," *Proc. 41st ACM/IEEE Design Automation Conf.,* pp. 686-691, June 2004.
[11] J. Villasenor and B. Hutchings, "The Flexibility of Configurable Computing," *IEEE Signal Processing Magazine,* pp. 67-84, Sept. 1998.
[12] A. Gathener et al., "DSP-Based Architectures for Mobile Communications: Past, Present, and Future," *IEEE Comm. Magazine,* vol. 38, pp. 84-90, Jan. 2000.
[13] M. Cummings and S. Haruyama, "FPGA in the Software Radio," *IEEE Comm. Magazine,* vol. 37, pp. 108-112, Feb. 1999.
[14] S.-Y. Lee and J.K. Aggarwal, "A Mapping Strategy for Parallel Processing," *IEEE Trans. Computers,* vol. 36, no. 4, pp. 433-442, Apr. 1987.
[15] V. Chaudhary and J.K. Aggarwal, "A Generalized Scheme for Mapping Parallel Algorithms," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 3, pp. 328-346, Mar. 1993.
[16] M. Tan et al., "Minimizing the Application Execution Time through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, pp. 857-871, Aug. 1997.
[17] I. Ahmad and Y.-K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 9, pp. 872-892, Sept. 1998.
[18] A.H. Alhusaini, V.K. Prasanna, and C.S. Raghavendra, "A Framework for Mapping with Resource Co-Allocation in Heterogeneous Computing Systems," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 273-286, May 2000.
[19] A.H. Alhusaini, C.S. Raghavendra, and V.K. Prasanna, "Run-Time Adaptation for Grid Environments," *Proc. 15th IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 864-874, Apr. 2001.
[20] K. Bondalapati, "Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures," PhD dissertation, Univ. of Southern California, Aug. 2001.

[21] A.-R. Rhiemeier and F. Jondral, "Mathematical Modeling of the Software Radio Design Problem," *IEICE Trans. Comm.,* vol. E86-B, no. 12, pp. 3456-3467, Dec. 2003.
[22] H. Topcuoglu, S. Hariri, and M.-Y. Wou, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 260-274, Mar. 2002.
[23] R. Bajaj and D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 2, pp. 107-118, Feb. 2004.
[24] A. Dogan and F. Özgüner, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 308-323, Mar. 2002.
[25] D.-T. Peng, K.G. Shin, and T.F. Abdelzaher, "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems," *IEEE Trans. Software Eng.,* vol. 23, no. 12, pp. 745-758, Dec. 1997.
[26] C.-J. Hou and K.G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," *IEEE Trans. Computers,* vol. 46, no. 12, pp. 1338-1356, Dec. 1997.
[27] K. Ramamritham, J.A. Stankovic, and P.-F. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 1, no. 2, pp. 184-194, Apr. 1990.
[28] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. Computers,* vol. 38, no. 8, pp. 1110-1123, Aug. 1989.
[29] A.W. Krings and M.H. Azadmanesh, "Resource Reclaiming in Hard Real-Time Systems with Static and Dynamic Workloads," *Proc. 30th IEEE Hawaii Int'l Conf. System Sciences,* pp. 116-625, Jan. 1997.
[30] O. Moreira, J.-D. Mol, M. Beckooij, and J. van Meerbergen, "Multiprocessor Resource Allocation for Hard-Real-Time Streaming with a Dynamic Job-Mix," *Proc. 11th IEEE Real Time and Embedded Technology and Applications Symp.,* pp. 332-341, Mar. 2005.
[31] S. Stuijk, T. Basten, M.C.W. Geilen, and H. Corporaal, "Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs," *Proc. 44th ACM/IEEE Design Automation Conf.,* pp. 777-782, June 2007.
[32] J.-K. Kim et al., "Collective Value of QoS: A Performance Measure Framework for Distributed Heterogeneous Networks," *Proc. 15th IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 810-823, Apr. 2001.
[33] S. Gertphol, Y. Yu, A. Alhusaini, and V.K. Prasanna, "A Metric and Mixed-Integer-Programming-Based Approach for Resource Allocation in Dynamic Real-Time Systems," *Proc. 16th IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 993-1000, Apr. 2002.
[34] "Algorithm Design and Scheduling Techniques (Realistic Platform Models) for Heterogeneous Clusters" *IEEE Trans. Parallel and Distributed Systems,* special section, H. Casanova, Y. Robert, and H. J. Siegel, eds., vol. 17, no. 2, pp. 97-190, Feb. 2006.
[35] X. Reves, A. Gelonch, V. Marojevic, and R. Ferrus, "Software Radios: Unifying the Reconfiguration Process over Heterogeneous Platforms," *EURASIP J. Applied Signal Processing,* vol. 2005, no. 16, pp. 2626-2640, Sept. 2005.
[36] D.F. Robinson and L.R. Foulds, *Digraphs: Theory and Techniques.* Gordon and Breach Science, 1980.
[37] R. Tanner and J. Woodard, *WCDMA—Requirements and Practical Design.* John Wiley & Sons, 2004.
[38] Technical Specification Group Radio Access Network (3GPP), "TS 25.212 V6.4.0—Multiplexing and Channel Coding (FDD)," www.3gpp.org, Mar. 2005.
[39] T. Faber and M. Schönle, "DSP-Platform Target Report," SLATS Consortium, Project no. 27016, Deliverable D23, Dec. 1999.
[40] Texas Instruments (TI) Website, www.ti.com, 2008.

Vuk Marojevic received the Dipl-Ing degree in electrical engineering from the University of Hannover in 2003. He is currently working toward the PhD degree in communications in the Department of Signal Theory and Communications at the Polytechnic University of Catalonia (UPC). He spent the Fall Term 2007 at Wireless@Virginia Tech. His research efforts are on SDR and cognitive radio architectures and algorithms for an efficient and flexible (computing) resource management. He is a student member of the IEEE.

Xavier Revés Ballesté received the telecommunication engineering and PhD degrees from the Universitat Politècnica de Catalunya (UPC), Barcelona, in 1998 and 2001, respectively. From 2001 to 2006, he was with the Department of Signal Theory and Communications at UPC as an assistant professor. His PhD research is focused on radio communications, with special emphasis on DSP platforms, including hardware and software architectures, software radio technologies, and analog/digital transceiver front-end architectures. Until 2006, he was participating in several projects of technological transfer to industry, including the European FUSE Program. He has also been involved in the European Framework Program projects ARROWS, EVEREST, and E2R. In 2006, he joined Gigle Semiconductor, a fables semiconductor company, where he designs and develops DSP algorithms for high-speed communications. He is a member of the IEEE.

Antoni Gelonch received the telecommunication engineering and PhD degrees from the Universitat Politècnica de Catalunya (UPC), Barcelona, in 1991 and 1997, respectively. In 1992, he joined UPC, where he became an associate professor in 1997. Since 1992, he has been mainly involved in the analysis and development of mobile systems for digital radio communications. His research interests include cellular and personal communication systems, multipath transceiver designs, mobility and radio resources management, and software radio techniques. He has participated in several research projects funded by public and private organizations.